Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference

Peter M. Fischer, University of Freiburg, Germany
Gustavo Alonso, ETH Zurich, Switzerland
Marcelo Arenas, Pontificia Universidad Catolica de Chile, Chile
Floris Geerts, University of Antwerp, Belgium

March 27th, 2015

Contents

Message from the Chairs

It is our pleasure to present to you, on behalf of the entire conference organizing committee and the workshop organizers, the proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference, held on March 27, 2015, in Brussels, Belgium.

The International Conference on Extending Database Technology (EDBT) and the International Conference on Database Theory (ICDT) are two prestigious venues for the exchange of the latest research results in data management and the theoretical foundations of database systems. While having the same overarching goal of presenting cutting-edge results, ideas, techniques, and theoretical advances in databases, the workshops of the EDBT/ICDT joint conference are separately tasked by focusing on emerging topics that complement the areas covered by the main technical program.

This year, our program includes workshops focusing on seven exciting topics:

- *Algorithms for MapReduce and Beyond (BeyondMR)*, aiming to explore algorithms and models computational for the specialized systems that have recently been developed to serve the needs of "big data",
- *Data (Co-)Processing on Heterogeneous Hardware (DAPHNE)*, investigating challenges and opportunities for data processing on existing and upcoming heterogeneous hardware architectures, ranging from multi-core CPUs to massively parallel accelerators, heterogeneous mobile phone processors to FPGAs,
- *Energy Data Management (EnDM)*, focusing on conceptual and system architecture issues related to the management of very large-scale data sets specifically in the context of the energy domain,
- *Event Processing, Forecasting and Decision-Making in the Big Data Era (EPForDM)*, bringing together computer scientists with interests in the fields of event processing, event forecasting and event-driven decision-making to present recent innovations, find topics of common interest and stimulate further development of new approaches to make sense of Big Data.
- *Querying Graph Structured Data (GraphQ)*, which aims to encourage discussions on how to efficiently and effectively support graph queries in different application domains and seeks to provide the opportunity for cross-fertilization among teams working on graph-structured data, with a particular focus on the querying issues.
- *Linked Web Data Management (LWDM)*, aiming at stimulating participants to discuss about data management issues related to the Linked Data and the relationships with other Semantic Web technologies, and proposes a glance at new issues,
- *Privacy and Anonymity in the Information Society (PAIS)*, which provides a platform for researchers and practitioners from computer science and other fields that are interacting with computer science in the privacy area, such as statistics, healthcare informatics, and law, to discuss and present research challenges and advances in data privacy and anonymity research.

This broad range of inciting workshops would not have been possible without the contributions and the support which we have received. First of all, we would like to thank all workshop organizers who have put together a highly interesting program as well as to all authors who submitted their works to the workshops. We specially thank the authors of the accepted papers and the invited speakers who presented their works in the workshops program. Needless to say, we are grateful to the members of the workshop program committees and external reviewers who have helped to put together a high-quality workshops program and we would like to acknowledge the conference organizers for their invaluable help at various stages of the process. We would also to thank the editors of the CEUR Workshop Proceedings (CEUR-WS.org) who have agreed to host these proceedings as well as ACM who are indexing them.

Sincerely,
Peter M. Fischer, *Workshops Chair*
Gustavo Alonso, *EDBT Program Chair*
Marcelo Arenas, *ICDT Program Chair*
Floris Geerts, *General Chair*

Algorithms for MapReduce and Beyond (BeyondMR)

Frank McSherry,
Foto N. Afrati (National Technical University of Athens, Greece),
Jacek Sroka (University of Warsaw, Poland)

Jedi: A Storage Manager for SIMD-aware, Worst-case Optimal Join Processing

Christopher Ré,
Stanford University

ABSTRACT

This talk describes a new graph-pattern engine called Jedi. Using a recent simplification of worst-case optimal join algorithms due to Ngo et al., Jedi translates join queries into a series of set intersection and union operations. Such set operations are ideally suited to modern CPUs that provides single-instruction, multiple data (SIMD) instructions. Using these ideas, we demonstrate that Jedi outperforms specialized graph engines by over an order of magnitude and relational systems by over two orders of magnitude on standard graph processing queries over real data.

Short Bio

Christopher (Chris) Re is an assistant professor in the Department of Computer Science at Stanford University and a Robert N. Noyce Family Faculty Scholar. His work's goal is to enable users and developers to build applications that more deeply understand and exploit data. Chris received his PhD from the University of Washington in Seattle under the supervision of Dan Suciu. For his PhD work in probabilistic data management, Chris received the SIGMOD 2010 Jim Gray Dissertation Award. He then spent four wonderful years on the faculty of the University of Wisconsin, Madison, before moving to Stanford in 2013. He helped discover the first join algorithm with worst-case optimal running time, which won the best paper at PODS 2012. He also helped develop a framework for feature engineering that won the best paper at SIGMOD 2014. In addition, work from his group has been incorporated into scientific efforts including the IceCube neutrino detector and PaleoDeepDive, and into Cloudera's Impala and products from Oracle, Pivotal, and Microsoft's Adam. He received an NSF CAREER Award in 2011, an Alfred P. Sloan Fellowship in 2013, and a Moore Data Driven Investigator Award in 2014.

Bounds for Overlapping Interval Join on MapReduce

Foto Afrati*
National Technical University of Athens, Greece
afrati@softlab.ece.ntua.gr

Shlomi Dolev† and Shantanu Sharma
Ben-Gurion University of the Negev, Israel
{dolev,sharmas}@cs.bgu.ac.il

Jeffrey D. Ullman
Stanford University
USA
ullman@cs.stanford.edu

ABSTRACT

We consider the problem of 2-way interval join, where we want to find all pairs of *overlapping intervals*, *i.e.*, intervals that share at least one point in common. We present lower and upper bounds on the replication rate for this problem when it is implemented in MapReduce. We study three cases, where intervals in the input are: (*i*) unit-length and equally-spaced, (*ii*) variable-length and equally-spaced, and (*iii*) equally-spaced with specific distribution of the various lengths. Our algorithms offer intuition as how to build algorithms for other cases, especially when we have some statistical knowledge about the distribution of the lengths of the intervals. E.g., if mostly large intervals interact with small intervals and not within themselves, then we believe our techniques can be extended to achieve better replication rate.

1. INTRODUCTION

MapReduce [3] is a programming model used for parallel processing of large-scale data. A *mapper* is an application of a (user-defined) map function to a single input and provides outputs in the form of $\langle key, value \rangle$ pairs. A *reducer* is an application of a (user-defined) reduce function to a single key and its associated list of $value$s. The *reducer capacity* — an important parameter — is an upper bound on the sum of the total number of inputs that are assigned to the reducer. We denote the capacity of a reducer by q, and all the reducers have an identical capacity. Interval join using MapReduce was introduced by Chawda et al. [2].

Example: Employees involved in the phases of a project. We show an example to illustrate temporal relations (a relation that stores data involving timestamps), intervals, and the need for interval join of overlapping intervals. Consider two (temporal) relations (*i*) $Project(Phase, Duration)$ that includes several phases of a project with their durations, and (*ii*) $Employee(EmpId, Name, Duration)$ that shows data of employees according to their involvement in the project's phases and their durations; see Figure 1. Here, the duration of a phase or the duration of an employee's involvement in a phase is given by an interval. It is interesting to find all the employee that are involved in a phase of the project. Formally, a query: find the name of all employees who worked in a phase of the project; requires us to join the relations to find all overlapping intervals of the relations. For example, the answer to the query includes employees U with id e_1, W with id e_3, X with id e_4, and Y with id e_5 are involved in RA phase the project.

Phase	*Duration*
Requirement Analysis (RA)	1-Mar – 1-May
Design (D)	1-Apr – 1-June
Coding (C)	1-May – 1-Aug

Project

EmpId	*Name*	*Duration*
e_1	U	1-Apr – 1-June
e_2	V	1-May – 1-July
e_3	W	1-Apr – 1-July
e_4	X	1-Mar – 1-June
e_5	Y	1-Mar – 1-Aug

Employee

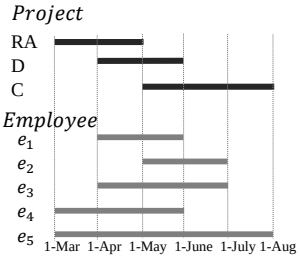


Figure 1: Two temporal relations ($Project(Phase, Duration)$ and $Employee(EmpId, Name, Duration)$) and their representation on a time diagram.

Problem Statement. We consider the problem of interval join of overlapping intervals, where two relations X and Y are given. Each relation contains binary tuples that represent intervals, *i.e.*, each tuple corresponds to an interval and contains the starting point and ending point of this interval. Each pair of intervals $\langle x_i, y_j \rangle$, where $x_i \in X$ and $y_j \in Y$, $\forall i, j$, such that intervals x_i and y_j share at least one common time, corresponds to an output.

A MapReduce job can be described by a mapping schema. A *mapping schema*, for this problem, assigns each interval to a number of reducers (via the formation of key-value pairs) so that (*i*) for each output (*i.e.*, pair of overlapping intervals), there exists a reducer that receives the corresponding pair of overlapping intervals that participate in the computation of this output and (*ii*) each reducer has a capacity (denoted by q hereon) that constraints the total number of intervals assigned to this reducer. The replication rate of a mapping schema is the average number of key-value pairs for each interval and is a significant performance parameter in a MapReduce job. We analyze here lower and upper bounds on the replication rate for the problem of overlapping intervals.

Our Contribution. We provide lower and almost matching

*Supported by the project Handling Uncertainty in Data Intensive Applications, co-financed by the European Union (European Social Fund) and Greek national funds, through the Operational Program "Education and Lifelong Learning," under the program THALES

†Supported by the Rita Altura Trust Chair in Computer Sciences, Lynne and William Frankel Center for Computer Sciences, Israel Science Foundation (grant 428/11), the Israeli Internet Association, and the Ministry of Science and Technology, Infrastructure Research in the Field of Advanced Computing and Cyber Security.

upper bounds for three cases: (*i*) unit-length and equally-spaced (Section 3), (*ii*) variable-length and equally-spaced, and (*iii*) equally-spaced with specific distribution of the various lengths (Section 4.1). In the third case, we assume that one set contains only small intervals and the other set only large intervals. We offer an algorithmic simple technique that takes advantage of this knowledge to build an algorithm that improves the replication rate of the second case above.

Related Work. Several types of join operations and a detailed review of join algorithms for temporal relations are given in [4]. MapReduce-based 2-way and multiway interval join algorithms of overlapping intervals *without regarding the reducer capacity* are presented in [2]. However, the analysis of a lower bound on replication of individual intervals is not presented; neither is an analysis of the replication rate of the algorithms offered therein.

2. THE SETTING

A (time) interval, i, is represented by a pair of times $[T_s^i, T_e^i]$, $T_s^i < T_e^i$, where T_s^i and T_e^i show the *starting-point* and the *ending-point* of the interval i, respectively. T_s^i - T_e^i is the *length* of the interval i. Two intervals, say interval i and interval j are called *overlapping intervals* if the intersection of both the interval is nonempty.

Mapping Schema. A mapping schema is an assignment of overlapping intervals to some given reducers under the following two constraints: (*i*) a reducer is assigned only q intervals, and (*ii*) for each output, we must assign the corresponding intervals to at least one reducer in common.

Replication rate, r: The *replication rate* [1] is the average number of key-value pairs created for an interval.

3. UNIT-LENGTH AND EQUALLY-SPACED INTERVALS

Two relations X and Y, each of n unit-length intervals are given. We assume that all the intervals have their starting-points in a closed interval $[0, k]$, *i.e.*, there is no interval that starts before 0 or after k. Thus, the space between every two successive intervals is $\frac{k}{n} < 1 \ll k$. In other words, the first interval starts at time 0, the second interval starts at time $\frac{k}{n}$, the third interval starts at time $\frac{2k}{n}$, and the last n^{th} interval starts at time $k - \frac{k}{n}$; see Figure 2.

The output we want to produce is a set of all pairs of intervals such that one interval overlaps with the other interval in the pair. The problem is not really interesting if all these intervals exist on the input. The real assumption is that some fraction of them exist, and the reducer capacity q is selected so that the expected number of inputs that actually arrive at a given reducer is within the desired limits, *e*.g., no more than what can be processed in main memory. In addition, the case of unit-length and equally-spaced interval is not realistic, but is explored because it gives us an idea of what optimal algorithms for more general and more realistic cases would look like.

A *solution* to the problem of interval join of overlapping unit-length and equally-spaced intervals is a mapping schema that assigns each interval of the relation X with all its overlapping intervals of the relation Y to at least one reducer in common, without exceeding q. Since every two consecutive intervals have an equal space ($\frac{k}{n}$), an interval $x_i \in X$ overlaps with at least $2\lfloor 1/\frac{k}{n} \rfloor + 1 = 2\lfloor \frac{n}{k} \rfloor + 1$ intervals of Y, where at least $\lfloor \frac{n}{k} \rfloor$ intervals of the relation Y have their ending-points between the starting-point and the ending-point of x_i, at least $\lfloor \frac{n}{k} \rfloor$ intervals of the relation Y have their starting-points between the starting-point and the ending-point of x_i, and an interval $y_i \in Y$ that have identical end-points as x_i (this inequality does not true for the intervals that have starting-points before 1 and after $k-1$). In this section, we will show a lower bound on the replication rate for interval join of overlapping unit-length and equally-spaced intervals. After that, we provide an algorithm, its correctness, and an upper bound on the replication rate obtained by the algorithm.

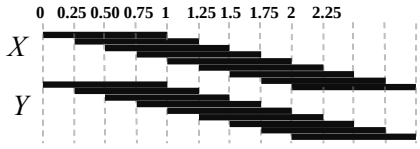


Figure 2: An example of unit-length and equally-spaced intervals, where $n = 9$ and $k = 2.25$.

Theorem 1 (Minimum replication rate) *For two relations, X and Y, of unit-length and equally-spaced intervals, the minimum replication of an interval, for joining each interval of the relation X with all its overlapping intervals of the relation Y, is (i) at least 2 when $2n > q \geq 2\lfloor \frac{n}{k} \rfloor + 2$, and (ii) at least $\frac{2}{q}\lfloor \frac{n}{k} \rfloor$ when $2 < q < 2\lfloor \frac{n}{k} \rfloor + 2$, where each relation holds n intervals, q is the reducer capacity, and k denotes that the starting points of intervals are in $[0, k]$.*

PROOF. First, we consider the case of $2n > q \geq 2\lfloor \frac{n}{k} \rfloor + 2$. When $q \geq 2n$, a single reducer is enough to hold all the intervals of both the relations, and hence, the reducer is able to provide all output pairs (of interval join of overlapping intervals). When $2\lfloor \frac{n}{k} \rfloor + 1 < q < 2n$, a single reducer may hold an interval $i \in X$ and all its $2\lfloor \frac{n}{k} \rfloor + 1$ corresponding overlapping intervals of the relation Y, and such a reducer is enough to provide all-pairs of the interval i with its overlapping intervals. However, at the same time, there must be at least a single interval, say interval j, that is assigned to the same reducer where the interval i is assigned, but the interval j is not assigned with all its corresponding overlapping intervals. Hence, the interval j must be assigned to at least one more reducer to be coupled with all its $2\lfloor \frac{n}{k} \rfloor + 1$ overlapping intervals. Therefore, the minimum replication of an interval is at least 2.

Now, we consider the case of $2 < q < 2\lfloor \frac{n}{k} \rfloor + 2$. Consider an interval i. Since the interval i has at least $2\lfloor \frac{n}{k} \rfloor + 1$ overlapping intervals, all these $(2\lfloor \frac{n}{k} \rfloor + 2)$ intervals cannot be assigned to a single reducer. The interval i can share a reducer with at most $q-1$ ($< 2\lfloor \frac{n}{k} \rfloor + 1$) intervals (of the relation Y). In order to assign the interval i with all the remaining overlapping intervals, it is required to assign subsets of the $2\lfloor \frac{n}{k} \rfloor + 1$ intervals, each subset with at most $q-1$ intervals. Such an assignment results in at least $2\lfloor \frac{n}{k} \rfloor + 1/q - 1$ subsets of $2\lfloor \frac{n}{k} \rfloor + 1$ overlapping intervals. Thus, the interval i must be sent to at least $2\lfloor \frac{n}{k} \rfloor + 1/q - 1 > \frac{2}{q}\lfloor \frac{n}{k} \rfloor$ reducers. □

Algorithm 1. We propose an algorithm for interval join of overlapping intervals, where two relations X and Y (each is of n intervals of unit-length and equally-spaced) are inputs. Recall that it is expected that not all possible intervals are present.

We divide the time-range from 0 to k into equal-sized partition of length $w = \frac{q-c}{3\lceil n/k \rceil}$, where $c = \lceil \frac{n}{k} \rceil + 2$. Consider that by partitioning of the time-range, we have P partitions. We now arrange P reducers, one for each partition. We consider a partition p_i, $1 \leq i \leq P$, and assign all the intervals of the relation X that exist in the partition p_i to the i^{th} reducer. In addition, we assign all the intervals of the relation Y that have their starting or ending-point in the partition p_i to the i^{th} reducer.

Explaining pseudocode of Algorithm 1. A mapper takes an interval $x_i \in X$ (line 2) and produces $\langle key, value \rangle$ pairs (line 4). The key represents a partition where the interval x_i exists and the

Cases	Solutions	Theorems	Replication rate
The **lower bounds**			
Unit-length and equally-spaced intervals		1	2 or $\frac{2}{q}\left\lfloor\frac{n}{k}\right\rfloor$
Variable-length and equally-spaced intervals		3	2 or $\frac{2}{q}\left\lfloor\frac{l_{min}}{s}\right\rfloor$
The **upper bounds**			
Unit-length and equally-spaced intervals	Algorithm 1	5	$\frac{3}{qT-S}\frac{S}{2}$
Variable length and equally-spaced (big-small) intervals	Algorithm 2	5	$\frac{3}{qT-S}\frac{S}{2}$
Variable length (different-length) and equally-spaced intervals	Algorithms 3 and 4	5	$\frac{3}{qT-S}\frac{S}{2}$

Table 1: The bounds for interval joins of overlapping intervals.

Algorithm 1: 2-way interval join algorithm for overlapping intervals of unit-length and equally-spaced intervals.

Inputs: X and Y: two relations, each is of n intervals.
Variables: k: A point on the timeline after that no interval can have a starting-point; w: The length of a partition $w = \frac{q-c}{3\lceil n/k\rceil}$, where $c = \lceil\frac{n}{k}\rceil + 2$; P: The total number of partitions and reducers.

```
1  Partition the time-range into P partitions, each of length w
2  Function Map_for_X(x_i ∈ X) begin
3      z ← count_partitions(x_i)
4      for j ← 1 to z do emit⟨j, x_i⟩ ;
5  Function Map_for_Y(y_i ∈ Y) begin
6      sp ← starting_points(y_i), ep ← ending_points(y_i)
7      emit⟨sp, y_i⟩, emit⟨ep, y_i⟩
8  Function reduce(⟨key, list_of_values[]⟩) begin
9      for j ← 1 to P do
10         Reducer i is having
           ⟨i, list_of_values[x_a, x_b, ..., y_a, y_b, ...]⟩
11         Perform interval join over overlapping intervals
12 Function count_partitions(x_i) begin
       c ← Count the total number of partitions that x_i crosses
       return c
```

total number of $\langle key, value\rangle$ pairs for the interval x_i depends on the total number of partitions that the interval x_i crosses, by calling function $count\_partitions()$ (lines 3 and 12). Also, a mapper processes an interval $y_i \in Y$ (line 5) and produces at most two $\langle key, value\rangle$ pairs (line 7), where the first pair and the second pair are corresponding to a partition where y_i has the starting-point and the ending-point, respectively (line 6). The $value$ represents the interval x_i or y_i itself. In the reduce phase, a reducer i fetches all the intervals of the relations X and Y that have a key i (line 10) and provides the final outputs, line 11.

Theorem 2 (Algorithm correctness) *Let $c = \lceil\frac{n}{k}\rceil + 2$ and let $q = 3w\lceil\frac{n}{k}\rceil + c$, Algorithm 1 assigns each pair of overlapping intervals to at least one reducer in common, where each relation, X and Y, holds n intervals, q is the reducer capacity, k denotes that the starting points of intervals are in $[0, k]$, and w is the length of a partition.*

PROOF. Since every two successive intervals have $\frac{k}{n}$ spacing, an interval $i \in X$ can overlap with at most $2\lceil\frac{n}{k}\rceil$ intervals of the relation Y. First, we consider $w < 1$; in a partition, p of length w, an interval i can overlap with at most $2w\lceil\frac{n}{k}\rceil$ intervals of the relation Y. Note that there are at most $w\lceil\frac{n}{k}\rceil$ intervals (of the relation X) that have their starting-points after the starting-point of the interval i in the partition p, and we called these intervals *post-intervals* of the interval i. Also, there are at most $c = \lceil\frac{n}{k}\rceil$ intervals (of the relation X) that have either their ending-points in the partition p or cross the partition p; we call these intervals *pre-intervals* of the interval i.

Thus, for $w < 1$, $q = 3\lceil\frac{n}{k}\rceil + c$, we can assign the interval i, post-intervals of i that lie in the partition p, and pre-intervals of i that lie in partition p at a single reducer. Such an assignment occupies $w\lceil\frac{n}{k}\rceil + c - 1$ capacity of the reducer. The remaining capacity, $2w\lceil\frac{n}{k}\rceil + 1$, of the reducer is used to assign all $2w\lceil\frac{n}{k}\rceil$ overlapping intervals of the interval i and an interval, $i' \in Y$ that have an identical starting-point as the interval i. (Note that i' is an overlapping interval for some of the pre-intervals and the post-intervals of i.) Thus, the interval i is assigned to a reducer with all its $2w\lceil\frac{n}{k}\rceil$ overlapping intervals of the relation Y. Further, the interval i will also be paired with all its remaining $2\lceil\frac{n}{k}\rceil - 2w\lceil\frac{n}{k}\rceil$ overlapping intervals at some reducers.

Now, we consider $w \geq 1$. In this case, for a partition p, there must be an interval $i \in X$ that can be assigned to a reducer with all its $2\lceil\frac{n}{k}\rceil$ overlapping intervals of the relation Y. Also, there are at most $\lceil\frac{n}{k}\rceil$ post-intervals and $c = \lceil\frac{n}{k}\rceil$ pre-intervals (of the interval i) that lie in the partition p. Thus, we can assign interval i, post-intervals of i, and pre-intervals of i at a single reducer. In addition, an interval, $i' \in Y$ such that i and i' have an identical starting-point, is also assigned to the reducer. Therefore, the interval i is paired with all $2\lceil\frac{n}{k}\rceil$ overlapping intervals (of the relation Y) at the reducer. □

4. VARIABLE-LENGTH AND EQUALLY-SPACED INTERVALS

Two relations X and Y, each of n intervals, are given, where all intervals can have non-identical length but equally-spaced. We assume that the first interval starts at time 0, and the space between every two successive intervals is $s < 1$; see Figure 3, where a relation X has 6 intervals, and a relation Y has also 6 intervals. A *solution* to the problem of interval join of overlapping variable-length and equally-spaced intervals is a mapping schema such that each pair of overlapping intervals, one from each of the relations, is sent to at least one reducer in common without exceeding q.

We consider two types of intervals, as follows: (*i*) *big and small intervals*: one of the relation, say X, is holding most of the intervals of length l and the other relation, say Y, is holding most of the intervals of length $l' \gg l$; we call intervals of the relations X and Y as *small intervals* and *big intervals*, respectively; and (*ii*) *different-length intervals*: all the intervals of both the relations are of different-length (we will consider the second case in Appendix). In this section, we will provide lower bounds on the replication rate for both types of intervals. We then provide algorithms for interval join of overlapping intervals and show a upper bound on the replication rate. Throughout this section, we will use the following notations: l_{max}: the maximum length of an interval, l_{min}: the minimum length of an interval, and w: length of a partition.

4.1 Big and small intervals

In this section, we consider a special case of variable-length and equally-spaced intervals, where all of the intervals of two relations X and Y have length l_{min} and l_{max}, respectively, such that $l_{min} \ll l_{max}$; see Figure 3. We call the intervals of the relations X and Y as *small intervals* and *big intervals*, respectively.

Since every two successive intervals have an equal space, s, an interval $x_i \in X$ of length l_{min} can overlap with at least $2\lfloor\frac{l_{min}}{s}\rfloor + 1$ intervals of the relation Y, where at least $\lfloor\frac{l_{min}}{s}\rfloor$ intervals of the

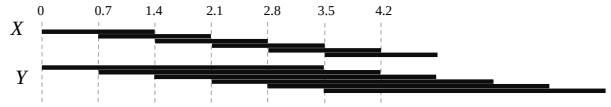


Figure 3: An example of big and small length but equally-spaced intervals, where $n = 6$ and $s = 0.7$.

relation Y have their ending-points between the starting and the ending-points of x_i, at least $\lfloor \frac{l_{min}}{s} \rfloor$ intervals of the relation Y have their starting-points between the starting and the ending-points of x_i, and an interval $y_i \in Y$ has an identical starting-point as x_i. In addition, an interval $x_i \in X$ of length l_{max} can overlap with at most $2\lfloor \frac{l_{max}}{s} \rfloor + 1$ intervals of the relation Y, where at most $\lfloor \frac{l_{max}}{s} \rfloor$ intervals of the relation Y have the ending-points between the starting and the ending-points of x_i and at most $\lfloor \frac{l_{max}}{s} \rfloor$ intervals of the relation Y have the starting-points between the starting and the ending-points of x_i, and an interval $y_i \in Y$ has an identical starting-point as x_i.

Theorem 3 (Minimum replication rate) *For a relation X of n small and equally-spaced intervals and a relation Y of n big and equally-spaced intervals, the minimum replication of an interval, for joining each interval of the relation X with all its overlapping intervals of the relation Y, is (i) at least 2 when $2n > q \geq 2\lfloor \frac{l_{min}}{s} \rfloor$, and (ii) at least $\frac{2}{q}\lfloor \frac{l_{min}}{s} \rfloor$ when $2 < q < 2\lfloor \frac{l_{min}}{s} \rfloor$, where q is the reducer capacity, s is the spacing between every two successive intervals, and l_{min} is the length of the smallest interval.*

PROOF. First we consider the case of $2n > q \geq 2\lfloor \frac{l_{min}}{s} \rfloor + 2$. When $q \geq 2n$, all the $2n$ intervals of the relations X and Y can be assigned to a single reducer, which is able to provide all output pairs. When $2\lfloor \frac{l_{min}}{s} \rfloor + 2 < q < 2n$, a single reducer cannot hold all the $2n$ intervals of the relations X and Y. Hence, at least a single interval, say j, that is not assigned with all its $2\lfloor \frac{l_{min}}{s} \rfloor + 1$ overlapping intervals must be assigned to another reducer. Therefore, the minimum replication of an interval is at least 2.

Now, we consider the case of $2 < q < 2\lfloor \frac{l_{min}}{s} \rfloor + 2$. Consider an interval i of length l_{min}. Since the interval i has at least $2\lfloor \frac{l_{min}}{s} \rfloor + 1$ overlapping intervals, all these $(2\lfloor \frac{l_{min}}{s} \rfloor + 2)$ intervals cannot be assigned to a single reducer. The interval i can share a reducer with at most $q - 1$ intervals of the relation Y. Hence, in order to assign the interval i with all the remaining overlapping intervals, it is required to assign subsets of overlapping intervals of the relation Y such that each subset holds at most $q - 1$ intervals. Thus, the interval i must be sent to at least $2\lfloor \frac{l_{min}}{s} \rfloor + 1/q - 1 > \frac{2}{q}\lfloor \frac{l_{min}}{s} \rfloor$ reducers. □

Algorithm 2. Algorithm 2 for interval join of overlapping intervals of a relation X of small and equally-spaced intervals and a relation Y of big and equally-spaced intervals works in a similar fashion as Algorithm 1 performs the join operation. However, Algorithm 2 creates P partitions of the time-range (from 0 to ns), each of length of length $w = \frac{q-c}{3\lceil l_{min}/s \rceil}$, where $c = \lceil \frac{l_{min}}{s} \rceil + 2$. Note that in Algorithm 2, small intervals are assigned to several reducers corresponding to their partitions that they cross, and large intervals are assigned to only two reducers corresponding to their stating and ending points' partitions. The correctness of Algorithm 2 proves that each pair of overlapping intervals is assigned to at least one reducer in common, where $q = 3w\lceil \frac{l_{min}}{s} \rceil + c$, where $c = \lceil \frac{l_{min}}{s} \rceil + 2$.

4.2 An upper bound for the general case

In this section, we show an algorithm and an upper bound on the replication rate for the problem of interval join of variable-length but equally-spaced intervals. We use the following notations: T: the length of time in which all intervals exist, *i.e.*, all intervals begin at some time greater than or equal to 0 and end by time T; n: the number of intervals in each of the two relations, X and Y, S: the total length of all the intervals in one relation; and w: the length of time corresponding to one reducer, *i.e.*, we divide T into $\frac{T}{w}$ equal-length segments, each of length w.

Algorithm 3. Algorithm 3 works in a manner similar to Algorithms 1 and 2 do. But this algorithm does more than Algorithms 1 and 2. It finds all intervals that intersect, regardless of whether they overlap, are superimposed, or any other relation. We divide the time-range into $\frac{T}{w}$ equal-sized partitions and arrange $\frac{T}{w}$ reducers, one for each partition. After that, we follow the same procedure as followed in Algorithms 1 and 2.

Theorem 4 (Algorithm correctness) *Algorithm 3 assigns each pair of overlapping intervals to at least one reducer in common, where $q = \frac{3nw+S}{T}$, each of the two relations, X and Y, holds n intervals, q is the reducer capacity, S is the total length of all the intervals in one relation, w is the length of a partition, and T is the length of time in which all intervals exist.*

PROOF. Following the algorithm, each of the n intervals of the relation Y is sent to at most two reducers. Since there are $\frac{T}{w}$ reducers, a reducer receives $\frac{2nw}{T}$ inputs from Y in average. Since the length of all the intervals of the relation X is S, the average length of intervals is $\frac{S}{n}$. Following the algorithm, an interval of X is sent to $1 + \frac{S}{nw}$ reducers. Since there are $\frac{T}{w}$ reducers, the reducer receives $(1 + \frac{S}{nw})\frac{nw}{T}$ inputs from X in average. Thus, a reducer receives at most $\frac{2nw}{T} + \frac{nw}{t}(1 + \frac{S}{nw}) = \frac{3nw+S}{T}$ inputs, which is equal to the given reducer capacity. □

Theorem 5 (Replication rate) *For $q = \frac{3nw+S}{T}$ and two relations, X and Y, of variable-length but equally-spaced, the replication rate of an interval, for joining each interval of the relation X with all its overlapping intervals of the relation Y is $\frac{3}{qT-S}\frac{S}{2}$.*

5. REFERENCES

[1] F. N. Afrati and et al. Upper and lower bounds on the cost of a map-reduce computation. *PVLDB*, 6(4):277–288, 2013.

[2] B. Chawda and et al. Processing interval joins on map-reduce. In *EDBT*, pages 463–474, 2014.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, pages 137–150, 2004.

[4] D. Gao, C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Join operations in temporal databases. *VLDB J.*, 14(1):2–29, 2005.

APPENDIX

We consider a case of different-length intervals, *i.e.*, all the n intervals of each relation, X and Y, can have different-length. For a relation X and a relation Y, each is of n different-length but equally-spaced intervals, the minimum replication of an interval, for joining each interval of the relation X with all its overlapping intervals of the relation Y, is same as given in Theorem 3.

Algorithm 4. We propose an algorithm for interval join of overlapping different-length and equally-spaced intervals, which belong to two relations X and Y, each is of n intervals. Algorithm 4 works identically to Algorithms 1, 2, and 3. However, Algorithm 4 is different from Algorithms 1, 2 and 3, when it divides the time-range from 0 to ns into P partitions, each of length $w = \frac{q-c}{3\lceil l_{max}/s \rceil}$, where $c = \lceil \frac{l_{max}}{s} \rceil + 2$. The algorithm correctness shows that Algorithm 4 assigns each pair of overlapping intervals to at least one reducer in common, where $q = 3w\lceil \frac{l_{max}}{s} \rceil + c$, where $c = \lceil \frac{l_{max}}{s} \rceil + 2$.

Cuneiform

A Functional Language for Large Scale Scientific Data Analysis

Jörgen Brandt Marc Bux Ulf Leser

Humboldt-Universität zu Berlin
Unter den Linden 6, D-10099 Berlin, Germany
{brandjoe, bux, leser}@informatik.hu-berlin.de

ABSTRACT

The need to analyze massive scientific data sets on the one hand and the availability of distributed compute resources with an increasing number of CPU cores on the other hand have promoted the development of a variety of languages and systems for parallel, distributed data analysis. Among them are data-parallel query languages such as Pig Latin or Spark as well as scientific workflow languages such as Swift or Pegasus DAX. While data-parallel query languages focus on the exploitation of data parallelism, scientific workflow languages focus on the integration of external tools and libraries. However, a language that combines easy integration of arbitrary tools, treated as black boxes, with the ability to fully exploit data parallelism does not exist yet. Here, we present Cuneiform, a novel language for large-scale scientific data analysis. We highlight its functionality with respect to a set of desirable features for such languages, introduce its syntax and semantics by example, and show its flexibility and conciseness with use cases, including a complex real-life workflow from the area of genome research. Cuneiform scripts are executed dynamically on the workflow execution platform Hi-WAY which is based on Hadoop YARN. The language Cuneiform, including tool support for programming, workflow visualization, debugging, logging, and provenance-tracing, and the parallel execution engine Hi-WAY are fully implemented.

1. INTRODUCTION

Over the recent years, data sets in typical scientific (and commercial) areas have grown tremendeously. Also, the complexity of analysis procedures has increased at essentially the same pace. For instance, in the field of bioinformatics the cost and speed at which data can be produced is improving steadily [3, 18, 35]. This makes possible entirely novel forms of scientific discoveries which require ever more complex analysis pipelines (e.g., personalized medicine, meta-genomics, genetics at population scale) developed by tens of thousands of scientists around the world. Analysis infrastructures have to keep up with this development. In particular, they must be able to scale to very large data sets, and they must be extremely flexible in terms of integrating and combining existing tools. Scientific workflow management systems have been proposed to deal with the latter problem [12]. Scientific workflows are descriptions of data processing steps and the flow of information between them [10]. Most scientific workflow languages like Galaxy [17] or Taverna [21] allow the user to create light-weight wrappers around existing tools and libraries. In doing so, they avoid the necessity of reimplementing these tools to conform with the API of the execution environment. However, many scientific workflow systems do not take advantage of splitting input data into partitions to exploit data parallelism which limits their scalability. In contrast, partitioning input data to achieve data parallelism is a main advantage of the execution environments underlying data-parallel query languages like Pig Latin [16, 33] or Spark [45], thus, addressing the former issue. However, these are not designed to make native integration of external tools easy. Instead, they require developers to create heavy wrappers around existing tools as well as to perform costly conversion of data from their native formats to the system-specific data model, or they expect developers to reimplement all algorithms in their specific data-parallel languages. While creating wrappers is laborious, error-prone, and incurs runtime penalties, reimplementation is infeasible in areas like genomics where new algorithms, new data types, and new applications emerge essentially every day.

To the best of our knowledge, a language for large-scale scientific computing that offers both light-weight wrapping of foreign tools and high-level data-parallel structures currently does not exist. Here, we present Cuneiform, a language that aims at filling this gap. Cuneiform is a universal functional workflow language offering all important features currently required in scien-

tific analysis like abstractions and a dynamic execution model in a language with implicit state. Its main focus, however, is (i) the ease with which external tools written in any language can be integrated, and (ii) the support for a rich set of algorithmic skeletons (or second-order functions) enabling automatic parallelization of execution. Cuneiform is fully implemented and comes with decent tool support, including executable provenance traces, visualization of conceptual and physical workflow plans, and debugging facilities. Cuneiform workflows can be executed on the workflow execution engine Hi-WAY which builds on Hadoop YARN. Together, Cuneiform and Hi-WAY form a fully functional, scalable, and easily extensible scientific workflow system.

The remaining parts of this paper are structured in the following way: Section 2 introduces characteristics of workflow languages important for scientific analysis that drove the design of Cuneiform. We introduce Cuneiform by example in Section 3. In Section 4 we explain the implementation of two exemplary workflows in Cuneiform to highlight its versatility and expressive power. Hi-WAY is briefly introduced in Section 5.[1] Related work is presented in Section 6, and Section 7 concludes the paper.

2. LANGUAGE CHARACTERIZATION

In this section we outline the language properties that we consider important for parallel scientific analysis and that drove the design of Cuneiform. We categorize different types of languages and discuss important language features.

First, we can distinguish languages with *implicit state* and *explicit state* [36]. Implicit state languages do not have mutable memory. Purely functional languages like Haskell or Miranda have implicit state. In these languages variables are only place-holders for immutable expressions. In contrast, imperative languages like C or Java and multi-paradigm languages like Scala or Lisp have explicit state. In these languages variables can change their values throughout the execution of a program.

Scientific workflow languages are typically implicit state languages while Spark [44, 45], FlumeJava [7], or DryadLINQ [14, 43] inherit explicit state from their respective host languages. There are arguments promoting either of both approaches: Implicit state languages profit from their ability to fully exploit task parallelism because the order of tasks is constrained only by data dependencies [6].[2] Explicit state is preferable if functions need to learn from the past and change their behavior [36]. However, the introduction of explicit state incurs additional constraints on the task execution order, thereby, limiting the ability to automatically infer parallelism.

In the following we outline the requirements towards a scalable workflow specification language. By focusing on this specific set of requirements, we define the scope for the discussion of Cuneiform and for the comparison of different languages. This list of requirements is, however, not comprehensive.

Abstractions In the Functional Programming (FP) paradigm the term abstraction refers to an expression that binds one (or more) free variables in its body. When a value is applied to the expression, the first bound variable is replaced with the applied value. In scientific workflows, abstractions are referred to as subworkflows where the subworkflow's input ports represent the bound variables. While abstractions are common in functional languages for distributed computation (like Eden [5, 26]) or distributed multi-paradigm languages (like Spark), some scientific workflow languages do not allow for the definition of abstractions in the form of subworkflows, e.g., Galaxy [17]. Other scientific workflow languages like Pegasus DAX [13], KNIME [4], Swift, or Taverna do provide subworkflows. Pig Latin [16, 33] introduces abstractions through its macro definition feature. Since abstractions facilitate the reuse of reoccurring patterns, they are an important feature of any high-level programming model.

Conditionals A conditional is a control structure that evaluates to its then-branch only if a condition is true and otherwise evaluates to its else-branch. Like abstractions, conditionals are common in functional and multi-paradigm languages. Many scientific workflow languages provide conditionals as top-level language elements [2], e.g., KNIME, Taverna, or Swift. However, in some other scientific workflow languages they are omitted, e.g., Galaxy or Pegasus DAX. Also, Pig Latin comes without conditionals that would allow for alternate execution plans depending on a boolean condition. Spark, on the other hand, inherits its conditionals from Scala. Conditionals are important when a workflow has to follow a different execution path depending on a computational result that cannot be anticipated a priori. For instance, consider a scenario where two algorithms can be employed to solve a problem. One algorithm performs comparably better on noisy input data, while the other performs better on clean data. If assessing the quality of the data is part of the workflow then

[1] Note that the focus of this paper is on Cuneiform, while a complete description of Hi-WAY will be published elsewhere.
[2] Taverna is an exception as it introduces control links to explicitly constrain the task execution order in addition to data dependencies.

the decision what algorithm to use has to be made at execution time. Another example is the application of an iterative learning algorithm. If the exit condition of the algorithm is determined by some convergence criterion, the number of iterations cannot be anticipated a priori. This way, conditionals introduce uncertainty in the concrete workflow structure making it impossible to infer a workflow's invocation graph prior to execution. Nevertheless, conditionals are an important language feature.

Composite data types Composite data types are data structures composed of atomic data items. Lists, sets, or bags are composite data types. In many cases, languages with support for composite data types also provide algorithmic skeletons (see below) to process them, e.g., map, reduce, or cross product. Swift, KNIME, and Taverna are scientific workflow languages with support for composite data types. Other scientific workflow languages, like Galaxy or Pegasus DAX, support only atomic data types. Data-parallel query languages, like Spark or Pig Latin, however, provide extensive support for composite data types. Note that composite data types, like conditionals, introduce uncertainty in the concrete workflow structure before its actual execution. For instance, if a task outputs a list with an unknown size and each list item is consumed by a proper subsequent task, the number of such tasks is unknown prior to execution. This calls for a dynamic, adaptive approach to task scheduling. Using composite data types is a powerful and elegant way to specify data-parallel programs.

Algorithmic skeletons Algorithmic skeletons are second order functions that represent common programming patterns. From the perspective of imperative languages, they can be seen as templates that outline the coarse structure of a computation [8]. To exploit the capabilities of parallel, distributed execution environments, a language can emphasize parallelizable algorithmic skeletons and de-emphasize structures that could impose unnecessary constraints on the task execution order. For instance, expressing the application of a function to each element of a list as a for-loop with an exit condition dismisses the parallel character of the operation. Expressing the exact same operation as a map, on the other hand, retains the parallelism of the operation in its language representation. Some scientific workflow languages, like Pegasus DAX or Galaxy, do not provide any algorithmic skeletons. In contrast, Taverna, Swift, or KNIME provide algorithmic skeletons in various forms. For instance, Taverna implicitly iterates lists if an unary task is applied to a list. Moreover, Taverna provides cross- and dot product skeletons. Swift provides the foreach and iterate-until skeletons. Algorithmic skeletons are particularly important in Scala. Thus, Spark exposes a number of algorithmic skeletons to control distributed computation [44]. Pig Latin uses algorithmic skeletons based on the SQL model of execution. Like general abstractions, algorithmic skeletons facilitate the reuse of reoccurring algorithmic patterns. Such patterns commonly appear in scientific data analysis applications.

Foreign Function Interface (FFI) An FFI allows a program to call routines written in a language other than the host language. Many programming languages provide an FFI with the goal of accelerating common subroutines by interfacing with machine-oriented languages like C. Scientific workflow languages provide FFIs in the form of simple wrappers for external tools. For instance, Swift and Pegasus DAX allow language users to integrate Bash scripts. Taverna provides Beanshell and R services, and KNIME provides snippet-nodes for Java, R, Perl, Python, Groovy, and Matlab. These FFIs do not have the purpose to accelerate routines but to integrate existing tools and libraries with minimum effort. In Pig Latin or Meteor [19], User Defined Functions (UDFs) are provided in the form of Java libraries which need to be wrapped by an extra layer of code providing particular data transformations from the tools native file formats to the system's data model and back. Similar wrappers have to be implemented to use foreign tools in Spark. The FFI is the language feature that makes integration of external tools and libraries easy. It is the entry point for any piece of software that has not been written in the host language itself. A general and light-weight FFI enables researchers to reuse their tools in a data-parallel fashion without further adaptation or the additional layer of complexity of a custom wrapper.

Universality Universal languages can express any computable function. Most general purpose programming languages are universal. Scientific workflow languages including Swift, Galaxy, Taverna, and Pegasus DAX are not universal. Additionally, some data-parallel query languages like Pig Latin are not universal. In contrast, Skywriting is an example for a universal language for distributed computation [28]. Spark inherits the universality property from Scala. Similarly, FlumeJava and DryadLINQ inherit the universality property from their

respective host languages Java and C#. We do not consider universality a requirement for a workflow specification language. Nonetheless, it is a language property worth investigating.

3. CUNEIFORM

In this section we present Cuneiform. We show that it is simple to express data-parallel structures and to integrate external tools in Cuneiform. Furthermore, we demonstrate fundamental language features by example and discuss how Cuneiform workflows are evaluated and mapped to distributed compute resources for scheduling and execution.

Cuneiform is a Functional Programming (FP) language with implicit state. Cuneiform has in common with scientific workflow languages its light-weight, versatile FFI allowing users to directly use external tools or libraries from scripting languages including Lisp, Matlab, Octave, Perl, Python, and R. In principle, Cuneiform can interface with any programming language that has support a string and list data type. Cuneiform has in common with data-parallel query languages that it provides facilities to exploit data parallelism in the form of composite data types and algorithmic skeletons to process them. Cuneiform comes in the form of a universal FP language providing abstractions and conditionals.

In the following, we introduce important concepts of Cuneiform by example. We highlight the interplay of Cuneiform's features using more complex workflows in Section 4, while Section 5 briefly sketches the Hi-WAY execution environment.

3.1 Task definition and Foreign Function Interface

The *deftask* statement lets users define Cuneiform tasks, which are the same as functions in FP languages. It expects a task name and a prototype declaring the input/output variables a task invocation consumes/produces. A task definition can be either in Cuneiform or in any of the supported foreign scripting languages. In the following example we define a task *greet* in Bash which consumes an input variable *person* and produces an output variable *out*.

```
deftask greet( out : person )in bash *{
  out="Hello $person"
}*
```

The task defined in this listing can be applied by binding the parameter *person* to a value. In this example we bind it to the string "Peter".

```
greet( person: 'Peter' );
```

The value of this expression is the string "Hello Peter". Cuneiform assumes foreign tasks to be side effect-free. I.e., the result of a task should be deterministic and depend only on the value of its arguments. However Cuneiform has no way of enforcing this behavior.

3.2 Lists

Cuneiform has one built-in composite data type: the list. There is no atomic data type. In the following example, we define a variable *friends* to be a list of two strings being "Jutta" and "Peter".

```
friends = 'Jutta' 'Peter';
greet( person: friends );
```

Applying the function *greet* to this list, evaluates to a list with two string elements: "Hello Jutta" and "Hello Peter". Thus, the standard way of applying a task to a single parameter, is to map this task to all elements in the list.

To consume a list as a whole, we have to aggregate the list. We can mark a parameter as aggregate by surrounding it with angle brackets. The following listing defines the task *cat* that takes a list of files and concatenates them.

```
deftask cat
  ( out( File ) : <inp( File )> )in bash *{
  cat ${inp[@]} > $out
}*
```

When a list is aggregated in a foreign task call, Cuneiform has to hand over this list as a whole. Thus, Cuneiform loses control over the way data parallelism is exploited in processing this list. Furthermore, the interpreter has to defer the aggregating task unless the whole list has been computed.

3.3 Parallel algorithmic skeletons

Cuneiform provides three basic algorithmic skeletons: aggregate, n-ary cross product, and n-ary dot product. A map can be viewed as any unary product. These basic skeletons are the building blocks to form arbitrarily complex skeletons. If a task has multiple parameters, the standard behaviour is to apply the function to the cross product of all parameters.

Suppose there is a command line tool *sim* that takes a temperature in $°C$ and a pH value, performs some simulation, and outputs the result in the specified file. We could wrap and call this tool in the following way:

```
deftask simulate
  ( out( File ) : temp ph )in bash *{
  sim -o $out -t $temp -p $ph
}*

temp = -5 0 5 10 15 20 25 30;
ph = 5 6 7 8 9;

simulate( temp: temp ph: ph );
```

This script performs a parameter sweep from -5 to $30°C$ and from pH value 5 to 9. Herein, each of the 8 temperature values is paired with each of the 5 pH values resulting in 40 invocations of the *sim* tool. How multiple lists are combined is generally determined by the prototype of a task. The cross product is the default algorithmic skeleton to combine task parameters.

Lastly, suppose we are given two equally long lists of strings. We want to concatenate each string from the first list with each string from the second list separated by a white space character. A dot product between two or more parameters is denoted in the task prototype by surrounding them with square brackets. We choose to perform the string concatenation in Python.

```
deftask join( c : [a b] )in python *{
c = a+' '+b
}*
```

In the following listing we define the variables a and b to be a pair of two-element lists and call the previously defined task *join* on them. The result of this operation is a two-element list with the members "Hello world" and "Goodnight moon".

```
a = 'Hello' 'Goodnight';
b = 'world' 'moon';

join( a: a b: b );
```

3.4 Execution semantics

Cuneiform workflow scripts are parsed and transformed into a graph representation prior to interpretation. Variables are associated not with concrete values but with uninterpreted expressions thereby constituting a call-by-name evaluation strategy. Consequently, an expression is evaluated only if that expression is actually used (lazy evaluation). This ensures, not only, that all computation actually contributes to the result, but also, since evaluation is deferred to the latest possible moment, that parallelization is performed on the level of the whole workflow rather than the level of only subexpressions. Furthermore, instead of traversing the workflow graph during execution, Cuneiform performs workflow graph reduction. This means that subexpressions in the workflow graph are continuously replaced with what they evaluate until the result of the computation remains. Accordingly, workflow execution is dynamic, i.e., the order in which which tasks are evaluated is determined only at runtime, a model which naturally supports data dependent loops and conditions. This aspect discerns Cuneiform from many other systems that require a fixed execution graph to be compiled from the workflow specification. Herein, Cuneiform resembles Functional Programming language interpretation.

When an expression involves external software, a ticket is created and passed to the execution environment. Expressions depending on that ticket are deferred while other expressions continue to be evaluated. A ticket, encapsulating a concrete task on a concrete input, thus, is the basic computational unit in Cuneiform. For any given point in time, the set of available tickets may be evaluated in any order. This order has to be determined by the scheduler of the runtime environment, taking into account the currently available resources. Each time the execution environment finishes evaluation of a ticket, the result is reported back to the Cuneiform interpreter which then continues reduction of the respective expression. When there are no more tickets to evaluate and the expression cannot be further reduced, execution stops and the workflow result is returned.

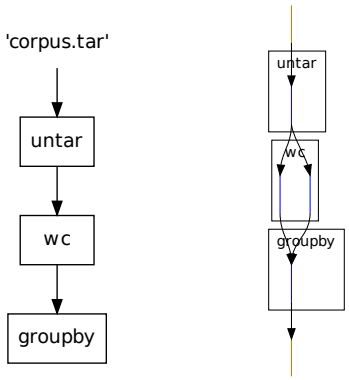


Figure 1: Static call graph (left) and invocation graph (right) for canonical word count with a corpus of 2 text files.

4. WORKFLOW EXAMPLES

In this section we present two example workflows in Cuneiform. The first, a canonical word count example, is chosen for its simplicity and comparability with other programming models (e.g., MapReduce [11]). The second workflow performs variant calling on Next-Generation Sequencing data [34].

4.1 Canonical word count

The canoncical word count workflow consumes a corpus of text files and, for each file, counts the occurrences of words. It outputs a table that sums up the occurrences of words in all files. The workflow consists of two steps. In the first step, words are counted individually in each file. In a second step, the occurrence tables are aggregated by summing up the corresponding occurrence counts. Figure 1 displays (a) the static call graph automatically derived from the workflow script and (b) the invocation graph that unfolds during workflow execu-

tion. Herein, the static call graph is a visualization that takes into account only the unevaluated workflow script. In contrast, the invocation graph is derived from the workflow execution trace. Each yellow line in the invocation graph stands for a single data item. Each blue line stands for a task invocation. A task invocation depends only on its predecessors connected to it via directed edges.

To specify the word count workflow we express both tasks separately as R scripts. First, we use R's *table* function to extract word counts from a string:

```
deftask wc( csv( File ) : txt( File ) )in r *{
  dtm <- table( scan( txt, what='character' ) )
  df  <- as.data.frame( dtm )
  write.table( df, csv, col.names=FALSE,
    row.names=FALSE )
}*
```

Next, we use the function *rbind* to concatenate the list of tables, generated in the previous step and aggregate the resulting table using *ddply* which is part of the R library *plyer*.

```
deftask groupby
  ( result( File ) : <csv( File )> )in r *{

  library( plyr )
  all <- NULL
  for( i in csv )
    all <- rbind( all,
             read.table( i, header=FALSE ) )
  x <- ddply( all, .( V1 ), summarize,
         count=sum( V2 ) )
  write.table( x, result, col.names=FALSE,
    row.names=FALSE )
}*
```

To extract all files in an archive holding the text corpus to be analyzed we use the following task definition:

```
deftask untar
  ( <list( File )> : tar( File ) )in bash *{
  tar xf $tar
  list=`tar tf $tar`
}*
```

The workflow definition calls the tasks *untar*, *wc*, and *groupby* in order. Finally, we query the workflow result:

```
txt = untar( tar: 'corpus.tar' );
csv = wc( txt: txt );
result = groupby( csv: csv );
result;
```

Called this way, *wc* is invoked once for each file. Each invocation is processed in parallel by Hi-WAY. In contrast, the tasks *groupby* and *untar* each have a single invocation.

Note that the two tasks, *wc* and *groupby*, implement a complete word count, including file I/O, parsing, dictionary management, and two-phase counting. No other tools are needed. Furthermore, we are free to choose the programming language. For instance, in a different implementation we might use Perl libraries or the command line tool *awk*.

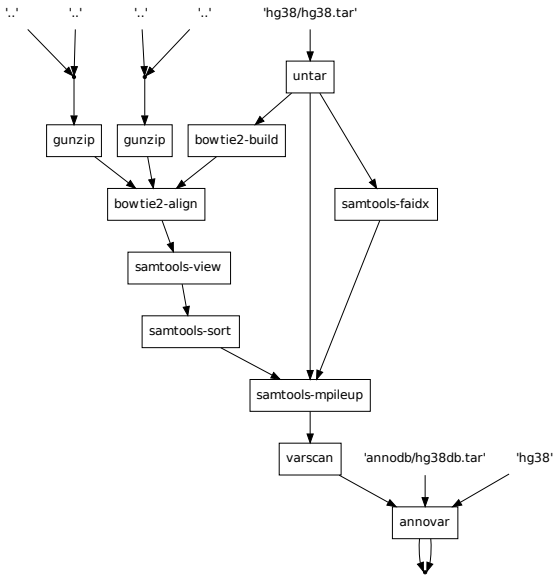


Figure 2: Static call graph for variant calling workflow

4.2 NGS variant calling

The second workflow demonstrates how variant calling in the application domain of Next-Generation Sequencing (NGS) can be performed in Cuneiform. In this workflow, a set of DNA sequence read files in FastQ format is mapped against a reference genome. Subsequently, the alignments are sorted, a multiple pileup is performed, and variants are called and annotated. As typical for scientific analysis pipelines, all steps are performed by external command line tools [34]. Figure 2 shows the static call graph and Figure 3 shows the invocation graph for this workflow. In the following discussion we omit all foreign task definitions.[3]

The input to the workflow is a reference genome, a set of sample files, as well as an annotation database. The workflow calls two nested subworkflows *per-sample* and *per-chromosome* which reflect the data parallelization scheme. Up to this point, we defined only variable assignments which would not trigger any computation. Thus, we need to query the variables *fun* and *exonicfun* to define the workflow output.

[3]The full workflow can be downloaded from https://github.com/joergen7/cuneiform/blob/master/cuneiform-dist/src/main/cuneiform/variant-call11.cf

```
hg38-tar = 'hg38/hg38.tar';

fastq1-gz =
  '1000genomes/SRR062634_1.filt.fastq.gz'
  '1000genomes/SRR062635_1.filt.fastq.gz';
fastq2-gz =
  '1000genomes/SRR062634_2.filt.fastq.gz'
  '1000genomes/SRR062635_2.filt.fastq.gz';

db = 'annodb/hg38db.tar';

deftask per-chromosome(
    vcf( File )
  : fa( File )
    [fastq1( File ) fastq2( File )] ) {

  bt2idx = bowtie2-build( fa: fa );
  fai = samtools-faidx( fa: fa );

  sam = bowtie2-align(
    idx:    bt2idx
    fastq1: fastq1
    fastq2: fastq2 );

  bam = samtools-view( sam: sam );

  sortedbam = samtools-sort( bam: bam );

  mpileup = samtools-mpileup(
    sortedbam: sortedbam
    fa:        fa
    fai:       fai );

  vcf = varscan( mpileup: mpileup );
}

deftask per-sample(
    fun exonicfun
  : <fa( File )> db( File )
    [fastq1( File ) fastq2( File )] ) {

  vcf = per-chromosome(
    fa:     fa
    fastq1: fastq1
    fastq2: fastq2 );

  fun exonicfun = annovar(
    vcf:      vcf
    db:       db
    buildver: 'hg38' );
}
```

In this workflow parallelism is exploited along two dimensions: (i) Each self-contained region in the reference genome can be processed individually and (ii) each sample can be processed individually. Consequently, the workflow interpreter performs a cross-product of reference regions and samples. This leads to a high degree of parallelism not only for read alignment, which is the computationally most expensive task, but also for all subsequent tasks. The cross product behavior needs no extra denotation in the task prototypes since it is the default behavior. Thus, we can exploit data parallelism in variant calling to execute the workflow in a parallel, distributed compute environment while reusing established tools.

```
fa = untar( tar: hg38-tar );
fastq1 = gunzip( gz: fastq1-gz );
fastq2 = gunzip( gz: fastq2-gz );

fun exonicfun = per-sample(
  fa:     fa
  fastq1: fastq1
  fastq2: fastq2
  db:     db );
fun exonicfun;
```

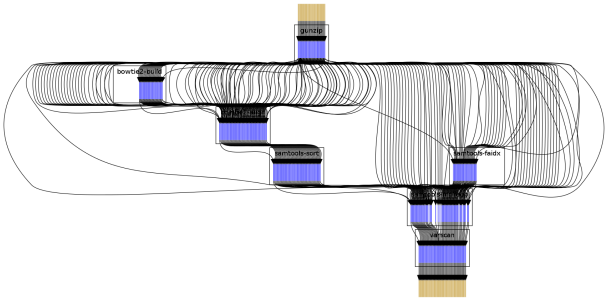


Figure 3: Invocation graph for variant calling workflow

5. EXECUTION PLATFORM

In this section we describe Hi-WAY, an execution environment for Cuneiform workflows. Cuneiform, as a workflow specification language, depends on an execution environment that executes tasks in parallel. To this end, the Cuneiform interpreter can either execute a script on a single, multi-threaded machine (using a simple built-in greedy task scheduler) or feed a distributed workflow engine. Currently, it interfaces only with Hi-WAY, a novel scientific workflow management system running on top of Apache Hadoop. Hi-WAY offers features like adaptive scheduling and, this way, embraces the dynamic nature of Cuneiform workflows. By using Hi-WAY as its distributed execution environment, Cuneiform takes advantage of the Hadoop ecosystem, including the distributed file system HDFS, multi-user resource management, job monitoring, and failure recovery. Details on Hi-WAY will be published in a separate publication.

As a proof-of-concept, the variant calling workflow described in Section 4.2 has been executed using Cuneiform and Hi-WAY on a Hadoop YARN cluster comprising 24 Xeon E52620 2GHz nodes each representing one Hadoop YARN container with 24GB main memory and 24 logical cores at its disposal (as well as 2 additional master nodes). 12 Samples from the 1000

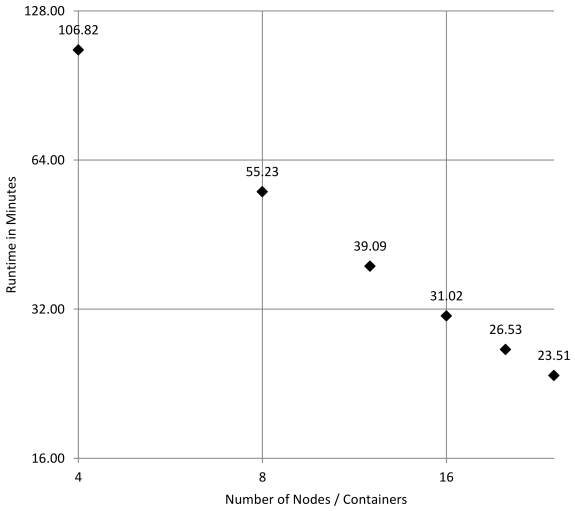


Figure 4: Workflow runtime with increasing number of containers

genomes project [38] amounting to 10GB of compressed input data have been processed. Figure 4 shows the runtime behaviour for the variant calling workflow for different cluster sizes. Within the limits of this experiment the workflow shows a linear scaling behaviour with an increasing number of available containers.

6. RELATED WORK

A number of scientific workflow systems have emerged, some with a particular focus on large scale data analysis. The exponential growth of data sets in many scientific areas, such as Next-Generation Sequencing (NGS), promotes scientific workflow systems that run in parallel and distributed environments, like e-Science Central [20], Pegasus [13], or Swift [42]. Some scientific workflow systems have been extended to this end, e.g., Kepler [41] or Galaxy [17]. These systems, however, either do not take full advantage of partitioning input data to exploit data parallelism or their integration with data-parallel compute platforms is only partial.

The advent of data-parallel query languages enabled researchers to exploit parallel, distributed compute infrastructures to analyze large-scale data sets. A number of dataflow systems with their according query languages have been proposed, most notably Pig [16, 33], FlumeJava [7], Flink [1], DryadLINQ [14, 43], and Spark [44, 45]. Their aptitude for NGS problems has been assessed [46] and they are the underlying execution environments for a number of emerging workflow systems like Nova [32] or Oozie [22]. However, the integration of external tools in these systems can be achieved only through wrapping or reimplementing the external tools. The speed-up potential from data parallelism has been exploited in many scientific application domains and particularly in NGS: CloudBurst [37] is a read alignment implementation for Hadoop. Crossbow [25] wraps the read aligner Bowtie [24] to run on Hadoop. Both Adam [27], an alignment processor, and Avocado [30], a variant caller, are algorithm reimplementations for Spark. The BioPig [29] project extends Pig Latin by providing User Defined Functions (UDFs) that wrap tools commonly used in NGS data analysis. These approaches show that it is feasible to integrate diverse scientific algorithms in data-parallel programming models either through wrapping or reimplementing. However, in use cases in which the cost for tool reimplementation is prohibitive (and in which the scientific community is very reluctant to accept algorithm reimplementations from outside their domain), the optimal programming model is one that minimizes the effort to create wrappers for existing tools.

Scientific workflow systems and data-parallel query languages are linked to Functional Programming (FP). Pig Latin maps execution plans to MapReduce, a programming model inspired by the algorithmic skeletons map and reduce which originate from FP [11, 15]. Spark extends Scala, a multi-paradigm language that combines concepts from Object Orientation and FP [31]. Furthermore, Scala provides a large number of algorithmic skeletons, of which Spark uses a subset including map, reduceByKey, and crossProduct to derive parallelism and distribute computation [44]. The scientific workflow language Taverna has its semantics defined in functional terms [40] and Kelly et al. [23] showed that scientific workflow languages can be considered a subset of FP. A number of FP languages are designed for parallel, distributed environments. For instance, Skywriting is a universal functional scripting language for distributed computation [28]. GUM [39] is a parallel implementation of Haskell and Eden [5, 26] extends Haskell with parallel algorithmic skeletons. However, in many parallel, distributed FP languages the user has to take control over the way parallelism is exploited, how processes are created, or how computation is distributed.

7. CONCLUSION

We presented Cuneiform[4], a functional workflow language for parallel and distributed execution that facilitates the reuse of existing tools and libraries. Cuneiform can process large-scale data sets by providing data parallel algorithmic skeletons operating on lists. Furthermore, it can integrate foreign tools in a straightforward way by providing a versatile Foreign Function Interface and offers many of the high-level language features commonly encountered in Functional Programming languages. We have contrasted the advantages and disadvantages of current scientific workflow languages and data-parallel query languages and discussed their relation to

[4] https://github.com/joergen7/cuneiform

Functional Programming. We demonstrated the versatility and power of Cuneiform using two exemplary workflows. In its current implementation Cuneiform can be executed locally on a single machine or using Hi-WAY[5], a scientific workflow execution environment running on Hadoop YARN. In future work, we intend to integrate Cuneiform with scientific computing platforms other than Hadoop like, e.g, HTCondor [9] which enjoys wide adoption. Furthermore, we intend to create compilers that consume Pegasus or Galaxy workflows and generate Cuneiform scripts. This way, researchers may run their existing Pegasus and Galaxy workflows in any data-parallel execution environment supporting Cuneiform without extra effort.

8. ACKNOWLEDGEMENTS

Jörgen Brandt and Marc Bux are funded by the European Commission's 7th Framework Programme (FP7) through the BiobankCloud project (project no. 317871).

9. REFERENCES

[1] http://flink.incubator.apache.org/.

[2] E. M. Bahsi, E. Ceyhan, and T. Kosar. Conditional workflow management: A survey and analysis. *Scientific Programming*, 15(4):283–297, 2007.

[3] M. Baker. Next-generation sequencing: adjusting to data overload. *Nature Methods*, 7(7):495–499, 2010.

[4] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel. Knime: The konstanz information miner. In *Studies in Classification, Data Analysis, and Knowledge Organization*, 2007.

[5] S. Breitinger, U. Klusik, and R. Loogen. From (sequential) haskell to (parallel) eden: An implementation point of view. In *Principles of Declarative Programming*, pages 318–334. Springer, 1998.

[6] M. Bux and U. Leser. Parallelization in scientific workflow management systems. *Computing Research Repository (CoRR)*, 2013.

[7] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. R. Henry, R. Bradshaw, and N. Weizenbaum. Flumejava: Easy, efficient data-parallel pipelines. *SIGPLAN Not.*, 45(6):363–375, 2010.

[8] M. I. Cole. *Algorithmic skeletons: structured management of parallel computation*. Pitman London, 1989.

[9] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger. Workflow management in condor, 2007.

[5] https://github.com/marcbux/Hi-WAY

[10] V. Curcin and M. Ghanem. Scientific workflow systems-can one size fit all? In *Cairo International Biomedical Engineering Conference (CIBEC)*, pages 1–9, 2008.

[11] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[12] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. In *Future Generation Computer Systems*, 2008.

[13] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, A. Laity, J. Jacob, and D. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13:219–237, 2005.

[14] J. Ekanayake, T. Gunarathne, G. Fox, A. S. Balkir, C. Poulain, N. Araujo, and R. Barga. Dryadlinq for scientific analyses, 2009.

[15] J. Ekanayake, S. Pallickara, and G. Fox. Mapreduce for data intensive scientific analyses. In *IEEE Fourth International Conference on eScience*, pages 277–284, 2008.

[16] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava. Building a high-level dataflow system on top of map-reduce: the pig experience. *Proc. VLDB Endow.*, 2(2):1414–1425, 2009.

[17] J. Goecks, A. Nekrutenko, J. Taylor, and T. G. Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8), 2010.

[18] R. R. Gullapalli, K. V. Desai, L. Santana-Santos, J. A. Kant, and M. J. Becich. Next generation sequencing in clinical medicine: Challenges and lessons for pathology and biomedical informatics. *Journal of pathology informatics*, 3, 2012.

[19] A. Heise, A. Rheinländer, M. Leich, U. Leser, and F. Naumann. Meteor/sopremo: An extensible query language and operator model. In *International Workshop on End-to-end Management of Big Data*, 2012.

[20] H. Hiden, P. Watson, S. Woodman, and D. Leahy. *e-Science Central: Cloud-based e-Science and its application to chemical property modelling*. Newcastle University, Computing Science, 2010.

[21] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:729–732, 2006.

[22] M. Islam, A. K. Huang, M. Battisha, M. Chiang,

S. Srinivasan, C. Peters, A. Neumann, and A. Abdelnur. Oozie: towards a scalable workflow management system for hadoop. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, 2012.

[23] P. M. Kelly, P. D. Coddington, and A. L. Wendelborn. Lambda calculus as a workflow model. *Concurrency and Computation: Practice and Experience*, 21(16):1999–2017, 2009.

[24] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg. Searching for snps with cloud computing. *Genome Biology*, 10(11), 2009.

[25] B. Langmead, C. Trapnell, M. Pop, S. L. Salzberg, et al. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3), 2009.

[26] R. Loogen, Y. Ortega-Mallén, and R. Peña-Marí. Parallel functional programming in eden. *Journal of Functional Programming*, 15(03):431–475, 2005.

[27] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, A. D. Joseph, and D. A. Patterson. Adam: Genomics formats and processing patterns for cloud scale computing. Technical report, EECS Department, University of California, Berkeley, 2013.

[28] D. G. Murray and S. Hand. Scripting the cloud with skywriting. *Proceedings of HotCloud*, (3), 2010.

[29] H. Nordberg, K. Bhatia, K. Wang, and Z. Wang. Biopig a hadoop-based analytic toolkit for large scale sequence data. *Bioinformatics*, 29(23):3014–3019, 2014.

[30] F. A. Nothaft, P. Jin, and B. Brown. avocado: A variant caller, distributed. Technical report, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 2013.

[31] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, and M. Zenger. An overview of the scala programming language. Technical report, École Polytechnique Fédérale de Lausanne, 2004.

[32] C. Olston, G. Chiou, L. Chitnis, F. Liu, Y. Han, M. Larsson, A. Neumann, V. B. N. Rao, S. Seth, C. Tian, T. Zicornell, and X. Wang. Nova: Continuous pig/hadoop workflows. In *SIGMOD*, 2011.

[33] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD*, pages 1099–1110, 2008.

[34] S. Pabinger, A. Dander, M. Fischer, R. Snajder, M. Sperk, M. Efremova, B. Krabichler, M. R. Speicher, J. Zschocke, and Z. Trajanoski. A survey of tools for variant analysis of next-generation genome sequencing data. *Briefings in bioinformatics*, 15(2):256–278, 2014.

[35] E. Pennisi. Will computers crash genomics? *Science*, 331(6018):666–668, 2011.

[36] P. V. Roy and S. Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, 2004.

[37] M. C. Schatz. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[38] N. Siva. 1000 genomes project. *Nature biotechnology*, 26(3):256–256, 2008.

[39] P. W. Trinder, K. Hammond, J. S. Mattson Jr, A. S. Partridge, and S. Peyton Jones. Gum: a portable parallel implementation of haskell. In *ACM SIGPLAN Notices*, volume 31, pages 79–88, 1996.

[40] D. Turi, P. Missier, C. Goble, D. De Roure, and T. Oinn. Taverna workflows: Syntax and semantics. In *IEEE International Conference on e-Science and Grid Computing*, pages 441–448, 2007.

[41] J. Wang, D. Crawl, and I. Altintas. Kepler + hadoop: A general architecture facilitating data-intensive applications in scientific workflow systems. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, 2009.

[42] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.

[43] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, and J. Currey. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI*, pages 1–14, 2008.

[44] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.

[45] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 2010.

[46] Q. Zou, X.-B. Li, W.-R. Jiang, Z.-Y. Lin, G.-L. Li, and K. Chen. Survey of mapreduce frame operation in bioinformatics. *Briefings in bioinformatics*, 2013.

A Spark-based workflow for probabilistic record linkage of healthcare data *

Robespierre Pita
Distributed Systems Lab.
Federal University of Bahia
Salvador, BA, Brazil
pierre.pita@gmail.com

Clicia Pinto
Distributed Systems Lab.
Federal University of Bahia
Salvador, BA, Brazil
cliciasp1@gmail.com

Pedro Melo
Distributed Systems Lab.
Federal University of Bahia
Salvador, BA, Brazil
pedronovaes@dcc.ufba.br

Malu Silva
Distributed Systems Lab.
Federal University of Bahia
Salvador, BA, Brazil
maludeleon89@gmail.com

Marcos Barreto
Distributed Systems Lab.
Federal University of Bahia
Salvador, BA, Brazil
marcoseb@dcc.ufba.br

Davide Rasella
Institute of Public Health
Federal University of Bahia
Salvador, BA, Brazil
davide.rasella@gmail.com

ABSTRACT

Several areas, such as science, economics, finance, business intelligence, health, and others are exploring big data as a way to produce new information, make better decisions, and move forward their related technologies and systems. Specifically in health, big data represents a challenging problem due to the poor quality of data in some circumstances and the need to retrieve, aggregate, and process a huge amount of data from disparate databases. In this work, we focused on Brazilian Public Health System and on large databases from Ministry of Health and Ministry of Social Development and Hunger Alleviation. We present our Spark-based approach to data processing and probabilistic record linkage of such databases in order to produce very accurate data marts. These data marts are used by statisticians and epidemiologists to assess the effectiveness of conditional cash transfer programs to poor families in respect with the occurrence of some diseases (tuberculosis, leprosy, and AIDS). The case study we made as a proof-of-concept presents a good performance with accurate results. For comparison, we also discuss an OpenMP-based implementation.

Categories and Subject Descriptors

J.1 [**Administrative data processing**]: Government; D.1.3 [**Concurrent Programming**]: Distributed programming.

*This work is partially sponsored by Federal University of Bahia (UFBA), with PIBIC and PRODOC grants, by FAPESB (PPSUS-BA 30/2013 grants), and by CNPq — Brazilian National Research Council.

1. INTRODUCTION

The term big data [18] was coined to represent the large volume of data produced daily by thousands of devices, users, and computer systems. These data should be stored in secure, scalable infrastructures in order to be processed using knowledge discovery and analytics tools. Today, there is a significant number of big data applications covering several areas, such as finance, entertainment, e-government, science, health etc. All these applications require performance, reliability, and accurate results from their underlying execution environments, as well as specific requisites depending on each context.

Healthcare data come from different information systems, disparate databases, and potential applications that need to be combined for diverse purposes, including the aggregation of medical and hospital services, analysis of patients' profile and diseases, assessment of public health policies, monitoring of drug interventions, and so on.

Our work focuses on the Brazilian Public Health System [23], specifically on supporting the assessment of data quality, pre-processing, and linkage of databases provided by the Ministry of Health and the Ministry of Social Development and Hunger Alleviation. The data marts produced by the linkage are used by statisticians and epidemiologists in order to assess the effectiveness of conditional cash transfer programs for poor families in relation to some diseases, such as leprosy, tuberculosis, and AIDS.

We present a four-stage workflow designed to provide the functionalities mentioned above. The second (pre-processing) and third (linkage) stages of our workflow are very data-intensive and time-consuming tasks, so we based our implementation in the Spark scalable execution engine [41] in order to produce very accu-

rate results in a short period of time. The first stage (assessment of data quality) is made through SPSS [17]. The last stage is dedicated to the evaluation of the data marts produced by our pre-processing and linkage algorithms and is realized by statisticians and epidemiologists. Once approved, they load these data marts into SPSS and Stata [6] in order to perform some specific case studies.

We evaluate our workflow by linking three databases: CadÚnico (social and economic data of poor families — approximately 76 million records), PBF (payments from "Bolsa Família" program), and SIH (hospitalization data from the Brazilian Public Health System — 56,059 records). We discuss the results obtained with our Spark-based implementation and also a comparison with an OpenMP-based implementation.

This paper is structured as follows: Section 2 presents the Brazilian Healthcare System to contextualize our work. In Section 3 we discuss some related works specially focusing on record linkage. Our proposed workflow is detailed in Section 4 and its Spark-based implementation is discussed in Section 5. We present results obtained from our case study, both in Spark and OpenMP, in Section 6. Some concluding remarks are presented in Section 7.

2. BRAZILIAN HEALTHCARE SYSTEM

As a strategy to combat poverty, the Brazilian government implemented cash transfer policies for poor families, in order to facilitate their access to education and healthcare, as well as to offer them allowances for consuming goods and services. In particular, the "Bolsa Família" Program [25] was created under the management of the Ministry of Social Development and Hunger Alleviation to support poor families and promote their social inclusion through income transfers.

Socioeconomic information about poor families are kept in a database called CadastroÚnico (CadÚnico) [24]. All families with a monthly income below half the minimum wage per person or a total monthly income of less than three minimum wages can be enrolled in the database. This registration must be renewed every two years in order to keep updated data. All social programs from the federal government should select their recipients based on data contained in CadÚnico.

In order to observe the influence of certain social interventions and their positive (or negative) effects for their beneficiaries, rigourous impact evaluations are required. Individual cohorts [19] have emerged as the primary method for this purpose, supporting the process of improving public policies and social programs in order to qualify the transparency of public investments. It is expected that these transfer programs can positively contribute to the health and the education of beneficiary families, but studies capable to prove this are highly desirable and necessary for the evaluation of public policies.

From an epidemiological standpoint, tuberculosis and leprosy are major public health problems in Brazil, with poverty as one of their main drivers. In addition, there is a broad consensus on the bidirectional relationship between these infectious diseases and poverty: one can lead to another. It is therefore clear that to reduce morbidity and mortality from poverty-related diseases is necessary to plan interventions that address their social determinants.

This work pertains to a project involving the longitudinal study of CadÚnico, PBF ("Bolsa Família" program), and three databases from the Brazilian Public Health System (SUS): SIH (hospitalization), SINAN (notifiable diseases), and SIM (mortality). Table 1 shows these databases with their years of coverage to which we have access. The main goal is to relate individuals in the existing SUS databases with their counterparts in the PBF and CadÚnico, through a process called linkage (or pairing). After linkage, the resulting databases (data marts) are used by statisticians and epidemiologists to analyze the incidence of some diseases in families benefiting from "Bolsa Família" compared to non-beneficiary families.

Databases	**Years**
SIH (hospitalization)	1998 to 2011
SINAN (notifications)	2000 to 2010
SIM (mortality)	2000 to 2010
CadÚnico (socieconomic data)	2007 to 2013
PBF (payments)	2007 to 2013

Table 1: Brazilan governmental databases.

The major obstacle for linkage is the absence of common identifiers (key attributes) in all databases, which requires the use of probabilistic linkage algorithms, resulting in a significant number of comparisons and in a large execution time. In addition, handling these databases requires the use of secrecy and confidentiality policies for personal information, especially those related to health data. Therefore, techniques for data transformation and anonymisation should be employed before the linkage stage.

The longitudinal study requires the pairing of all available versions for certain databases within the period to be analyzed. In the scope of our project, we must link versions of CadÚnico, PBF, and SIH between 2007 and 2011 to allow a retrospective analysis of the incidence of diseases in poor families and, thereafter, draw up prospects for the coming years. In this scenario, the amount of data to be analyzed, processed, and anonymised tends to increase significantly.

3. CHALLENGES AND RELATED WORK

Record linkage is not a new problem and its classic method was first proposed by [13]. This approach is the basis for most of the models developed later [5]. The basic idea is to use a set of common attributes present in records from different data sources in order to identify true matches.

In [32], probabilistic and deterministic record linkage methods were used to evaluate the impact of the "Bolsa Família" program in education, using some information also contained in CadÚnico. They have proven the importance of database relationships as a tool capable of allowing an integrated view of the information available from various sources, ensuring efficient comparative analysis and increasing the quality and quantity of information required for a search. In public health, many studies use matching records to evaluate impacts or to find patterns [27].

In [11], the authors used probabilistic methods to match records from two SUS databases — SIH (hospitalization) and SIM (mortality) — to identify deaths from ill-defined causes. They developed routines for standardizing variables, blocking based on identification keys, comparison algorithms, and calculation of similarity scores, They also used RecLink [4] to check dubious records for reclassification (as true pairs or not) purposes.

A crucial point is that as the size of databases increases, and therefore the number of comparisons required for record matching, traditional tools for data processing and analysis may not be able to run such applications in a timely manner. In the midst of several studies on software for record linkage, there are few that discuss issues related to the parallelization of processes and data distribution. In [33], some ways to parallelize matching algorithms are discussed, showing good scalability results.

MapReduce paradigm and following technologies have contributed to advance the big data scenario. Some methods to adapt the MapReduce model to deal with record matching are discussed in [16]. Despite these efforts, it is still difficult to find references addressing the problem of matching records using the advantages of MapReduce or similar tools.

Computation techniques related to the preparation steps for record linkage, such as data cleansing and standardization, are still few discussed in the literature. In [31], the authors claim that the cleansing process can represent 75% of the total linkage effort. In fact, preparation steps can directly affect the accuracy of results.

It is possible to observe that management and some aspects of service provision in this context are not yet sufficiently explored [22]. Regarding databases under coordination of public sectors, as CadÚnico and SUS databases, we can observe a high sensitivity and strict requirements for processing and storing such databases in private clusters. Also, there is a lack, mainly in Brazil, of probabilistic matching references over large databases that use the benefits of big data tools.

4. PROPOSED WORKFLOW

Our workflow is divided in four stages, further discussed in the following sections. The first stage corresponds to the analysis of data quality, aiming at to identify, for each database, the attributes more suitable for the probabilistic record linkage process. The set of attributes is chosen based on metrics such as missing values or misfiled records. This step is performed with the support of SPSS software. For security and privacy reasons, the ministries do not allow direct access to their databases; instead, they give us flat files extracted from the databases listed in Table 1. Two people of our team are the ones that manipulate these data based on a strict confidentiality term.

The next stage is pre-processing, being responsible for applying data transformation and cleansing routines in these attributes. We based our implementation on ETL (extract, transform, and load) techniques commonly found in data warehouse tools for standardizing names, filling null/missing fields with default values, and removing duplicate records.

An important step within this stage regards data privacy. We apply a technique based on Bloom filters [34] to anonymize relevant fields prior to the record linkage stage. As stated before, pre-processing is a time-consuming, data-intensive stage, so we use Spark to perform data transformation, cleansing, anonymization, and blocking.

The record linkage stage applies deterministic and probabilistic algorithms to perform pairing. Between CadÚnico and PBF databases, we can use a deterministic algorithm for record linkage based on a common attribute called NIS (social number ID). All beneficiaries of PBF are necessarily registered in CadÚnico and therefore have this attribute. Linkage between CadÚnico and any SUS database (SIH, SINAN, and SIM) must be done through probabilistic algorithms, since there are no common attributes to all databases.

Within SUS databases, the occurrence of incomplete records is quite high, since many records correspond to children or homeless people, which do not always have identification documents or are not directly registered in CadÚnico. In such cases, we try to find a record of an immediate family member, when available. Another very common problem regards incomplete or abbreviated names, which difficults pairing. Again, we use Spark to execute our linkage algorithms in a timely manner and produce the resulting data marts (files with matched and non-matched records).

The last stage is performed by statisticians and epi-

demiologists with the support of statistical tools (Stata and SPSS). The goal is to evaluate the accuracy of the data marts produced by the linkage algorithms, based on data samples from the databases involved. This step is extremely important to validate our implementation and provide some feedback for corrections and adjustments in our workflow.

In the following sections, we discuss a case study on the linkage of CadÚnico and SIH databases made as a proof-of-concept of our workflow. The goal was to generate a data mart covering such databases that is used to analyze the incidence of tuberculosis in PBF beneficiaries and non-beneficiaries families. We chose the databases of the year 2011, respectively with approximately 76 million records and 56,059 records.

4.1 Data Quality Assessment

Attributes suitable for probabilistic matching should be chosen taking into account their coexistence in all databases, their discriminatory capacity, and their quality in terms of filling requirements and constraints. The occurrence of null or missing values is the major problem considered at this stage. This problem can occur by omission or negligence of the operator responsible for filling out forms or by the faulty implementation of the involved information systems. The analysis of missing values is extremely important, because using a variable that has a high incidence of empty fields brings little or no benefit to the matching process.

In our case study, we analyzed the occurrence of null and missing values in the CadÚnico and SIH databases. Tables 2 and 3 show the results obtained for the most significant attributes in each database. Based on the results, we chose three attributes: NOME (person's or patient's name), NASC (date of birtyp), and MUNIC_RES (city of residence).

Attribute	Description	Missing (%)
NIS	Social number ID	0,7
NOME	Person's name	0
DT_NASC	Date of birth	0
MUNIC_RES	City of residence	55,4
SEXO	Gender	0
RG	General ID	48,7
CPF	Individual taxpayer ID	52,1

Table 2: CadÚnico — missing values.

4.2 Data Pre-processing

Datamarts produced in our case study are composed of linked information that reflect the pairing process output. They should contain information about people hospitalized in 2011 with a primary diagnosis of tuberculosis and their socioeconomic data, if registered in CadÚnico, relevant for epidemiological studies.

Attribute	Description	Missing (%)
MUNIC_RES	City of residence	0
NASC	Date of birth	0
SEXO	Gender	0
NOME	Patient's name	0
LOGR	Street name	0,9
NUM_LOGR	House number	16,4
COMPL_LOGR	Address' complement	80,7

Table 3: SIH — missing values.

To facilitate the linkage and increase accuracy, the values of NOME attribute are transformed to uppercase and accents (and possible pontuaction) are removed, so as not to influence the similarity degree between two records. Attributes with null or missing values are treated through a simple substitution to predefined values. This ensures all records are in the same format and contain the same information pattern.

A fundamental concern in our work is confidentiality. We must use privacy policies to guarantee that personal data is protected throughout the workflow. Linkage routines should not be able to identify any person in any database. To accomplish this, we use Bloom filters for record anonymization. Bloom filter is an approach that allows to check if an element belongs to a set. An advantage of this method is that it does not allow false negatives: if a record belongs to the set, the method always returns true. Furthermore, false positives (two records that do not represent the same entity) are allowed. This could be advantageous if the goal is to include records containing small differences in the matched pairs.

The construction of Bloom filters is described in [34] and involves a vector initially populated with 0's. Depending on each attribute, specific positions of this vector, determined by hash functions, are replaced with 1's. Our approach considers an array of 110 positions that maps each bigram (two characters) of the attributes involved. Each attribute affects a fraction of the vector: NOME comprises the first 50 bits, NASC comprises the following 40 bits, and MUNIC_RES the last 20 bits, as shown in Figure 1.

Fractions were chosen considering two aspects: the ideal size a filter must have in order to represent an attribute with a minimum probability error and the influence (or "weight") each field has in the matching decision. Accuracy depends on the filter size (and thus the weight of each attribute), the number of hash functions, and the number of elements added to the filter [36]. The smaller the filter, more errors and false positives are expected because different records can generate very similar vectors with 1's coincidentally mapped in same

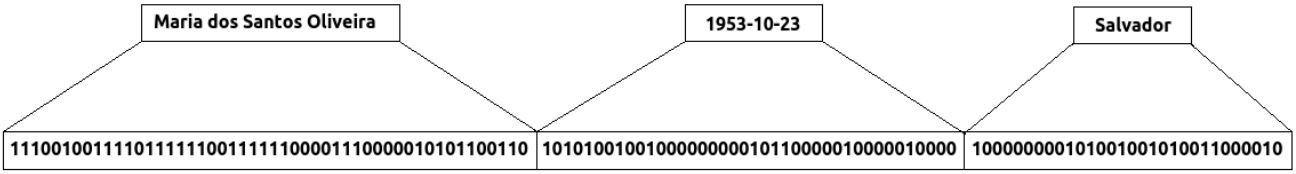


Figure 1: Bit vector generated by Bloom filter.

positions. So, there is a classic tradeoff between size and performance: the vector must be large enough to increase accuracy and, at the same time, small enough to not overload the similarity tests.

For testing our Bloom filter, we constructed three controlled scenarios and use two databases with 50 and 20 records, respectively. The idea was to determine the best vector size and the distribution (number of bits for each field) that provides the best accuracy. Table 4 shows our simulation results. In scenario 3, we simulated filling errors. We can observe that when one attribute has similarity index lower than the expected value, pairing can be saved by the other two attributes that have satisfactory similarity index.

Among all distributions that provide correct results, the distribution with 50, 40, and 20 bits is better for all scenarios. In this sense, the attribute MUNIC_RES must have less influence than NOME because the probability that the same value for NOME in different databases refers to the same person is more significant than two identical values for city.

Another important task performed during the preprocessing stage is blocking construction. The record linkage process requires all records from both databases be compared in order to determine whether they match or not. So, it demands M x N comparisons, being M and N the sizes of the databases. However, most of the comparisons will result in non-matched records.

In our case study, the number of comparisons between CadÚnico (approximately 76 million records) and SIH (56,059 records) could be quite prohibitively, so we decide to group records in each database according to a similarity criterion. We chose the MUNIC_RES (city of residence) attribute as blocking key, so that only individuals who live in the same city will be compared. As blocking strategies are a difficult problem, we are also considering another approaches such as adaptative blocking [2], predicates, and phonetic codes (such as Soundex [38], Metaphone [30], and BuscaBR [7]).

4.3 Calculation of Similarity

The decision on pairing two records depends on the analysis of their similarity factor. In this work, we use the Sørensen index [35], also known as Dice [9], to calculate the similarity based on bigrams (two characters) extracted from the bit vector generated by the Bloom filter.

Given a pair of records similar to those shown in Figure 1, the similarity test runs through every bit from both vectors in order to find three metrics: h — representing the count of 1's in the same position in both vectors, a and b — representing the total of 1's in the first and second vectors, respectively, regardless their positions. With these values, it is possible to calculate the Sørensen index using the following formula:

$$D_{a,b} = 2h \ / \ (|a| + |b|)$$

Perfect result expects the number of 1's contained in the first vector added to the second vector be exactly equal to twice the number of common 1's. When this happens, we have a result equal to 1 (great accuracy). If two records turn out different, the value of h decreases and the ratio starts to be smaller than 1.

We use a product by 10,000 to represent the Dice coefficient, therefore values ranging from 0 to 10,000 are used to represent the similarity degree between two vectors. Records are inserted in three distinct groups, as depicted in Figure 2. Every pair whose similarity degree is less than 9,000 is considered non-matched. Values between 9,000 and 9,600 are included in an indecison group for manual analysis, whereas values above 9,600 are considered true matches.

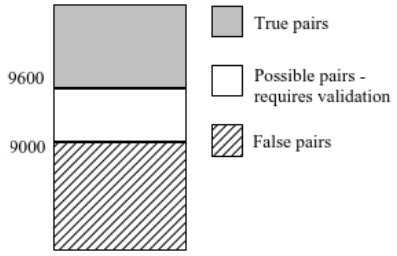


Figure 2: Similarity degrees for Dice calculation.

4.4 Record Linkage

Practical applications of record linkage exist in several areas. For the impact assessment of strategies, for example, it is often necessary to use individual search methods to prove if a specific situation happens in the

Total size and weight distribution	**Scenario 1** No matched records expected		**Scenario 2** Five perfectly matched records expected		**Scenario 3** Expected five matched records with one incorrect character	
	Expected pairings	Pairings found	Expected pairings	Pairings found	Expected pairings	Pairings found
20x20x20	0	310	5	347	5	348
30x30x30	0	29	5	41	5	42
40x40x40	0	11	5	17	5	16
50x50x50	0	0	5	5	5	5
50x50x40	0	0	5	5	5	5
50x40x40	0	0	5	5	5	5
50x40x30	0	0	5	5	5	5
50x30x30	0	2	5	6	5	6
50x40x20	0	0	5	5	5	5

Table 4: Comparison of different vector sizes and weight distributions.

whole group being analysed. Therefore, record linkage is a suitable method to follow cohorts of individuals by monitoring databases that contain continuous outcomes [32]. The interest group can be individually observed in order to obtain more accurate results or to identify variations in the characteristics of each individual. This situation is called a longitudinal study [19].

Probabilistic approaches can be used to match records without common keys from disparate databases. To succeed, we must use a set of attributes for which a probability of pairing can be set. This method requires a careful choice of the keys involved in matching or indeterminancy decisions [10]. This is the case, for example, of determining whether the records "Maria dos Santos Oliveira, Rua Caetano Moura, Salvador" and "Maria S. Oliveira, R. Caetano Moura, Salvador" refer to the same person. The main disadvantages of probabilistic approaches are their long execution times and the debug complexity they impose.

One of the big challenges in probabilistic record linkage is to link records with different schemas and get a good accuracy [11]. There are many problems that hinder pairing, such as abbreviations, different naming conventions, omissions, transcription, and gathering errors. Another big issue is scaling algorithms for large data sets. Transformation and similarity calculation are important challenges for the execution environment when scaled for large databases.

5. SPARK-BASED DESIGN ISSUES

The pioneering programming model capable of handling hundreds or thousands of machines in a cluster, providing fault tolerance, petascale computing, and high abstraction in building applications was MapReduce [8], further popularized by its open-source implementation provided by Hadoop [1]. Basically, this model proposed the division of the input data into splits that must be processed by threads, cores or machines in a cluster responsible for implementing map or reduce functions written by the developer. Intermediate data generated by the first phase are stored on the local disks of processing machines and are accessed remotely by machines performing reduce jobs.

Hadoop was responsible for driving a number of variations seeking to meet specific requirements. Hive [37], Pig [28], GraphLab [20], and Twister [12] are examples of initiatives classified as "beyond Hadoop" [26], which basically keep the MapReduce paradigm but intend to generate new levels of abstractions. However, some authors have indicated significant lacks in MapReduce specially for applications that need to process large volumes of data with strong requirements regarding iterations, machine learning or even with different performance requisites. New frameworks classified as "beyond MapReduce", such as Dremel [21], Jumbo [15], Shark [39], and Spark [41], were created to deal with these new requirements.

Spark is a framework that allows the design of applications based on working sets, the use of some general-purpose languages (such as Java, Scala, and Python), in-memory data processing and a new data distribution model called RDD (resilient distributed dataset) [40]. RDD is a collection of read-only objects partitioned across a set of machines that can be rebuilt if a partition is lost.

The main benefits of using Spark are related to the creation of a RDD for a dataset that must be processed. There are two basic ways to create a RDD, both use the *SparkContext* class: parallelizing a vector of iterable items created at runtime or referencing a dataset in an external storage system (such as a shared filesystem), HDFS [3], HBase [14], or any data source offering a

Hadoop-like InputFormat interface [41].

RDDs can be used through two classes of basic operations: *transformations*, which creates a new dataset from an existing one; and *actions*, which returns a value to the driver program after running a computation on the dataset. The first class is implemented using lazy evaluation and is intended to provide better performance and management of large data sets. Transformations are only computed when an action requires a value to be returned to the driver program [41]. Table 5 shows the main features of Spark framework we used to implement our probabilistic record linkage algorithms.

Transformation	**Meaning**
map(func)	Returns a new RDD by passing each element of the source through *func*
mapPartitions(func)	Similar to map, but runs separately on each partition (block) of the RDD.
Action	**Meaning**
collect()	Returns all the elements of the dataset as an array at the driver program.
count()	Returns the number of elements in the dataset.

Table 5: RDD API used for record linkage.

Another advantage of Spark is its ability to perform tasks in-memory. Views generated during execution are kept in memory, avoiding the storage of intermediate data on hard disks. Spark's developers claim that it is possible to reduce the execution time up to 100 times thanks to the use of working sets, and up to 10 times if hard disks are used. So, our choice to use Spark is justified by its performance, scalability, RDD's fault tolerance, and a very comfortable learning curve due to its compatibility with different programming languages.

The pre-processing stage follows Algorithm 1, which shows how this flow is implemented by the processing of input data transformations using map functions calling other procedures. The intention is that the function *map(blocking)* starts running as *map(normalize)* delivers its results; so we use the *collect()* action to ensure this. It is important to highlight the use of the *cache()* function that fits the memory with the splits extracted from the input files.

Algorithm 1 PreProcessing

```
1: Input ← OriginalDatabase.csv
2: Output ← TreatedDatabaseAnom.bloom
3: InputSparkC ← sc.textFile(Input)
4: NameSize ← 50
5: BirthSize ← 40
6: CitySize ← 20
7: ResultBeta ← InputSparkC.cache().map(normalize)
8: Result ← ResultBeta.cache().map(blocking).collect()
9: for line in Result:
10: write line in Output
11: procedure NORMALIZE(rawLine)
12:     splitedLine ← rawLine.split(;)
13: for fields in splitedLine:
14:     field ← field.normalized(UTF8) return splitedLine.join(;)
15: procedure BLOCKING(treatedLine)
16:     splLine ← treatedLine.split(;)
17:     splLine[0] ← applyBloom(splLine[0], NameSize)
18:     splLine[1] ← applyBloom(splLine[1], BirthSize)
19:     splLine[2] ← applyBloom(splLine[2], CitySize)
    return splitedLine.join()
20: procedure APPLYBLOOM(field, vectorSize)
21:     instanceInitialVectorWithSize ← vectorSize
22: for n-grams in field:
23:     bitsVector ← Calculate positions of 1s in Vector
    return bitsVector
```

Algorithm 2 Record linkage

```
1: InputMinor ← TreatedDatabaseAnom1.bloom
2: InputLarger ← TreatedDatabaseAnom2.bloom
3: InputSC1 ← sc.textFile(InputMinor)
4: InputSC2 ← sc.textFile(InputLarger)
5: var ← InputSC1.cache().collect()
6: varbc ← sc.broadcast(var)
7: InterResult ← InputSC2.cache().map(compare)
8: Result ← InputSC2.cache().collect()
9: for line in recordLinkageResult:
10: write line in Output
11: procedure COMPARE(line)
12: for linebc in varbc.value:
13:     get Dice index of (linebc) and (line) comparison
14:     decide about the similarity
15:     if Dice = 9000 then return line
16:     else return None
```

Algorithm 2 shows our record linkage flow. We use a RDD object, since it is read-only, to map the smallest database (SIH). We also use a shared variable, called *broadcast* by Spark, to give every node a copy of the largest database (CadÚnico) in an efficient manner, preventing communication costs, file loads, and split management. A comparsion procedure calculates the Dice index and decides about matching.

6. PERFORMANCE EVALUATION

In order to evaluate the proposed workflow, we ran out our Spark implementation on a cluster with 8 processors Intel Xeon E74820, 16 cores, 126 GB of RAM and a storage machine with up to 10 TB disks connected by the NFS protocol. We compared this implementation with our OpenMP version of the same workflow, also considering other multicore machines: an i5 processor with 4 GB of RAM and 300 GB of hard disk and an i7 processor with 32 GB of RAM and 350 GB of hard disk.

6.1 Spark

For Spark, we chose three samples from CadÚnico and SIH databases, each representing all the cities from the states of Amapá (Sample A), Sergipe (Sample B), and Tocantins (Sample C). These samples represent the smallest Brazilian states in terms of number of records in CadÚnico. Based on them, we can get an idea of the number of comparisons and the rise in the execution time in each case, as shown in Table 6.

Sample Name	Size (in lines) CadÚnico x SIH	Comparisons (millions)	Exec. Time (seconds)
A	367,892 x 147	54,0	96,26
B	1,6 mi x 171	289,5	479
C	1,02 mi x 389	397,63	656,79

Table 6: Spark results for record linkage.

These preliminary results are very promising if we consider the possibility of scaling up the number of machines involved in data processing. Table 7 details the time spent in each stage of the workflow. Standardization, anonymization, and blocking stages are detailed by Algorithm 1 and take only a few minutes in the larger database, while the similarity test and the decision on pairing require a longer running time. The last step consists in recovering a pair of linked records for creating a data mart. Together, all steps do not take more than 12 hours of execution.

	CadÚnico	**SIH**
Size (lines)	approx. 87 mi	approx. 61 k
Standardization	2310.4 s	36.5 s
Anonymization		
Blocking		
Record Linkage	9,03 hours	
Paired Recovery	1,31 hours	

Table 7: Execution time within the workflow.

6.2 OpenMP

The OpenMP interface [29] was chosen due to its syntax simplicity. This kind of implementation divides a task between threads that execute simultaneously, distributed through processors or functional units. The OpenMP API supports C, C++ and Fortran program languages. The C language was chosen because of its worldwide understanding.

The database sets used for this implementation were files containing the results from the Bloom filter applied during the pre-processing stage. These files have N bits in each line (record). The goal is to make the record linkage by calculating the Dice coefficient for each pair of records and writing out the positive Dice results and its respectives lines in an output file.

Access to the database sets in C language is made through pointer types. When a parallel region of the code is initialized, it is necessary to specify global and local variables to the threads. If a pointer is global to the threads, there is a race condition problem if they try to access different positions from the same file. This problem was solved by making these pointers private to each thread. As it is not possible to pass pointer types (only native types), they are created for every line (record) from one of the files. As the files have always the same N bits in each line, it is possible to specify each thread to access uniquely some lines from these files.

Tools	**i5**	**i7**	Cluster
Spark	507.5 s	235.7 s	96.26 s
OpenMP	104.9 s	65.5 s	13.36 s

Table 8: OpenMP x Spark metrics (Sample A).

Table 8 shows the execution time achieved by OpenMP for our Sample A. The machines we used have the following configuration: i5 (4 cores, 8 execution threads), i7 (8 cores, 16 execution threads). The cluster has been described in section 6. The execution time over i7 processor was 37% shorter than i5, showing that the execution time could be even shorter when using computers with more threads per core. Despite its shorter execution time, this approach does not provide a number of advantages offered by Spark, such as scalability and fault tolerance.

The number of matching record was 245. The results were very satisfactory, taking into account that the application runs in only one computer. This shows that the OpenMP implementation is indicated for small linkages or bigger linkages blocked by smaller parts. We are also considering the use of OpenMP for generating the Bloom filter and grouping records to compose the data marts.

7. CONCLUDING REMARKS

The development of a computational infrastructure to support projects focusing on big data from health

systems, like the case study discussed here, was motivated by two factors. First, the need to provide a tool capable of link disparate databases with socioeconomic and healthcare data, serving as a basis for decision-making processes and assessment of the effectiveness of governmental programs. Second, the availability of recent tools for big data processing and analytics, such as those mentioned in this work, with interesting capabilities to deal with new requirements imposed by the applications.

Among the available tools, we chose Spark due to its in-memory facility, its scalability, and ease of programming. Our preliminary tests present very promising results, reinforcing the need for some adjustments in our implementation. New features recently included in Spark could help us, such as the SparkR extension for data quality assessment. We are also testing other techniques throughout the workflow, like phonetic codes, predicates (for blocking) and multi-bit trees.

We plan to continue our tests with OpenMP in order to identify scenarios for which it can provide good performance. The exploration of hybrid architectures (multicore + multi-GPUs) is also in our roadmap.

The execution platform developed in this work represents a major advance in the face of existing solutions for record linkage in Brazil. It will serve as a basis architecture for the installation of a Referral Center for Probabilistic Linkage, and should be supplemented with new features regarding privacy, security, storage, among others.

8. REFERENCES

[1] Apache. Apache Hadoop, 2014. [Online; accessed 12-december-2014].

[2] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: learning to scale up record linkage. In *ICDM*, pages 87–96. IEEE Computer Society, 2006.

[3] D. Borthakur. HDFS architecture guide. *Hadoop Apache Project, http://hadoop. apache. org/common/docs/current/hdfs_design.pdf*, 2008.

[4] K. R. Camargo Jr. and C. M. Coeli. RecLink: aplicativo para o relacionamento de bases de dados, implementando o método *probabilistic record linkage*. *Cadernos de Saúde Pública*, 16:439 – 447, 06 2000.

[5] P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer, 1st edition, 2012.

[6] S. Corporation. Stata — data analysis and statistical software, 2014. [Online; accessed 12-december-2014].

[7] F. J. T. de Lucena. Busca fonética em português do Brasil, 2006. [Online; accessed 13-december-2014].

[8] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[9] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3), 1945.

[10] A. Doan, A. Halevy, and Z. Ives. *Principles of data integration*. Morgan Kaufmann. Morgan Kaufmann, 2012.

[11] C. L. dos Santos Teixeira, C. H. Klein, K. V. Bloch, and C. M. Coeli. Reclassificação dos grupos de causas prováveis dos óbitos de causa mal definida, com base nas Autorizações de Internação Hospitalar no Sistema Único de Saúde, Estado do Rio de Janeiro, Brasil. *Cad. Saúde Pública*, 22(6):1315–1324, 2006.

[12] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative MapReduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[13] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[14] L. George. *HBase: the definitive guide*. " O'Reilly Media, Inc.", 2011.

[15] S. Groot and M. Kitsuregawa. Jumbo: beyond MapReduce for workload balancing. In *36th International Conference on Very Large Data Bases, Singapore*, 2010.

[16] Y. Huang. Record linkage in an Hadoop environment. Technical report, School of Computing, National University of Singapore, 2011.

[17] IBM. SPSS software — predictive analytics software and solutions, 2014. [Online; accessed 12-december-2014].

[18] IBM, P. Zikopoulos, and C. Eaton. *Understanding big data: analytics for enterprise class Hadoop and streaming data*. McGraw-Hill Osborne Media, 1st edition, 2011.

[19] K. M. Keyes and S. Galea. *Epidemiology matters: a new introduction to methodological foundations*. Oxford University Press, 1st edition, 2014.

[20] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: a new framework for parallel machine learning. *arXiv preprint arXiv:1006.4990*, 2010.

[21] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: interactive analysis of Web-scale datasets. In *Proc. of the 36th Int'l Conf on Very Large Data Bases*, pages 330–339, 2010.

[22] I. M. Merelli, H. P'erez-Snchez, S. Gesing, and

D. DAgostino. Managing, analysing, and integrating big data in medical bioinformatics: open problems and future perspectives. *BioMed Research International*, 2014(1), 2014.

[23] Ministério da Saúde. Portal da Saúde — Sistema Único de Saúde, 2014. [Online; accessed 12-december-2014].

[24] Ministério do Desenvolvimento Social e Combate à Fome (MDS). Cadastro único para programas sociais do governo federal, 2014. [Online; accessed 12-december-2014].

[25] Ministério do Desenvolvimento Social e Combate à Fome (MDS). Programa Bolsa Família, 2014. [Online; accessed 12-december-2014].

[26] G. Mone. Beyond hadoop. *Communications of the ACM*, 56(1):22–24, 2013.

[27] H. B. Newcombe. *Handbook of record linkage: methods for health and statistical studies, administration, and business*. Oxford University Press, 1st edition, 1988.

[28] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[29] OpenMP.org. The OpenMP API specification for parallel programming, 1998. [Online; accessed 13-december-2014].

[30] L. Philips. Hanging on the Metaphone. *Computer Language*, 7(12), 1990.

[31] S. M. Randall, A. M. Ferrante, and J. Semmens. The effect of data cleaning on record linkage quality. *BMC Medical Informatics and Decision Making*, 13, 06 2013.

[32] J. Romero. Utilizando o relacionamento de bases de dados para avaliação de políticas públicas: uma aplicação para o Programa Bolsa Família. Doutorado, UFMG/Cedeplar, 2008.

[33] W. Santos. Um algoritmo paralelo e eficiente para o problema de pareamento de dados. Mestrado, Universidade Federal de Minas Gerias, 2008.

[34] R. Schnell, T. Bachteler, and J. Reiher. Privacy-preserving record linkage using Bloom filters. *BMC Medical Informatics and Decision Making*, 9:41, 2009.

[35] T. Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. *Kongelige Danske Videnskabernes Selskab*, 5(4), 1948.

[36] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz. Theory and practice of Bloom filters for distributed systems. *IEEE Communications Surveys and Tutorials*, 14(1), 2012.

[37] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[38] U.S. National Archives and Records Administration. The soundex indexing system, 2007. [Online; accessed 13-december-2014].

[39] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: SQL and rich analytics at scale. In *Proceedings of the 2013 international conference on Management of data*, pages 13–24. ACM, 2013.

[40] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[41] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

Communication Cost in Parallel Query Processing

Dan Suciu, University of Washington

ABSTRACT

Fix a full, conjunctive query, and consider the following problem: what is the amount of communication required to compute the query in parallel, on p servers, over a large database instance? We define the Massively Parallel Communication (MPC) model, where the computation proceeds in rounds consisting of local computations followed by a global reshuffling of the data. Servers have unlimited computational power and are allowed to exchange any data, the only cost parameters are the number of rounds and the maximum amount of communication per server. I will describe tight bounds on the amount of communication for the case of a single round and data without skew, then discuss extensions to skewed data and multiround.

This is joint work with Paul Beame and Paris Koutris

Short Bio

Dan Suciu is a Professor in Computer Science at the University of Washington. He received his Ph.D. from the University of Pennsylvania in 1995, was a principal member of the technical staff at AT&T Labs and joined the University of Washington in 2000. Suciu is conducting research in data management, with an emphasis on topics related to Big Data and data sharing, such as probabilistic data, data pricing, parallel data processing, data security. He is a co-author of two books Data on the Web: from Relations to Semistructured Data and XML, 1999, and Probabilistic Databases, 2011. He is a Fellow of the ACM, holds twelve US patents, received the best paper award in SIGMOD 2000 and ICDT 2013, the ACM PODS Alberto Mendelzon Test of Time Award in 2010 and in 2012, the 10 Year Most Influential Paper Award in ICDE 2013, the VLDB Ten Year Best Paper Award in 2014, and is a recipient of the NSF Career Award and of an Alfred P. Sloan Fellowship. Suciu serves on the VLDB Board of Trustees, and is an associate editor for the VLDB Journal, ACM TWEB, and Information Systems and is a past associate editor for ACM TODS and ACM TOIS. Suciu's PhD students Gerome Miklau and Christopher Re received the ACM SIGMOD Best Dissertation Award in 2006 and 2010 respectively, and Nilesh Dalvi was a runner up in 2008.

Assignment of Different-Sized Inputs in MapReduce*

Foto Afrati†
School of Electrical and Computing Engineering
National Technical University of Athens, Greece
afrati@softlab.ece.ntua.gr

Shlomi Dolev‡
Department of Computer Science
Ben-Gurion University of the Negev, Israel
dolev@cs.bgu.ac.il

Ephraim Korach
Department of Industrial Engineering and Management
Ben-Gurion University of the Negev, Israel
korach@bgu.ac.il

Shantanu Sharma
Department of Computer Science
Ben-Gurion University of the Negev, Israel
sharmas@cs.bgu.ac.il

Jeffrey D. Ullman
Department of Computer Science
Stanford University
USA
ullman@cs.stanford.edu

ABSTRACT

A MapReduce algorithm can be described by a *mapping schema*, which assigns inputs to a set of reducers, such that for each required output there exists a reducer that receives all the inputs that participate in the computation of this output. Reducers have a capacity, which limits the sets of inputs that they can be assigned. However, individual inputs may vary in terms of size. We consider, for the first time, mapping schemas where input sizes are part of the considerations and restrictions. One of the significant parameters to optimize in any MapReduce job is communication cost between the map and reduce phases. The communication cost can be optimized by minimizing the number of copies of inputs sent to the reducers. The communication cost is closely related to the number of reducers of constrained capacity that are used to accommodate appropriately the inputs, so that the requirement of how the inputs must meet in a reducer is satisfied. In this work, we consider a family of problems where it is required that each input meets with each other input in at least one reducer. We also consider a slightly different family of problems in which, each input of a set, X, is required to meet each input of another set, Y, in at least one reducer. We prove that finding an optimal mapping schema for these families of problem is NP-hard, and present several approximation algorithms for finding a near optimal mapping schema.

*More details appear in [1].

†Supported by the project Handling Uncertainty in Data Intensive Applications, co-financed by the European Union (European Social Fund) and Greek national funds, through the Operational Program "Education and Lifelong Learning," under the program THALES

‡Supported by the Rita Altura Trust Chair in Computer Sciences, Lynne and William Frankel Center for Computer Sciences, Israel Science Foundation (grant 428/11), the Israeli Internet Association, and the Ministry of Science and Technology, Infrastructure Research in the Field of Advanced Computing and Cyber Security.

1. INTRODUCTION

MapReduce (was introduced by Dean and Ghemawat [6]) is a programming system used for parallel processing of large-scale data. Input data is processed by the *map phase* that applies a user-defined map function to produce intermediate data (of the form $\langle key, value \rangle$). Afterwards, intermediate data is processed by the *reduce phase* that applies a user-defined reduce function to keys and their associated values. The final output is provided by the reduce phase. A detailed description of MapReduce can be found in Chapter 2 of [11].

Reducers and Reducer Capacity. An important parameter to be considered in MapReduce algorithms is the "reducer capacity." A *reducer* is an application of the reduce function to a single key and its associated list of $value$s. The *reducer capacity* is an upper bound on the sum of the sizes of the $value$s that are assigned to the reducer. For example, we may choose the reducer capacity to be the size of the main memory of the processors on which the reducers run. We always assume in this paper that all the reducers have an identical capacity, denoted by q.

The term *reducer capacity* is introduced, here, for the first time. There are various works in the field of MapReduce algorithms design (*e.g.*, [10, 13, 2, 7, 12, 3]); none of them considers the reducer capacity.

Motivation and Examples. We demonstrate a new aspect of the reducer capacity in the scope of several special cases. One useful special case is where an output depends on *exactly* two inputs. We present two examples where each output depends on exactly two inputs and define two problems that are based on these examples.

Similarity-join. Similarity-join is used to find the similarity between any two inputs, *e.g.*, Web pages or documents. A set of m inputs (*e.g.*, Web pages) $WP = \{wp_1, wp_2, \ldots, wp_m\}$, a similarity function $sim(x, y)$, and a similarity threshold t are given, and each pair of inputs $\langle wp_x, wp_y \rangle$ corresponds to one output such that $sim(wp_x, wp_y) \geq t$.

It is necessary to compare all-pairs of inputs when the similarity measure is sufficiently complex that shortcuts like locality-sensitive hashing are not available. Therefore, it is mandatory that every two inputs (Web pages) of the given input

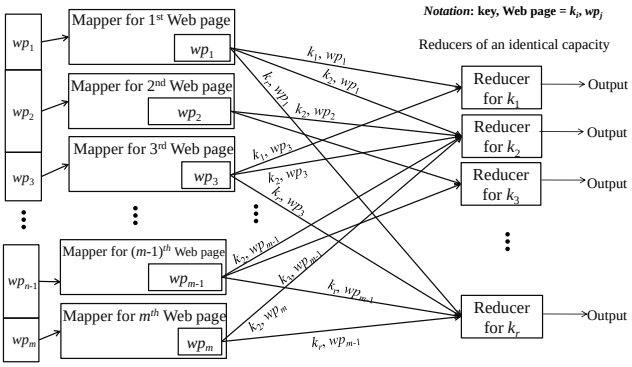


Figure 1: Similarity-join example.

set (WP) are compared. The similarity-join is useful in various applications, mentioned in [4], *e.g.*, near-duplicate document detection and collaborative filtering.

In Figure 1, an example of similarity-join is given as it is applied to Web pages. We are given a set of m Web pages, and a mapper (a mapper is an application of the map function to a single input) would take only a single Web page, and a reducer produces pairs of every two Web pages and their similarity score.

Skew join of two relations $X(A, B)$ and $Y(B, C)$. The join of relations $X(A, B)$ and $Y(B, C)$, where the joining attribute is B, provides the output tuples $\langle a, b, c\rangle$, where (a, b) is in X and (b, c) is in Y. One or both of the relations X and Y may have a large number of tuples with the same B-value. A value of the joining attribute B that occurs many times is known as a *heavy hitter*. In skew join of $X(A, B)$ and $Y(B, C)$, all the tuples of both the relations with the same heavy hitter should appear together to provide the output tuples.

In Figure 2, b_1 is considered as a heavy hitter, hence, it is required that all the tuples of $X(A, B)$ and $Y(B, C)$ with the heavy hitter, b_1, should appear together to provide the desired output tuples, $\langle a, b_1, c\rangle$ ($a \in A, b_1 \in B, c \in C$), which depend on exactly two inputs. It is worth noting that all the tuples of both the relations that have a common value of the joining attribute B, except b_1, are now also required to appear together to provide the remaining output tuples.

Problem Statement. We define two problems where exactly two inputs are required for computing an output, as follows:

All-to-All problem. In the *all-to-all* (*A2A*) problem, a set of inputs is given, and each pair of inputs corresponds to one output. Computing common friends on a social networking site, similarity-join [4, 15, 14], the drug-interaction problem [13], and the Hamming distance 1 problem [2] are examples of tasks for which an output depends on exactly two inputs, and the set of outputs requires us to consider each pair of inputs.

X-to-Y problem. In the *X-to-Y* (*X2Y*) problem, two disjoint sets X and Y are given, and each pair of elements $\langle x_i, y_j\rangle$, where $x_i \in X, y_j \in Y, \forall i, j$, of the sets X and Y corresponds to one output. Skew join and outer product or tensor product are examples.

The *communication cost*, *i.e.*, the total amount of data transmitted from the map phase to the reduce phase, is a significant factor in the performance of a MapReduce algorithm. The communication cost comes with tradeoff in the degree of parallelism, however.

A reducer of large enough capacity can be used to accommodate all the given inputs, and provide the desired outputs. This results in the minimum communication cost but also in the minimum parallelism. Higher parallelism requires more

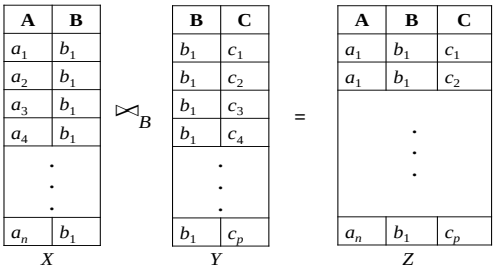


Figure 2: Skew join example for a heavy hitter, b_1.

reducers (hence, of smaller reducer capacity), and hence a larger communication cost (because the copies of the given inputs are required to be assigned to more reducers).

A substantial level of parallelism can be achieved with fewer reducers, and hence, yield a smaller communication cost. Thus, we focus on minimizing the total number of reducers, for a given reducer capacity q. A smaller number of reducers results in a smaller communication cost. Thus, the reducer capacity, q, reflects also the degree of parallelism we want, since if we want more parallelism we can explore the problem in question for smaller q.

Related Work. Afrati et al. [2] presents a model for MapReduce algorithms where an output depends on two inputs, and shows a tradeoff between communication cost and parallelism. In [3], the authors consider the case where each pair of inputs produces an output and present an upper bound that meets the lower bound on communication cost as a function of the total number of inputs sent to a reducer. However, both in [2] and [3] the authors regard the reducer capacity in terms of the total number of inputs (assuming each input is of an identical size) sent to a reducer. Our setting is closely related to the settings given by Afrati et al. [2] but we allow the input sizes to be different. Thus, we consider a more realistic setting for MapReduce algorithms that can be used in various practical scenarios.

Our Contribution. In this paper, we provide:

- Mapping schemas for the *A2A* and the *X2Y* problems, which take into account the fact that inputs have different sizes, while all the reducers have an identical and fixed capacity (Section 2).
- A tradeoff between the reducer capacity and the total number of reducers, which is demonstrated using similarity-join and skew join (Section 2). A tradeoff between the reducer capacity and the parallelism at the reduce phase, and a tradeoff between the reducer capacity and the communication cost is detailed in Section 2 as well.
- A proof that the *A2A mapping schema problem* for one and two reducers has a polynomial solution, and the same problem is NP-hard in the case of more than two reducers of an identical capacity (Section 3). Also, we prove that the *X2Y mapping schema problem* for one reducer has a polynomial solution, and the same problem is NP-hard in the case of more than one reducer of an identical capacity (Section 3).
- A set of heuristics, for the *A2A mapping schema problem* and the *X2Y mapping schema problem*, that is based on First-Fit Decreasing (FFD) or Best-Fit Decreasing (BFD) bin-packing algorithm, and a pseudo polynomial bin-packing algorithm (Sections 4 and 5).

2. MAPPING SCHEMA AND TRADEOFFS

Our system setting is an extension of the standard system setting [2] for MapReduce algorithms, where we consider, for the first time, inputs of different sizes. The system setting is suitable for a variety of problems where *exactly* two inputs are required for an output. To demonstrate the influence of the extra considerations, we define

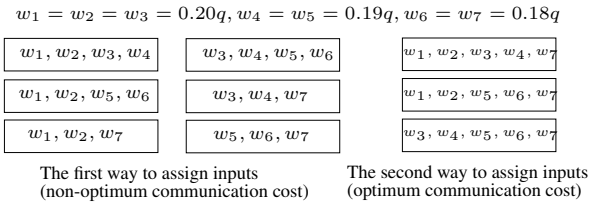


Figure 3: An example to the *A2A mapping schema problem*.

mapping schema and consider the communication cost tradeoff, as we elaborate next.

Mapping Schema. A mapping schema is an assignment of the set of inputs to some given reducers under the following two constraints:

• A reducer is assigned inputs whose sum of the sizes is less than or equal to the reducer capacity q.

• For each output, we must assign the corresponding inputs to at least one reducer in common.

Tradeoffs. The following tradeoffs appear in MapReduce algorithms and in particular in our setting:

• A tradeoff between the reducer capacity and the total number of reducers. For example, large reducer capacity allows the use of a smaller number of reducers.

• A tradeoff between the reducer capacity and the parallelism. For example, large reducer capacity results in less parallelism.

• A tradeoff between the reducer capacity and the communication cost.

In the subsequent subsections, we present two types of mapping schema problems with fitting examples and explain the three tradeoffs.

2.1 The A2A Mapping Schema Problem

The *A2A mapping schema problem* is defined in terms of a set of *inputs*, a size for each input, a set of reducers, and a mapping from outputs to sets of inputs. An instance of the *A2A mapping schema problem* consists of a set of m inputs whose input size set is $W = \{w_1, w_2, \ldots, w_m\}$ and a set of z reducers of capacity q. A solution to the *A2A mapping schema problem* assigns every pair of inputs to at least one reducer in common, without exceeding q at any reducer.

Example. We are given a set of seven inputs $I = \{i_1, i_2, \ldots, i_7\}$ whose size set is $W = \{0.20q, 0.20q, 0.20q, 0.19q, 0.19q, 0.18q, 0.18q\}$ and reducers of capacity q. In Figure 3, we show two different ways that we can assign the inputs to reducers. The best we can do to minimize the communication cost is to use three reducers. However, there is less parallelism at the reduce phase as compared to when we use six reducers. Observe that when we use six reducers, then all reducers have a lighter load, since each reducer may have capacity less than $0.8q$.

Explanation of tradeoffs. Similarity-join is an example of the *A2A mapping schema problem*, and the tradeoffs can also be explained with the help of similarity-join example. Consider that m Web pages are of $w_1, w_2, \ldots, w_m$ sizes. A single reducer of capacity $q = w_1 + w_2 + \ldots + w_m$ is able to find the similarity between every pair of Web pages. The use of only one reducer results in no parallelism at the reduce phase. But at the same time, the use of a single reducer yields the minimum possible communication cost. On the other hand, in case q is small but is still greater than or equal to $w_i + w_j$, for any i and j, then more reducers are required, and a higher level of parallelism is obtained. But, at the same time, the communication cost is higher, since every input is communicated to $m - 1$ reducers.

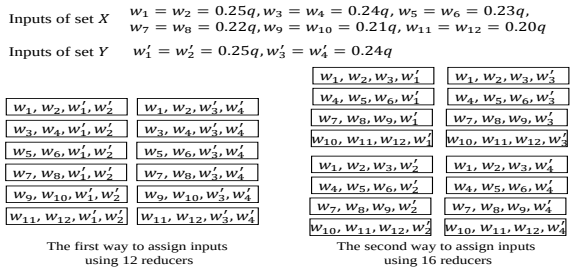


Figure 4: An example to the *X2Y mapping schema problem*.

2.2 The X2Y Mapping Schema Problem

The *X2Y mapping schema problem* is defined in terms of two disjoint sets X and Y of *inputs*, a size for each input, a set of reducers, and a mapping from outputs to sets of inputs. An instance of the *X2Y mapping schema problem* consists of two disjoint sets X and Y and a set of reducers of capacity q. The inputs of the set X are of sizes $w_1, w_2, \ldots, w_m$, and the inputs of the set Y are of sizes $w'_1, w'_2, \ldots, w'_n$. A solution to the *X2Y mapping schema problem* assigns every two inputs, the first from one set, X, and the second from the other set, Y, to at least one reducer in common, without exceeding q at any reducer.

Example. We are given two sets X of 12 inputs and Y of 4 inputs, see Figure 4, and reducers of capacity q. We show that we can assign each input of the set X with each input of the set Y in two ways. In order to minimize the communication cost, the best way is to use 12 reducers. Note that we cannot obtain a solution for the given inputs using less than 12 reducers. However, the use of 12 reducers results in less parallelism at the reduce phase as compared to when we use 16 reducers.

Explanation of tradeoffs. Skew join of two relations $X(A, B)$ and $Y(B, C)$ for a heavy hitter is an example of the *X2Y mapping schema problem*. We also explain the tradeoffs using the example of skew join. We assume that both the relations $X(A, B)$ and $Y(B, C)$ have only a single heavy hitter, say b_1. Note that we do not consider tuples with no heavy hitter.

A reducer of capacity q that is sufficient to hold all the tuples of $X(A, B)$ and $Y(B, C)$ with the heavy hitter results in the minimum communication cost. However, due to a single reducer, there is no parallelism at the reduce phase. In addition, a single reducer takes a long time to produce all the desired output tuples of the heavy hitter.

In order to decrease the time (or when q is small but still enough to hold only two tuples, the first from $X(A, B)$ and the second from $Y(B, C)$, which have the common B-value), we can restrict reducers in a way that they can hold many tuples, but not all the tuples with the heavy-hitter-value. In this case, we use more reducers, which result in a higher level of parallelism at the reduce phase. But, there is a higher communication cost, since each tuple with the heavy hitter must be sent to more than one reducer.

3. INTRACTABILITY OF FINDING A MAPPING SCHEMA

In this section, we will show that the *A2A* and the *X2Y* mapping schema problems do not possess a polynomial solution. In other words, we will show that the assignment of *two required inputs* to the minimum number of reducers to find solutions to the *A2A* and the *X2Y* mapping schema problems cannot be achieved in polynomial time.

NP-hardness of the A2A Mapping Schema Problem. A set of inputs $I = \{i_1, i_2, \ldots, i_m\}$ whose input size set is $W = \{w_1, w_2, \ldots, w_m\}$ and a set of reducers $R = \{r_1, r_2, \ldots, r_z\}$, are

an input instance to the *A2A mapping schema problem*. The *A2A mapping schema problem* is a decision problem that asks whether or not there exists a mapping schema for the given input instance such that every input, i_x, is assigned with every other input, i_y, to at least one reducer in common. An answer to the *A2A mapping schema problem* will be "yes," if for each pair of inputs ($\langle i_x, i_y \rangle$), there is at least one reducer that holds them.

Now we prove that the *A2A mapping schema problem* has a polynomial solution to one and two reducers. If there is only one reducer, then the answer is "yes" if and only if the sum of the input sizes $\sum_{i=1}^{m} w_i$ is at most q. On the other hand, if $q < \sum_{i=1}^{m} w_i$, then the answer is "no." In case of two reducers, if a single reducer is not able to accommodate all the given inputs, then there must be at least one input that is assigned to only one of the reducers, and hence, this input is not paired with all the other inputs. In that case, the answer is "no." Therefore, we achieve a polynomial solution to the *A2A mapping schema problem* for one and two reducers. Next, we will prove that the *A2A mapping schema problem* is an NP-hard problem for $z > 2$ reducers.

Theorem 1 *The problem of finding whether a mapping schema of m inputs of different input sizes exists, where every two inputs are assigned to at least one of $z \geq 3$ identical-capacity reducers, is NP-hard.*

Proof is omitted from here and given in [1].

NP-hardness of the X2Y Mapping Schema Problem. Two sets of inputs, $X = \{i_1, i_2, \ldots, i_m\}$ whose input size set is $W_x = \{w_1, w_2, \ldots, w_m\}$ and $Y = \{i'_1, i'_2, \ldots, i'_n\}$ whose input size set is $W_y = \{w'_1, w'_2, \ldots, w'_n\}$, and a set of reducers $R = \{r_1, r_2, \ldots, r_z\}$ are an input instance to the *X2Y mapping schema problem*. The *X2Y mapping schema problem* is a decision problem that asks whether or not there exists a mapping schema for the given input instance such that each input of the set X is assigned with each input of the set Y to at least one reducer in common. An answer to the *X2Y mapping schema problem* will be "yes," if for each pair of inputs, the first from X and the second from Y, there is at least one reducer that has both those inputs.

The *X2Y mapping schema problem* has a polynomial solution for the case of a single reducer. If there is only one reducer, then the answer is "yes" if and only if the sum of the input sizes $\sum_{i=1}^{m} w_i + \sum_{i=1}^{n} w'_i$ is at most q. On the other hand, if $q < \sum_{i=1}^{m} w_i + \sum_{i=1}^{n} w'_i$, then the answer is "no." Next, we will prove that the *X2Y mapping schema problem* is an NP-hard problem for $z > 1$ reducers.

Theorem 2 *The problem of finding whether a mapping schema of m and n inputs of different input sizes that belongs to set X and set Y, respectively, exists, where every two inputs, the first from X and the second from Y, are assigned to at least one of $z \geq 2$ identical-capacity reducers, is NP-hard.*

Proof is omitted from here and given in [1].

4. HEURISTICS FOR THE A2A MAPPING SCHEMA PROBLEM

Since the *A2A Mapping Schema Problem* is NP-hard, in polynomial time we cannot assign each pair of inputs to the minimum number of reducers, which results in the optimum communication between the map phase and the reduce phase. In this section, we propose several heuristics for the *A2A mapping schema problem* that are based on bin-packing algorithms, selection of a prime number p, and division of inputs into two sets based on their sizes. In addition, the proposed heuristics assume that a *fixed* reducer capacity q is given. The heuristics have two cases depending on the sizes of the inputs, as follows:

1. Input sizes are upper bounded by $\frac{q}{2}$,

2. One input is of size, say w_i, greater than $\frac{q}{2}$, but less than q, and all the other inputs have size less than or equal to $q - w_i$,

The idea of the heuristics is: if the two largest inputs are greater than the given reducer capacity q, then there is no solution to the *A2A mapping schema problem* because these two inputs cannot be assigned to a single reducer in common.

Parameters for bounds analysis. We analyze our heuristics on three parameters, as follows:

1. Minimum number of reducers, $r(m, q)$. The minimum number of reducers of capacity q that can solve the *A2A* (and *X2Y*) mapping schema problem(s) for the given inputs with certain size restrictions.

2. Replication of individual inputs. Inputs of different sizes need to be replicated at different numbers of reducers. We therefore need to consider the minimum number of reducers to which each individual input is sent.

3. The total communication cost, c. The total communication cost is defined to be the sum of all the bits that are required to transfer from the map phase to the reduce phase.

4.1 All the inputs sizes are upper bounded by $\frac{q}{2}$

We first consider the case where all the input sizes are at most $\frac{q}{2}$ size. We consider the following two cases in this section: (*i*) all the inputs are potentially of different sizes but at most size $\frac{q}{2}$, and (*ii*) all the inputs sizes are almost equal or there are a lot of inputs of very small size. Particularity, all the inputs are of size at most $\frac{q}{k}$, where $k > 1$,

4.1.1 Different-sized inputs but at most size $\frac{q}{2}$

We first provide a heuristic for inputs of potentially different sizes, where the largest input can have at most size $\frac{q}{2}$. The heuristic uses a bin-packing algorithm to place the given m inputs into bins of size $\frac{q}{2}$. Before going into details of the heuristic, we look at the lower bound on the replication of an input i (of size w_i), the total number of reducers, and the total communication cost between the map phase and the reduce phase. The bounds are given in Table 1.

Theorem 3 (Replication of individual inputs) *For a set of m inputs and a given reducer capacity q, an input i of size $w_i < q$ is required to be sent to at least $\lceil \frac{s-w_i}{q-w_i} \rceil$ reducers for a solution to the A2A mapping schema problem, where s is the sum of all the input sizes.*

PROOF. Consider an input i of size w_i. The input i can share a reducer with inputs whose sum of the sizes is at most $q - w_i$. In order to assign the input i with all the remaining $m - 1$ inputs, it is required to assign subsets of the $m - 1$ inputs, each subset with sum of sizes at most size $q - w_i$. Such an assignment results in at least $\lceil \frac{s-w_i}{q-w_i} \rceil$ subsets of the $m - 1$ inputs. Thus, the input i is required to be sent to at least $\lceil \frac{s-w_i}{q-w_i} \rceil$ reducers to be paired with all the remaining $m - 1$ inputs. □

Theorem 4 (The total communication cost and number of reducers) *For a set of m inputs and a given reducer capacity q, the total communication cost and the total number of reducers, for the A2A mapping schema problem, are at least $\frac{s^2}{q}$ and $\frac{s^2}{q^2}$, respectively, where s is the sum of all the input sizes.*

PROOF. Since an input i is replicated to at least $\lceil \frac{s-w_i}{q-w_i} \rceil$ reducers, the communication cost for the input i is $w_i \times \lceil \frac{s-w_i}{q-w_i} \rceil$. Hence, the total communication cost for all the inputs will be at least $\sum_{i=1}^{m} w_i \frac{s-w_i}{q-w_i}$. Since $s \geq q$, we can conclude $\frac{s-w_i}{q-w_i} \geq \frac{s}{q}$. Thus, the total communication cost is at least $\sum_{i=1}^{m} w_i \frac{s}{q} = \frac{s^2}{q}$.

Since the total communication cost, the total number of bits to be assigned to reducers, is at least $\frac{s^2}{q}$, and a reducer can hold inputs whose sum of the sizes is at most q, the total number of reducers must be at least $\frac{s^2}{q^2}$. □

Bin-packing-based Heuristic. First-Fit Decreasing (FFD) and Best-Fit Decreasing (BFD) [5] are most notable bin-packing algorithms. We use the FFD or BFD bin-packing algorithm to place the given m inputs to bins of size $\frac{q}{2}$. Assume that FFD or BFD bin-packing algorithm needs x bins to place m inputs, and now, each of such bins is considered as a single input of size $\frac{q}{2}$. Since the reducer capacity is q, any two bins can be assigned to a single reducer. Hence, the heuristic uses at most $r(m, q) = \frac{x(x-1)}{2}$ reducers; see Figure 5. There also exists a pseudo polynomial bin-packing algorithm, suggested by Karger and Scott [9], that can place the m inputs in as few bins as possible of size $\frac{q}{2}$.

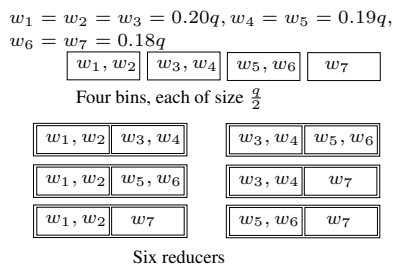


Figure 5: Bin-packing-based heuristic.

Total required reducers. FFD and BFD bin-packing algorithms provide an $\frac{11}{9} \cdot$ OPT approximation ratio [8], *i.e.*, if any optimum bin-packing algorithm needs OPT bins to place (m) inputs in the bins of a given size (say, $\frac{q}{2}$), then FFD and BFD bin-packing algorithms always use at most $\frac{11}{9} \cdot$ OPT bins of the same size (to place the given m inputs). Since we require at most $\frac{x(x-1)}{2}$ reducers for a solution to the *A2A mapping schema problem*, the heuristic requires at most $r(m, q) = \frac{(\frac{11}{9} \cdot \text{OPT})^2}{2}$ reducers.

Note that, here in this case, OPT does not indicate the optimum number of reducers to assign m inputs that satisfy the *A2A mapping schema problem*; OPT indicates the optimum number of bins of size $\frac{q}{2}$ that are required to place m inputs.

The following theorem gives the upper bounds that this heuristic achieves on the replication of an inputs, the total communication cost and the number of reducers.

Theorem 5 (Upper bounds from the heuristic) *The above heuristic using a bin size $b = \frac{q}{2}$ where q is the reducer capacity achieves the following three upper bounds: The total number of reducers, the replication of individual inputs, and the total communication cost, for the A2A mapping schema problem, are at most $\frac{8s^2}{q^2}$, at most $4\frac{s}{q}$, and at most $4\frac{s^2}{q}$, respectively, where s is the sum of all the input sizes.*

PROOF. A bin i can hold inputs whose sum of the sizes is at most b. Since the total sum of the sizes is s, it is required to divide the inputs into at least $\frac{s}{b}$ bins. Since the FFD or BFD bin-packing algorithm ensures that all the bins (except only one bin) are at least half-full, each bin of size $\frac{q}{2}$ has at least inputs whose sum of the sizes is at least $\frac{q}{4}$. Thus, all the inputs can be placed in at most

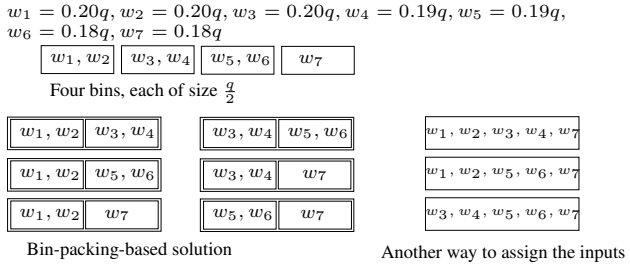


Figure 7: Comparison between the bin-packing-based heuristic and the proposed algorithms for almost equal-sized inputs.

$\frac{s}{q/4}$ bins of size $\frac{q}{2}$. Since each bin is considered as a single input, we can assign every two bins at a reducer, and hence, we require at most $\frac{8s^2}{q^2}$ reducers. Since each bin is replicated to at most $4\frac{s}{q}$ reducers, the replication of individual inputs is at most $4\frac{s}{q}$ and the total communication cost is at most $\sum_{1 \leq i \leq m} w_i \times 4\frac{s}{q} = 4\frac{s^2}{q}$. □

4.1.2 Almost equal-sized inputs

In this section, we provide two algorithms for m almost equal-sized ($\frac{q}{k}$, where $k > 1$) inputs to assign each pair of inputs to reducers of capacity q. In other word, we are given a lot of inputs of very small sizes as compared to q. We pack all these inputs to bins of unit size, and then consider each bin as a single input of unit-size. Equivalently, we can take the reducer capacity to be q and the inputs to be of unit size. In what follows, we will continue to use q as the reducer capacity and assume all inputs are of unit size.

The two algorithms can be summarized as follows: *2-step algorithms* (Algorithm 1 and Algorithm 2) handle the case of m unit-sized inputs and odd-even values of the reducer capacity q. Algorithms 1 and 2 assume that q is an odd or an even number, respectively.

Aside. Algorithms 1 and 2 have an advantage over the bin-packing-based heuristic (Section 4.1.1). When inputs of almost identical sizes are given, the bin-packing-based heuristic uses more reducers as compared to Algorithms 1 and 2. For example, we are given a set of seven inputs $I = \{i_1, i_2, \ldots, i_7\}$ whose size set is $W = \{0.20q, 0.20q, 0.20q, 0.19q, 0.19q, 0.18q, 0.18q\}$. In this case, the bin-packing-based heuristic uses at least six reducers while we can assign them to three reducers, see Figure 7.

Our goal to use Algorithms 1, 2, 3, and 4 is to minimize the communication cost between the map and reduce phases for a given number of unit-sized inputs and the reducer capacity q. Before going into details of algorithms for $q > 2$, we look at the lower bound on the total communication cost between the map and reduce phases. The case of m inputs of size one and reducers of capacity two is trivial. In this case, we can assign every pair of inputs to a single reducer, which results in $r(m, q) = \frac{m(m-1)}{2}$ reducers, and moreover, it is clearly impossible to use fewer reducers.

Theorem 6 (Replication of individual inputs) *For a given reducer capacity $q > 1$ and a set of m inputs of size one, an input i required to be sent to at least $\lceil \frac{m-1}{q-1} \rceil$ reducers for a solution to the A2A mapping schema problem.*

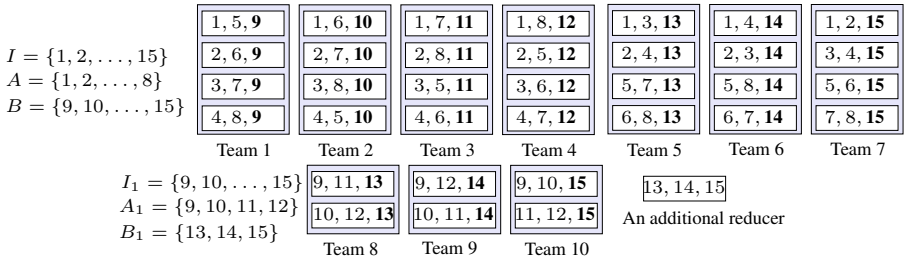
PROOF. Consider an input i. The input i can share a reducer with only $q - 1$ inputs. In order to assign the input i with all the remaining $m - 1$ inputs, it is required to create disjoint subsets of the remaining $m - 1$ inputs such that each subset can hold at most $q - 1$ inputs. In this manner, there are at least $\lceil \frac{m-1}{q-1} \rceil$ subsets of

Cases	Solutions	Sections	Theorems	Replication of individual inputs	Reducers	Communication cost
The **lower bounds** for the *A2A mapping schema problem*						
Different-sized inputs		4.1.1	3 and 4	$\frac{s}{q}$	$\frac{s^2}{q^2}$	$\frac{s^2}{q}$
Almost equal-sized inputs		4.1.2	6 and 7	$\lceil \frac{m-1}{q-1} \rceil$	$\lfloor \frac{m}{q} \rfloor \lceil \frac{m-1}{q-1} \rceil$	$m\lceil \frac{m-1}{q-1} \rceil$
The **lower bounds** for the *X2Y mapping schema problem*						
Different-sized inputs		5	13 and 14	$\frac{sum_x}{q}$, $\frac{sum_y}{q}$	$\frac{2 \cdot sum_x \cdot sum_y}{q^2}$	$\frac{2 \cdot sum_x \cdot sum_y}{q}$
The **upper bounds** for the *A2A mapping schema problem*						
Different-sized inputs	Bin-packing-based heuristic	4.1.1	5	$\frac{4s}{q}$	$\frac{8s^2}{q^2}$	$\frac{4s^2}{q}$
Almost equal-sized inputs	Algorithm 1	4.1.2	9	$\lceil \frac{2m}{q-1} \rceil - 1$	$(\lceil \frac{2m}{(q-1)} \rceil)^2/2$	$m\big(\lceil \frac{2m}{(q-1)} \rceil - 1\big)$
	Algorithm 2	4.1.2	11	$\lceil \frac{2m}{q} \rceil - 1$	$(\lceil \frac{2m}{q} \rceil)^2/2$	$m\big(\lceil \frac{2m}{q} \rceil - 1\big)$
An input of size $> \frac{q}{2}$	Bin-packing-based heuristic	4.2	12	$m-1$	$m-1+\frac{8s^2}{q^2}$	$(m-1)\cdot q + \frac{4s^2}{q}$
The **upper bounds** for the *X2Y mapping schema problem*						
Different-sized inputs, $q = 2b$	Bin-packing-based heuristic	5	15	$\frac{2 \cdot sum_x}{b}$, $\frac{2 \cdot sum_y}{b}$	$\frac{4 \cdot sum_x \cdot sum_y}{b^2}$	$\frac{4 \cdot sum_x \cdot sum_y}{b}$

Table 1: The bounds for heuristics for the *A2A* and the *X2Y* mapping schema problems.

Algorithm 1, with $q = 3$, divides m unit-sized inputs into two disjoint sets A and B of y and $x \leq y - 1$ inputs respectively. (The selection of the value of y will be described later. But, note that we prefer y to be a power of 2, and if $y \neq 2^i$ and $y > 4$, $i > 2$, then we add unit-sized dummy inputs so that y is a power of 2.) When $q = 3$, we organize $(y-1) \times \lceil \frac{y}{2} \rceil$ reducers (each of capacity three) in the form of $y - 1$ *teams* of $\lceil \frac{y}{2} \rceil$ *players* (or reducers) in each team, and these reducers are used to assign each input of the set A with all the remaining inputs of the sets A and B. Note that a team must have each of the inputs of the set A occurring exactly once among the reducers of that team, and this fact will be clear soon.

There are $(y-1) \times \lceil \frac{y}{2} \rceil$ pairs of inputs of the set A (each of size two) and there are the same number of reducers (each of capacity three); hence, it is possible to assign one pair to each reducer, and these two inputs become two of the three inputs allowed to each reducer. Once, we assign every pair of inputs of the set A to $(y-1) \times \lceil \frac{y}{2} \rceil$ reducers, then we assign i^{th} input of the set B to all the $\lceil \frac{y}{2} \rceil$ reducers of i^{th} team. Further, we follow a similar procedure on inputs of the set B to assign each pair of the remaining x inputs.



Example. We are given 15 inputs ($I = \{1, 2, \ldots, 15\}$) of size one and $q = 3$. We create two sets, namely A of $y = 8$ inputs and B of $x = 7$ inputs, and arrange $(y-1) \times \lceil \frac{y}{2} \rceil = 28$ reducers in the form of 7 *teams* of 4 *players* (or reducers) in each team. These 7 teams assign each input of the set A with all the remaining inputs of the set A and the set B. We pair every two inputs of the set A and assign them to exactly one of 28 reducers. (All these pairs of the inputs of the set A are created and assigned using lines 10, 12, and 13 of Algorithm 1.) Once every pair of $y = 8$ inputs of the set A is assigned to exactly one of 28 reducers, then we assign i^{th} input of the set B to all the four reducers of i^{th} team, see Team 1 to Team 7. Of course, the third input in i^{th} team is i^{th} input of the set B.

Now note that the first four teams pair inputs 1-4 with inputs 5-8. The first team (Team 1) has pairs $\{1, 5\}$, $\{2, 6\}$, $\{3, 7\}$, and $\{4, 8\}$. Team 2-4 has pairs by rotation of the 5-8 inputs. Teams 5 and 6 handle pairs of 1-2 with 3-4 and 5-6 with 7-8, respectively, in the same way, and the last team has pairs $\{1, 2\}, \{3, 4\}, \ldots, \{7, 8\}$.

Next, we implement the same procedure on 7 inputs of the set B. We create two sets, say $A_1 = \{9, 10, 11, 12\}$ of $y_1 = 4$ inputs and $B_1 = \{13, 14, 15\}$ of $x_1 = 3$. Then, we arrange $(y_1 - 1) \times \lceil \frac{y_1}{2} \rceil = 6$ reducers in the form of 3 *teams* of 2 reducers in each team. We assign each pair of inputs of the set A_1 to these 6 reducers, and then i^{th} input of the set B_1 to all the two reducers of a team, see Team 8 to Team 10. Further, we assign the remaining inputs of the set B_1 to a single reducer. The assignment of inputs to Teams 8-10 follows the same procedure as we did for Teams 1-7.

We have three claims, as follows: (*i*) each input of the set A appears exactly once in each team, (*ii*) the set B holds $x \leq y - 1$ inputs when $q = 3$, and (*iii*) the given algorithm assigns each pair of inputs to at least one reducer. We will prove these claims in algorithm correctness.

Figure 6: 2-step algorithm (Algorithm 1) for the reducer capacity $q = 3$ and $m = 15$.

$m-1$ inputs. Hence, the input i is required to be sent to at least $\lceil \frac{m-1}{q-1} \rceil$ reducers. □

Theorem 7 (The total communication cost and number of reducers) *For a given reducer capacity $q > 1$ and a set of m inputs of size one, the total communication cost and the total number of reducers, for the A2A mapping schema problem, are at least $m\lceil \frac{m-1}{q-1} \rceil$ and at least $\lfloor \frac{m}{q} \rfloor \lceil \frac{m-1}{q-1} \rceil$, respectively.*

PROOF. Since an input i is required to be sent to at least $\lceil \frac{m-1}{q-1} \rceil$ reducers, the sum of the number of copies of (m) inputs sent to reducers is at least $m\lceil \frac{m-1}{q-1} \rceil$, which result in at least $m\lceil \frac{m-1}{q-1} \rceil$ communication cost.

There are at least $m\lceil \frac{m-1}{q-1} \rceil$ total number of copies of (m) inputs to be sent to reducers and a reducer can hold at most q inputs; hence, at least $\lfloor \frac{m}{q} \rfloor \lceil \frac{m-1}{q-1} \rceil$ reducers are required. □

Algorithm 1: 2-step algorithm when the reducer capacity q is an odd number. For the sake of understanding and presentation, we first present two examples, where $q = 3$, *i.e*, a reducer can hold at most three unit-sized inputs; see Figure 6 (and $q = 5$, *i.e*., a reducer can hold at most five unit-sized inputs; see Figure 11 in [1]).

Following the example given for $q = 3$ (in Figure 6), we present our algorithm (see Algorithm 1) that handles any odd value of q. The algorithm consists of five steps as follows:
1. Divide m inputs into two sets A and B of size $y = \lfloor \frac{q}{2} \rfloor (\lfloor \frac{2m}{q+1} \rfloor + 1)$ and $x = m - y$, respectively.
2. Group the y inputs into $u = \lceil \frac{y}{q-\lceil q/2 \rceil} \rceil$ disjoint groups, where each group holds $\lceil \frac{q-1}{2} \rceil$ inputs. (We consider each of the u $(= \lceil \frac{y}{q-\lceil q/2 \rceil} \rceil)$ disjoint groups as a single input that we call the *derived input*. By making u disjoint groups[1] (or derived inputs) of y inputs of the set A, we turn the case of any odd value of q to a case where a reducer can hold only three inputs, the first two inputs are pairs of the derived inputs and the third input is from the set B.)
3. Organize $(u-1) \times \lceil \frac{u}{2} \rceil$ reducers, each of capacity q, in the form of $u-1$ teams of $\lceil \frac{u}{2} \rceil$ reducers in each team. Assign every two groups to one of $(u-1) \times \lceil \frac{u}{2} \rceil$ reducers. To do so, we will prove the following Lemma 1, and its proof is omitted from here due to space and given in [1].

Lemma 1 *Each pair of $u = 2^i$, $i > 0$, inputs can be assigned to $2^i - 1$ teams of 2^{i-1} reducers in each team, where the reducer capacity is q and the size of each input is $\lceil \frac{q-1}{2} \rceil$.*

4. Once every pair of the derived inputs are assigned, then assign i^{th} input of the set B to all the reducers of i^{th} team.
5. Apply (the above mentioned) steps 1-4 on the set B until there is a solution to the *A2A mapping schema problem* for the x inputs.

Algorithm description. Algorithm 1 provides a solution to the *A2A mapping schema problem* for unit-sized inputs when q is an odd number. First, we divide m inputs into two sets A and B. Then, we make $u = \lceil \frac{y}{q-\lceil q/2 \rceil} \rceil$ disjoint groups of y inputs of the set A such that each group holds $\frac{q-1}{2}$ inputs, lines 1, 2. (Now, each of the groups is considered as a single input that we call the *derived input*.) We do not show the addition of dummy inputs and assume that u is a power of 2. Function $2\_step\_odd\_q(lower, upper)$ recursively divides the derived inputs into two halves, line 4. Function $Assignment(lower, mid, upper)$ (line 8) pairs every two derived inputs and assigns them to the respective reducers (line 10). Each reducer of the last team is assigned using function $Last\_Team(groupA[])$, lines 14, 15.

Note that functions $2\_step\_odd\_q(lower, upper)$, $Assignment(lower, mid, upper)$, and $value\_b(lower, t, mid, upper)$ take two common parameters, namely $lower$ and $upper$ where $lower$ is the first derived input and $upper$ is the last derived input (*i.e*., u^{th} group) at the time of the first call to functions, line 3. Once all-pairs of the derived inputs are assigned to reducers, line 10, function $Assign\_input\_from\_B(Team[])$ assigns i^{th} input of the set B to all the $\lceil \frac{u}{2} \rceil$ reducers of i^{th} team, lines 16, 17. After that, algorithm is invoked over inputs of the set B to assign each pair of the remaining inputs of the set B to reducers until every pair to the remaining inputs is assigned to reducers.

Algorithm correctness. The algorithm correctness proves that every pair of inputs is assigned to reducers. Specifically, we prove that all those pairs of inputs, $\langle i, j \rangle$ and $\langle i', j' \rangle$, of the set A are assigned to a team whose $i \neq i'$ and $j \neq j'$ (Claim 1). Then that all the inputs of the set A appear exactly once in each team (Claim 2). We then prove that the set B holds $x \leq y-1$ inputs, when $q = 3$ (Claim 3). At last we conclude in Theorem 8 that Algorithm 1 assigns each pair of inputs to reducers.

Note that we are proving all the above mentioned claims for $q = 3$; the cases for $q > 3$ can be generalized trivially where we make $u = \lceil \frac{y}{q-\lceil q/2 \rceil} \rceil$ derived inputs from y inputs of the set A (and assign in a manner that all the inputs of the A are paired with all the remaining $m-1$ inputs).

Claim 1 *Pairs of inputs $\langle i, j \rangle$ and $\langle i', j' \rangle$, where $i \neq i'$ or $j \neq j'$, of the set A are assigned to a team.*

PROOF. First, consider $i = i'$ and $j \neq j'$, where $\langle i, j \rangle$ and $\langle i', j' \rangle$ must be assigned to two different teams. If $j \neq j'$, then both the j values may have an identical value of $lower$ and mid but they must have two different values of t (see lines 12, 13 of Algorithm 1), where $j = lower + t + mid$ or $j = lower + t$. Thus, for two different values of j , we use two different values of t, say t_1 and t_2, that results in an assignment of $\langle i, j \rangle$ and $\langle i', j' \rangle$ to two different teams t_1 and t_2, (note that teams are also selected based on the value of t, $(y - 2 \cdot mid + 1) + t$, see line 10 of Algorithm 1, where for $q = 3$, we have $u = y$). Suppose now that $i \neq i'$ and $j = j'$, where $\langle i, j \rangle$ and $\langle i', j' \rangle$ must be assigned to two different teams. In this case, we also have two different values of t, and hence, two different t values assign $\langle i, j \rangle$ and $\langle i', j' \rangle$ to two different teams ($(y - 2 \cdot mid + 1) + t$, line 10 of Algorithm 1).

Hence, it is clear that pairs $\langle i, j \rangle$ and $\langle i', j' \rangle$, where $i \neq i'$ and $j \neq j'$, are assigned to a team. □

Claim 2 *All the inputs of the set A appear exactly once in each team.*

PROOF. There are the same number of pairs of inputs of the set A and the total number of reducers ($(y-1)\lceil \frac{y}{2} \rceil$) that can provide a solution to the *A2A mapping schema problem* for the y inputs of the set A. Note that if there is a input pair $\langle i, j \rangle$ in team t, then the team t cannot hold any pair that has either i or j in the remaining $\lceil \frac{y}{2} \rceil - 1$ reducers. For the given y inputs of the set A, there are at most $\lceil \frac{y}{2} \rceil$ disjoint pairs $\langle i_1, j_1 \rangle$, $\langle i_2, j_2 \rangle$, ..., $\langle i_{\lceil y/2 \rceil}, j_{\lceil y/2 \rceil} \rangle$ such that $i_1 \neq i_2 \neq \ldots \neq i_{\lceil y/2 \rceil} \neq j_1 \neq j_2 \neq \ldots \neq j_{\lceil y/2 \rceil}$. Hence,

[1]We suppose that u is a power of 2. In case u is not a power of 2 and $u > q$, we add dummy inputs each of size $\lceil \frac{q-1}{2} \rceil$ so that u becomes a power of 2. Consider that we require d dummy inputs. If groups of inputs of the set B each of size $\lceil \frac{q-1}{2} \rceil$ are less than equal to d dummy inputs, then we use inputs of the set B in place of dummy inputs, and the set B will be empty.

Algorithm 1: 2-step algorithm for an odd value of q.

Inputs:
m: total number of unit-sized inputs
q: the reducer capacity.
Variables:
A: A set A, where the total inputs in the set A is $y = \lfloor \frac{q}{2} \rfloor (\lfloor \frac{2m}{q+1} \rfloor + 1)$.
B: A set B, where the total inputs in the B is $x = m - (\lceil \frac{y}{q - \lceil q/2 \rceil} \rceil - 1)$.
$Team[i, j]$: represents teams of reducers, where index i indicates i^{th} team and index j indicates j^{th} reducer in i^{th} team. Consider $u = \lceil \frac{y}{q - \lceil q/2 \rceil} \rceil$. There are $u - 1$ teams of $v = \lceil \frac{u}{2} \rceil$ reducers in each team.
$groupA[]$: represents disjoint groups of inputs of the set A, where $groupA[i]$ indicates i^{th} group of $\lceil \frac{q-1}{2} \rceil$ inputs of the A.

Function $create\_group(y)$ **begin**
  for $i \leftarrow 1$ **to** u **do** $groupA[i] \leftarrow \langle i, i+1 \ldots, i + \frac{q-1}{2} - 1 \rangle, i \leftarrow i + \frac{q-1}{2}$;
  $2\_step\_odd\_q(1, u)$, $Last\_Team(groupA[])$, $Assign\_input\_from\_B(Team[])$
Function $2\_step\_odd\_q(lower, upper)$ **begin**
  if $\lfloor \frac{upper - lower}{2} \rfloor < 1$ **then return**;
  else
    $mid \leftarrow \lceil \frac{upper - lower}{2} \rceil$, $Assignment(lower, mid, upper)$, $2\_step\_odd\_q(lower, mid)$, $2\_step\_odd\_q(mid + 1, upper)$
Function $Assignment(lower, mid, upper)$ **begin**
  while $mid > 1$ **do**
    foreach $(a, t) \in [lower, lower + mid - 1] \times [0, mid - 1]$ **do**
      $Team\big[(u - 2 \cdot mid + 1) + t, a - \lfloor \frac{a-1}{mid} \rfloor \cdot \frac{mid}{2} \big] \leftarrow \langle groupA[a], groupA[value\_b(a, t, mid, upper)] \rangle$;
Function $value\_b(a, t, mid, upper)$ **begin**
  if $a + t + mid < upper + 1$ **then return** $(a + t + mid)$;
  else if $a + t + mid > upper$ **then return** $(a + t)$;
Function $Last\_Team(lower, mid, upper)$ **begin**
  foreach $i \in [1, v]$ **do** $Team[u - 1, i] \leftarrow groupA[2 \times i - 1], groupA[2 \times i]$;
Function $Assign\_input\_from\_B(Team[])$ **begin**
  foreach $(i, j) \in [1, u - 1] \times [1, v]$ **do** $Team[i, j] \leftarrow B[i]$;

all y inputs of the set A are assigned to a team, where no input is assigned twice in a team. □

Claim 3 *When the reducer capacity $q = 3$, the set B holds at most $x \leq y - 1$ inputs.*

PROOF. Since a pair of inputs of the set A requires at most $q-1$ capacity of a reducer and each team holds all the inputs of the set A, an input from the set B can be assigned to all the reducers of the team. In this manner, all the inputs of the set A are also paired with an input of the set B. Since there are $y - 1$ teams and each team is assigned an input of the set B, the set B can hold at most $x \leq y - 1$ inputs. □

Theorem 8 *Algorithm 1 assigns each pair of the given m inputs to reducers.*

PROOF. We have $(y - 1)\lceil \frac{y}{2} \rceil$ pairs of inputs of the set A of size $q - 1$, and there are the same number of reducers; hence, each reducer can hold one input pair. Further, the remaining capacity of all the reducers of each team can be used to assign an input of B. Hence, all the inputs of A are paired with every other input and every input of B (as we proved in Claims 2 and 3). Following the fact that the inputs of the set A are paired with all the m inputs, the inputs of the set B is also paired by following a similar procedure on them. Thus, Algorithm 1 assigns each pair of the given m inputs to reducers. □

Theorem 9 *Algorithm 1 requires at most $(\lceil \frac{2m}{(q-1)} \rceil)^2/2$ reducers and results in at most $m\big(\lceil \frac{2m}{(q-1)} \rceil - 1\big)$ communication cost.*

PROOF. There are at most $x = \lceil \frac{2m}{q-1} \rceil$ groups (derived inputs) of the given m inputs. In order to assign each pair of the derived inputs, each derived input is required to assign to at most $x - 1$ reducers. This fact results in at most $m\big(\lceil \frac{2m}{(q-1)} \rceil - 1\big)$ communication cost, and there are at most $(\lceil \frac{2m}{(q-1)} \rceil)^2/2$ pairs of the derived inputs that require at most $(\lceil \frac{2m}{(q-1)} \rceil)^2/2$ reducers. □

Algorithm 2: 2-step algorithm when the reducer capacity q is an even number. We present our algorithm (see Algorithm 2) that handles any even value of q. For the sake of understanding and presentation, we first present an example where $q = 4$, namely the case in which a reducer can hold at most four unit-sized inputs, as demonstrated in Figure 8 (Figure 8 is self-explainable; however, interested readers may refer to Figure 12 in [1] for details). Note that unlike the algorithm for odd values of q (Algorithm 1) the algorithm for even values of q (Algorithm 2) does not divide the m inputs into two sets. The algorithm consists of two steps, as follows:
1. Group the given m inputs into $u = \lceil \frac{2m}{q} \rceil$ disjoint groups,
2. Organize $(u-1) \times \frac{u}{2}$ reducers, each of capacity q, in the form of $u - 1$ teams of $\frac{u}{2}$ reducers in each team. Assign every two groups to one of $(u-1) \times \frac{u}{2}$ reducers. We use Lemma 1 for the assignment of every two groups.

Note that we consider each of the u $(= \lceil \frac{2m}{q} \rceil)$ groups as a single input that we call the *derived input*. By making u disjoint groups of the m inputs, we turn the case of any even value of q to a case when $q = 2$ (*i.e.*, a reducer can hold only two unit-sized inputs) and assign every two derived inputs to reducers. In this

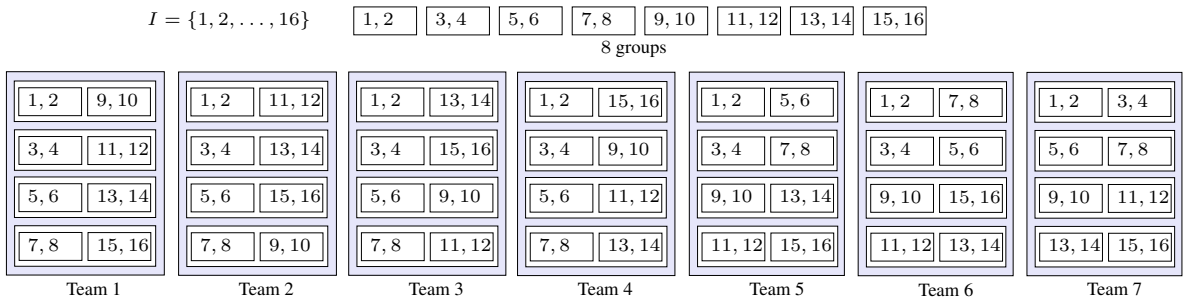


Figure 8: 2-step algorithm (Algorithm 2) when the reducer capacity $q = 4$.

Algorithm 2: 2-step algorithm for an even value of q.

Inputs: m: total number of unit-sized inputs.
q: the reducer capacity.
Variables: $Team[i, j]$: represents teams of reducers, where index i indicates i^{th} team and index j indicates j^{th} reducer in i^{th} team. Consider $u = \lceil \frac{2m}{q} \rceil$. There are $u - 1$ teams of $\lceil \frac{u}{2} \rceil$ reducers in each team.
$groupA[]$: represents disjoint groups of inputs of the set A, where $groupA[i]$ indicates i^{th} group of $\lceil \frac{q}{2} \rceil$ inputs of the set A.

1 **Function** $create\_group(m)$ **begin**
2 **for** $i \leftarrow 1$ **to** u **do**
   $groupA[i] \leftarrow \langle i, i+1 \ldots, i + \frac{q}{2} - 1 \rangle, i \leftarrow i + \frac{q}{2}$;
3 $2\_step\_even\_q(1, u)$, $Last\_Team(1, \lceil \frac{u-1}{2} \rceil, u)$
4 **Function** $2\_step\_even\_q(lower, upper)$ **begin**
5 **if** $\lfloor \frac{upper-lower}{2} \rfloor < 1$ **then return**;
6 **else**
7 $mid \leftarrow \lceil \frac{upper-lower}{2} \rceil$,
   $Assignment(lower, mid, upper)$,
   $2\_step\_even\_q(lower, mid)$,
   $2\_step\_even\_q(mid + 1, upper)$,

manner, each input of the set A is assigned with all the remaining $m - 1$ inputs.

Algorithm description. Algorithm 2 provides a solution to the *A2A mapping schema problem* for unit-sized inputs when q is an even number. Recall that Algorithm 1 and Algorithm 2 are almost similar except Algorithm 2 does not create two sets A and B. We first make $u = \lceil \frac{2m}{q} \rceil$ disjoint groups of the given m inputs such that each group holds $\frac{q}{2}$ inputs (lines 2), (and consider each of the groups as a single input, the *derived input*). Function $2\_step\_even\_q(lower, upper)$ recursively divides the derived inputs into two halves, lines 4 and 7. Function $Assignment(lower, mid, upper)$ (line 7) is a similar function as given for Algorithm 1 (see line 8 of Algorithm 1) and makes pairs of every two derived inputs. Function $Last\_Team(group[])$ (lines 3) assigns inputs to the last team, *i.e.*, team $u - 1$. Note that function $Last\_Team(group[])$ is same as given for Algorithm 1 (see line 14 of Algorithm 1).

Algorithm correctness. We show that every pair of inputs is assigned to reducers. Specifically, Algorithm 2 satisfies two claims, as follows:

Claim 4 *Pairs of derived inputs $\langle i, j \rangle$ and $\langle i', j' \rangle$, where $i \neq i'$ or $j \neq j'$, are assigned to a team.*

Claim 5 *All the given m inputs appear exactly once in each team.*

Theorem 10 *Algorithm 2 assigns each pair of the given m inputs to reducers.*

Theorem 11 *Algorithm 2 requires at most $(\lceil \frac{2m}{q} \rceil)^2/2$ reducers and results in at most $m(\lceil \frac{2m}{q} \rceil - 1)$ communication cost.*

Claim 4, Claim 5, Theorems 10, and 11 can be proved in a similar manner as Claim 1, Claim 2, Theorems 8, and 9, respectively. Detailed proofs are given in [1].

4.2 A big input of size greater than $\frac{q}{2}$

We now consider the case of an input of size w_i, $\frac{q}{2} < w_i < q$; we call such an input as a *big input*. Note that if there are two big inputs, then they cannot be assigned to a single reducer, and hence, there is no solution to the *A2A mapping schema problem*. We assume m inputs of different sizes are given. There is a big input and all the remaining $m - 1$ inputs, which we call the *small inputs*, have at most size $q - w_i$.

We use FFD or BFD bin-packing algorithm to place the small inputs to bins of size $q - w_i$. Now, we consider each of the bins as a single input of size $q - w_i$. Let x bins are used. We assign each of the x bins to one reducer with a copy of the big input. Further, we assign the small inputs to bins of size $\frac{q}{2}$, and consider each of such bins as a single input of size $\frac{q}{2}$. Now, we can assign each pair of bins (each of size $\frac{q}{2}$) to reducers. In this manner, each pair of inputs is assigned to reducers.

Theorem 12 (Upper bounds from the heuristic) *For a set of m inputs where a big input, i, of size $\frac{q}{2} < w_i < q$ and for a given reducer capacity q, $q < s' < s$, an input is replicated to at most $m-1$ reducers for the A2A mapping schema problem, and the total number of reducers and the total communication cost are at most $m - 1 + \frac{8s^2}{q^2}$ and $(m-1)q + \frac{4s^2}{q}$, respectively, where s' is the sum of all the input sizes except the size of the big input and s is the sum of all the input sizes.*

PROOF. The big input i can share a reducer with inputs whose sum of the sizes is at most $q - w_i$. In order to assign the input i with all the remaining $m - 1$ small inputs, it is required to assign a subset of $m - 1$ inputs whose sum of the sizes is at most $q - w_i$. If all the small inputs are of size almost $q - w_i$, then a reducer can hold the big input and one of the small inputs. Hence, the big input is required to be sent to at most $m - 1$ reducers that results in at most $(m - 1)q$ communication cost.

Also, each pair of all the small inputs is assigned to reducers (by first placing them to bins of size $\frac{q}{2}$ using FFD or BFD bin-packing algorithm). The assignment of all the small inputs results in at most $\frac{8s'^2}{q^2} < \frac{8s^2}{q^2}$ reducers and at most $\frac{4s'^2}{q} < \frac{4s^2}{q}$ communication cost (Theorem 5). Thus, the total number of

reducers are at most $m - 1 + \frac{8s^2}{q^2}$ and the total communication cost is at most $(m-1)q + \frac{4s^2}{q}$. $\square$

5. A HEURISTIC FOR THE X2Y MAPPING SCHEMA PROBLEM

We propose a heuristic for the *X2Y mapping schema problem* that is based on bin-packing algorithms. The proposed heuristic assumes a fixed reducer capacity q. Two sets, X of m inputs and Y of n inputs, are given. We assume that the sum of input sizes of the sets X, denoted by sum_x, and Y, denoted by sum_y, is greater than q. We analyze the heuristic on criteria given in Section 4. We look at the lower bounds in Theorems 13 and 14, and Theorem 15 gives an upper bound from a heuristic. The bounds are given in Table 1.

Theorem 13 (Replication of individual inputs) *For a set X of m inputs, a set Y of n inputs, and a given reducer capacity q, an input i of the set X is required to be sent to at least $\frac{sum_y}{q}$ reducers and an input j of the set Y is required to be sent to at least $\frac{sum_x}{q}$ reducers for a solution to the X2Y mapping schema problem.*

Theorem 14 (The total communication cost and number of reducers) *For a set X of m inputs, a set Y of n inputs, and a given reducer capacity q, the total communication cost and the total number of reducers, for the X2Y mapping schema problem, are at least $\frac{2 \cdot sum_x \cdot sum_y}{q}$ and $\frac{2 \cdot sum_x \cdot sum_y}{q^2}$, respectively.*

Bin-packing-based heuristic for the *X2Y mapping schema problem.* A solution to the *X2Y mapping schema problem* for different-sized inputs can be achieved using bin-packing algorithms. Let a fixed reducer capacity q, two sets X of m inputs, and Y of n inputs are given. The heuristic will not work when a set holds an input of size w_i and the another set holds an input of size greater than $q - w_i$, because these inputs cannot be assigned to a single reducer in common. Let the size of the largest input, i, of the set X is w_i; hence, all the inputs of the set Y have at most size $q - w_i$. We place inputs of the set X to bins of size w_i, and let x bins are used to place m inputs. Also, we place inputs of the set Y to bins of size $q - w_i$, and let y bins are used to place n inputs. Now, we consider each of the bins as a single input, and a solution to the *X2Y mapping schema problem* is obtained by assigning each of the x bins with each of the y bins to reducers. In this manner, we require $x \cdot y$ reducers.

Theorem 15 (Upper bounds from the heuristic) *For a bin size b, a given reducer capacity $q = 2b$, and with each input of sets X and Y being of size at most b, the total number of reducers, the replication of an individual input of the set X (resp. Y), and the total communication cost, for the X2Y mapping schema problem, are at most $\frac{4 \cdot sum_x \cdot sum_y}{b^2}$, at most $\frac{2 \cdot sum_y}{b}$ (resp. at most $\frac{2 \cdot sum_x}{b}$), and at most $\frac{4 \cdot sum_x \cdot sum_y}{b}$, respectively.*

Proofs of Theorems 13, 14, and 15 are given in [1].

6. CONCLUSION

Two new important practical aspects in the context of MapReduce, namely different-sized inputs and the reducer capacity, are introduced for the first time. The capacity of a reducer is defined in terms of the reducer's memory size. We note that processing time is typically proportional to the memory capacity. All reducers have an identical capacity, and any reducer cannot hold inputs whose input sizes are more than the reducer capacity. We demonstrated the importance of the capacity aspect by considering two common mapping schema problems of MapReduce, *A2A mapping schema problem* – every two inputs are required to be assigned to at least one common reducer – *X2Y mapping schema problem* – every two inputs, the first input from a set X and the second input from a set Y – is required to be assigned to at least one common reducer. Unfortunately, it turned out that finding solutions to the *A2A* and the *X2Y* mapping schema problems that use the minimum number of reducers is not possible in polynomial time. On the positive side, we present near optimal heuristics for the *A2A* and the *X2Y* mapping schema problems.

7. REFERENCES

[1] F. Afrati, S. Dolev, E. Korach, S. Sharma, and J. D. Ullman. Assignment problems of different-sized inputs in MapReduce. Technical Report 14-05, Department of Computer Science, Ben-Gurion University of the Negev, 2014. Also appears as a Brief Announcement in International Symposium on Distributed Computing (DISC) 2014.

[2] F. N. Afrati, A. D. Sarma, S. Salihoglu, and J. D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *PVLDB*, 6(4):277–288, 2013.

[3] F. N. Afrati and J. D. Ullman. Matching bounds for the all-pairs MapReduce problem. In *IDEAS*, pages 3–4, 2013.

[4] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*, pages 131–140, 2007.

[5] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for NP-hard problems. chapter Approximation algorithms for bin packing: a survey, pages 46–93. PWS Publishing Co., 1997.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[7] M. T. Goodrich. Simulating parallel algorithms in the MapReduce framework with applications to parallel computational geometry. *CoRR*, abs/1004.4708, 2010.

[8] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[9] D. R. Karger and J. Scott. Efficient algorithms for fixed-precision instances of bin packing and euclidean tsp. In *APPROX-RANDOM*, pages 104–117, 2008.

[10] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *SODA*, pages 938–948, 2010.

[11] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[12] A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-round tradeoffs for MapReduce computations. In *ICS*, pages 235–244, 2012.

[13] J. D. Ullman. Designing good MapReduce algorithms. *ACM Crossroads*, 19(1):30–34, 2012.

[14] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using MapReduce. In *SIGMOD Conference*, pages 495–506, 2010.

[15] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *Proceedings of the 17th international conference on World Wide Web*, pages 131–140, 2008.

Lower Bounds on the Communication of XPath queries in MapReduce*

Foto Afrati
National Technical University of Athens, Greece

Matthew Damigos
Ionian University, Greece

Manolis Gergatsoulis
Ionian University, Greece

ABSTRACT

We present two algorithms, each depending on a different data fragmentation of the XML tree. They both compute XPath queries in MapReduce, by first computing subqueries and then combining their results. We compute the replication rate of each algorithm and show it is less than 2.

1. INTRODUCTION

In this paper, we study how to use MapReduce to compute XPath queries on large XML files. We focus on optimizing the communication cost. It is known that the sequential complexity of evaluating XPath queries on XML trees falls into lower complexity classes with high parallelizable problems [3]. The tree structure of both the data and the query facilitate the low sequential complexity. However, when it comes to using a distributed computational environment to evaluate such queries, and especially when using the MapReduce framework, rigorous work that optimizes the significant performance measures is missing. In starting such an investigation, first we note that, unlike relational databases, XML files have a hierarchical structure that makes distribution to compute-nodes special, in that chunks of data in HDFS are already structured. This structure can be used already in the mappers to compute partial answers to the query [9, 10, 6, 7, 5]. Another approach (which is not discussed in this work) would be to view the data as a collection of one binary relation and a set of unary relations which are distributed to the compute-nodes (mappers) randomly, thus the tree structure of the data cannot be used. This approach however does not seem to have an obvious advantage – although it may be worth being investigated rigoursly in order to figure out its limits.

Communication cost is the size of data transferred among the compute-nodes during a MapReduce job and it affects performance. Communication cost per input is the *replication rate* [4] (one node counts as one input).

In this paper, we give lower bounds on the replication rate for XPath queries on XML trees, taking into account a limit on the size of each compute-node. The size of a compute-node is the number of XML tree nodes stored. The lower bounds we derive for each round are smaller than 2. Actually we give proof of the validity of these lower bounds, by providing an algorithm that achieves this replication rate.

The algorithms presented here use more than one rounds of MapReduce. In the first round, the data are distributed to the compute nodes and subqueries are computed. The next round(s) combine the partial results of the subqueries to compute the final result. For each of the algorithms we assume a different fragmentation of the XML tree. We do not discuss how to implement this fragmentation, which would be necessary for these algorithms to derive also upper bounds on the replication rate.

In particular we present two algorithms, one in Sec. 3, where we do the assumption that a path from the root to any leaf of the XML tree fits in one compute-node, so descendant edges of the query are accommodated. The other algorithm (Sec. 4) accommodates descendant edges in the next rounds after partial descendant-free subqueries are evaluated.

2. PRELIMINARIES

2.1 XML trees and XPath Queries

Consider a directed, rooted, labeled tree t, where its labels come from an infinite set Σ. We denote $\mathcal{N}(t)$ and $\mathcal{E}(t)$ the set of nodes and edges, respectively, of t, and we write $label(n)$ to denote the label of a node n of t. The number d of edges of the unique path through which n is reachable from root of t is said to be the *depth* of n. We define children and descendants of a node if there is an edge or a path, respectively.

We consider two types of trees, those that represent XML documents and those that represent XPath queries. An XML document is represented by a tree (also called *XML tree*) having labels from Σ on its nodes. XPath queries are different from XML trees in three aspects. First, the labels of a query come from the set $\Sigma \cup \{*\}$, where $*$ is the "wildcard" symbol. Second, a query P has two types of edges: $\mathcal{E}_{/}(P)$ is the set of child edges (represented by a single line) and $\mathcal{E}_{//}(P)$ is the set of descendant edges (represented by a double line). Third, a non-Boolean query P has an *output node*,

*This research was supported by the project "Handling Uncertainty in Data Intensive Applications", co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning", under the research funding program THALES.

denoted by $out(P)$, and is represented by a circled node. A *Boolean XPath query* does not have any output node. Without loss of generality, we will only consider Boolean queries here. A *subquery* of Q is a single XPath query having a subset of both the nodes and the edges of Q. Furthermore, given an XML tree t and a node n of t, we say that the tree rooted at n is a *subtree* of t. A subquery is Boolean or has the same output as the query if the output node is in the subquery.

The result of applying a query Q on an XML tree t is based on a set of mappings from the nodes of Q to the nodes of t, called embeddings. An *embedding* from Q to t is a mapping $e : \mathcal{N}(Q) \rightarrow \mathcal{N}(t)$ with the following properties: (1) Root preserving: $e(root(Q)) = root(t)$, (2) Label preserving: For all nodes $n \in \mathcal{N}(Q)$, either $label(n) = *$ or $label(n) = label(e(n))$, (3) Child preserving: For all edges $(n_1, n_2) \in \mathcal{E}_{/}(Q)$, we have that $(e(n_1), e(n_2)) \in \mathcal{E}(t)$, and (4) Descendant preserving: For all edges $(n_1, n_2) \in \mathcal{E}_{//}(Q)$, the node $e(n_2)$ is a proper descendant of the node $e(n_1)$.

The result $Q(t)$, now, of applying a non-Boolean query Q on a tree t is formally defined as follows:

$$Q(t) = \{e(out(Q))|e \textit{ is an embedding from } Q \textit{ to } t)\}.$$

If Q is a Boolean query then the result $Q(t)$ is "*true*", only if there is an embedding from Q to t. A *partial embedding* of the query is an embedding of a subtree of the query on the data tree.

According to Dewey encoding system [1], a unique identifier of the form $x_0.x_1.x_2.\ldots.x_d$ can be assigned to each node n of an XML tree. These labels help to decide whether one node is descendant of another (if and only if the Dewey label of the latter is a prefix of the Dewey label of the former), or what is the distance between nodes on the XML tree.

2.2 MapReduce

We will assume that the reader is familiar with MapReduce (details can be found in [2]). However, we need to explain our setting. Typically, each MapReduce job has a map phase and a reduce phase. If we have a sequence of such jobs, then the reducers of the first job send their data to the mappers of the second job, etc. However, the reducers of the first job may act also as mappers of the second job (if it is convenient for the problem at hand) and thus, distribute the data themselves to the reducers of the second job. This is the approach we take here. Hence, we will talk about compute-nodes, instead of distinguishing between mappers and reducers. There is another unconventionality we adopt. Since we use the algorithms we present to argue for lower bounds on the replication rate, we assume that the mappers of the first job have the ability to send any subtree of the XML to the first reducers. This is not totally unrealistic, since many experiments on XML data do a similar fragmentation as ours, because it is a natural way to obtain XML data from HDFS.

3. XML TREE OF SHORT DEPTH

In this section, we consider XML trees where the root-to-leafs paths fit into main memory of compute-nodes; i.e., the size of each compute-node is larger than the depth of the XML tree.

3.1 Data Fragmentation

The fragmentation of the XML tree is done so that in each compute-node we include one subtree of the data tree. Each subtree is rooted in some data node u and all its leaves are leaves of the data tree. We also include the path from the root of the XML tree to u. As we will prove later, including this path adds little extra cost to the replication rate – while, apparently, prunes more nodes.

3.2 Computing and Combining Subqueries

We name the nodes of the query tree by $n_i, i = 1, 2, \ldots$. E.g., in Figure 1, the tree on the left is an XPath query with 23 nodes.

DEFINITION 1. *If there is a partial embedding from the query to the XML tree that maps node n_i of the query to node u of the data tree such that all the descendants of n_i participate (are mapped on some data node) in the partial embedding, then we say that node u is a n_i-*node.

Note that the same node can be both a n_i-node and n_j-node, for distinct i and j. Thus, by considering partial embeddings, we say that we create *adorned nodes*, where the adornment is a nonempty set of nodes from the query tree. Hence, if n_i is in the adornment set of a data node m_j then m_j is a n_i-node.

After distributing the data, each compute-node calculates partial embeddings of the query and finds *maximal* n_i-nodes, for all i, i.e., the parent of a maximal n_i-node is not a n_j-node where n_j is the parent of n_i on the query tree.

We only distribute to the compute-nodes of second round a) the adorned nodes which have at least one maximal adornment in their adornment set and b) all their ancestors (remember they are in the same compute-node). If we can afford to send all such data nodes to one compute-node, then we begin to adorn more nodes as follows: If a node u with a non maximal adornment n_i has children, each child with adornment n_{i_j}, for all the $n_{i_j}, j = 1, 2, \ldots$ children of the query node n_i, then we maximally adorn u with n_i. We terminate this procedure when we find no more nodes to maximally adorn.

If we decide to apply multiple rounds to combine the partial results from the first round, then use the following observation:

- We call a node *candidate n_i-node* if some of its children are adorned accordingly maximally.
- If a data node u is a candidate n_i-node then all its maximally adorned children must meet in the same compute-node in the next round (otherwise "progress" is not made).

The above multi-round distribution is feasible because we do in each compute-node a special kind of deduplication, so that it never emits two siblings with the same adornment. Now, in the following subsection we calculate the replication rate that results from the kind of data fragmentation we

do. This calculation applies to both the data fragmentation method in this section and to each of the next necessary rounds that combine the subqueries, since in all cases we distribute similarly structured data (only less, when non-adorned nodes are not distributed).

3.3 Analysis of replication rate

We examine the replication rate of the phase where we distribute the data to the compute-nodes. We analyze in detail two special cases in this section.

3.3.1 Two level XML tree with high degree

Here, we assume that the XML tree has a root with m_0 children and each child c_i of the root has qm_i children itself, where q is the size of a compute-node. These are all leaves of the XML tree. Thus the XML tree T has $n = 1 + m_0 + q\Sigma_1^{m_0} m_i$ nodes in total. For convenience in the calculations below, we assume that each compute-node has size $q + 2$.

Each compute-node is identified by a number from 1 to $M = \Sigma_1^{m_0} m_i$. We send each child of the root c_i to m_i compute-nodes and each leaf to one compute-node. We send the root to all the compute-nodes. The total number of compute-nodes we use is $\Sigma_1^{m_0} m_i$.

In particular child c_i is sent to a number of compute-nodes with identifiers (here i can be thought of as the second dot in the Dewey label):

$$x + \Sigma_{j=1}^{i-1} m_j, \quad x = 1, 2, \ldots, m_i$$

Each leaf l_i is sent only to one compute-node. The communication cost is:

$$C = \Sigma_1^{m_0} m_i + \Sigma_1^{m_0} m_i + q\Sigma_1^{m_0} m_i$$

The first term corresponds to the root, the second term to the children of the root and the final term to the leaves. The replication rate is $r = C/n$. Since $m_0 \leq \Sigma_1^{m_0} m_i$, it is easy to prove that $r \leq 1 + \frac{1}{q}$.

3.3.2 XML tree being a full binary tree

Here, we assume the XML tree is a full binary tree with n nodes. Since we have assumed that the size q of a compute-node is larger than the length of the path from the root to a leaf, we have here that $q > \log n$. Again for convenience in the calculations, we assume that the compute-node size is $q + \log n - \log q$ with $q > \log n$.

In this case, each compute-node gets a whole subtree (with its leaves being all leaves of the XML tree) of size q. Thus the depth of this subtree is $\log q$. The nodes in the XML tree that are closer to the root than $\log n - \log q$ are replicated a number of times. In particular, the nodes at distance $\log n - \log q - i$ ($i = 1, \ldots, \log n - \log q - 1$) from the root are replicated 2^i times. Thus communication for each level (distance from root) is:

$$2^{\log n - \log q - i} \times 2^i = 2^{\log n - \log q} = \frac{n}{q}$$

Hence the total communication cost is:

$$(n - \frac{n}{q}) + \frac{n}{q} \times (\log n - \log q)$$

The first term counts for the nodes that are replicated once. By dividing the above by n, replication rate is

$$(1 - \frac{1}{q}) + \frac{1}{q} \times (\log n - \log q)$$

This is approximately $\log n/q$. Since the assumption is that a path from the root to any leaf of the XML tree fits in one compute-node, $\log n < q$.

3.3.3 General Remarks

In order to calculate the replication rate in the general case we combine the intuition from the two cases we analyzed in detail. The calculation is based on the following remark for the case where all roots (call them *primary roots*) in the data tree that define compute-nodes are in the same level (as in the cases we studied in detail, e.g., full binary tree). We believe that this remark can be extended for the general case too.

- The total communication cost for all nodes at any level is the same.

In order to prove this remark, we consider a node in the data tree that is a parent of some primary root. This node adds as much to the communication cost as add all its children, because it is sent to exactly all compute-nodes its children are sent (and no two children are sent to the same compute-node).

4. TALL XML TREES

Here we assume that a root-leaf path may not fit in one compute-node but a neigborhood of radius d_Q in the XML tree can fit, where d_Q is the maximum acceptable depth of a descendant-free (to be defined shortly) subquery.

4.1 Data fragmentation

Consider an XML tree t of depth d_t. Since there are root-to-leaf paths that cannot fit into main memory of the compute-nodes, we aim to split the root-to-leaf paths. Considering a positive number m (which will depend on compute-node size q), we construct a set of fragments for each $i = 1, \ldots, \lceil \frac{d_t}{m} \rceil$ which contains each tree node whose depth is included in the range $[(i-1)\frac{d_t}{m}, i\frac{d_t}{m} + d_Q]$. Furthermore, notice that every two adjacent fragments overlap. In particular, the i^{th} fragment contains the top d_Q nodes from the set $i+1$, where $i = 1, \ldots, \lceil (\frac{d_t}{m}) \rceil$. This overlap ensures that each subquery given by the decomposition described in next section can be completely answered in some fragment.

4.2 Computing and Combining Subqueries

DEFINITION 2. *Let Q be a query tree and $\mathcal{E}_{//}(Q)$ be the set of descendant edges of Q. Then the* descendant-free subqueries of Q *are the queries obtained by eliminating the descendant edges from Q. We denote the set of the descendant-free subqueries of a query Q as $\mathcal{C}(Q)$.*

It is easy to see that for each descendant edge $d = (n_1, n_2)$ in $\mathcal{E}_{//}(Q)$, there is a pair of queries Q_1, Q_2 in $\mathcal{C}_Q$ such that n_2 is the root node of Q_2 while n_1 is a leaf node of Q_1.

Here we need some more definitions. A node n of a subquery Q' in $\mathcal{C}(Q)$, such that there exists a descendant edge (n, m)

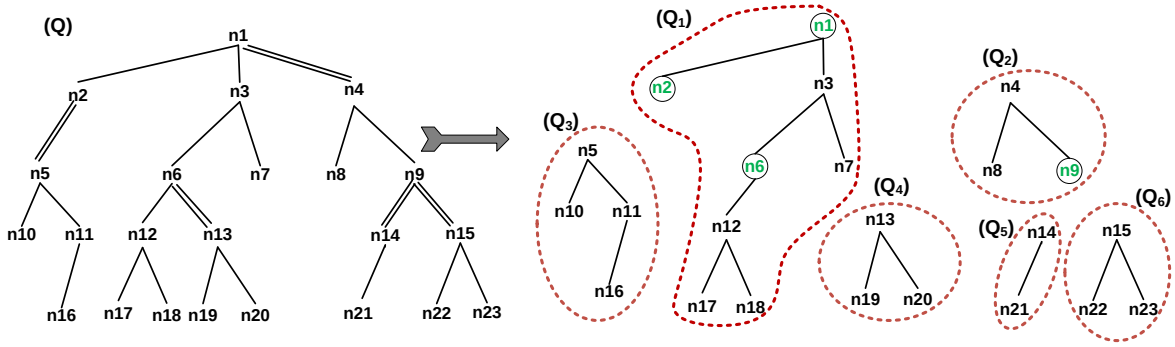


Figure 1: A query Q and its descendant-free subqueries Q_1, Q_2, Q_3, Q_4, Q_5, and Q_6.

in Q, is called a *border node* of Q'. The set of border nodes of Q' is denoted by $\mathcal{N}_{//}(Q')$. A descendant-free subquery Q' that does not contain border nodes is a *leaf subquery* while a subquery that contains a border node is said to be a *non-leaf subquery*.

Example 1. A query tree Q and the set of descendant-free trees $\mathcal{C}(Q) = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6\}$, obtained by its decomposition appear in Figure 1. The set of border nodes of Q is $\mathcal{N}_{//}(Q) = \{n1, n2, n6, n9\}$. Q_3, Q_4, Q_5 and Q_6 are leaf subqueries while Q_1 and Q_2 are non-leaf subqueries.

Definition 3. *Let t be an XML tree, Q be a query and $\mathcal{C}(Q) = \{Q_1, Q_2\}$. Assume that $\mathcal{N}(Q_1) = q_0$ and $q_1 = root(Q_2)$. Let e_1 be an embedding from Q_1 to t such that q_0 maps on data node u and e_2 be an embedding from Q_2 to t such that q_1 maps on data node v. Suppose that v is a descendant of u. The composition of e_1 and e_2, denoted as $e_1 \circ e_2$, is a mapping e from $\mathcal{N}(Q)$ to t such that for each $n \in \mathcal{N}(Q_1)$ then $e(n) = e_1(n)$, otherwise $e(n) = e_2(n)$.*

Evaluation Strategy 1. The query evaluation strategy consists in the following three steps:

1. Decompose the query Q into a set of descendant-free subqueries $\mathcal{C}(Q)$.
2. Evaluate separately each subquery in $\mathcal{C}(Q)$.
3. Combine appropriately (pairwise as per Definition 3) the embeddings of the queries in $\mathcal{C}(Q)$ to find the embeddings of Q.

To combine appropriately the embeddings of the subqueries in $\mathcal{C}(Q)$ we can follow either a multi-round approach or a single-round approach. In the i^{th} round of the multi-round approach, we construct one compute-node for each image u of the i^{th} border node (proceeding bottom-up in that we first consider border nodes that have descendant edges to roots of trees without border nodes). We send to this compute-node all the descendants of u (Dewey label is used here). The trade-off between the two approaches is that the amount of pairs received by a compute-node may exceed the size of the compute-node; while following multi-round approach we perform iterative pruning of the intermediate pairs and we reduce the amount of the pairs sent to each compute-node in each round.

4.3 Replication rate analysis

The replication rate is less than 2 during the data fragmentation, since some of the data are replicated only once and the rest only twice. For the replication rate during the other rounds, we assume again deduplication (in a similar sense as in the first algorithm) in the first round. Thus, each compute-node emits only one (of each n_j-nodes set) descendant of a specific data node. The Dewey labels are used to recognize that. Hence we can assume again that all "relevant" descendants of a specific data node fit in one compute-node.

5. REFERENCES

[1] S. Abiteboul, I. Manolescu, P. Rigaux, M.-C. Rousset, and P. Senellart. *Web Data Management*. Cambridge University Press, 2011.

[2] J. Leskovec, A. Rajaraman, and J.D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[3] G. Gottlob, C. Koch, and R. Pichler The Complexity of XPath Query Evaluation. *PODS, 179–190, 2003*.

[4] F. N. Afrati, A. D. Sarma, S. Salihoglu, J. D. Ullman. *Upper and Lower Bounds on the Cost of a Map-Reduce Computation*. PVLDB, 6(4):277–288, 2013

[5] N. Bidoit, D. Colazzo, N. Malla, F. Ulliana, M. Nolé, and C. Sartiani. Processing xml queries and updates on map/reduce clusters. In *EDBT*, pages 745–748, 2013.

[6] G. Cong, W. Fan, A. Kementsietsidis, J. Li, and X. Liu. Partial evaluation for distributed XPath query processing and beyond. *ACM Trans. Database Syst.*, 37(4):32:1–32:43, Dec. 2012.

[7] M. Damigos, M. Gergatsoulis, and S. Plitsos. Distributed processing of xpath queries using mapreduce. In *ADBIS (2)*, pages 69–77, 2013.

[8] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[9] S. Khatchadourian, M. P. Consens, and J. Siméon. Having a chuql at xml on the cloud. In *AMW*, 2011.

[10] L. Lewandowski. Using Map and Reduce for Querying Distributed XML Data. *MSc. Thesis*, 2012. `http://www.inf.uni-konstanz.de/gk/pubsys/publishedFiles/Lewandowski12.pdf`.

Computing NFA Intersections in Map-Reduce

Gösta Grahne
Concordia University
Montreal, Canada, H3G 1M8
grahne@cs.concordia.ca

Shahab Harrafi
Concordia University
Montreal, Canada, H3G 1M8
s_harraf@encs.concordia.ca

Ali Moallemi
Concordia University
Montreal, Canada, H3G 1M8
moa_ali@encs.concordia.ca

Adrian Onet
Concordia University
Montreal, Canada, H3G 1M8
adrian_onet@yahoo.com

1. INTRODUCTION

Nondeterministic Finite-state Automata (NFA) are simple, yet powerful devices that model a plethora of computationally oriented phenomena. One of the advantages of NFA's is that they are closed under several operations, such as concatenation, intersection, difference, and homomorphic images. This makes NFA's ideally suited for a modular approach, for instance in the context of protocol design and web service composition. A simple, but illustrative example of an e-commerce application designed from components can be found in Chapter 2 in [5]. The salient operation here is the intersection of several finite state automata.

Problems relating to NFA's have been widely studied in the literature. One of the main issues for the NFA intersection problem is that the size of the output NFA is the product of the size of all input NFA's. There is not much hope for improvement, since testing for emptiness of the intersection of a set languages represented by NFA's is known to be PSPACE-complete [8]. The most commonly used algorithm for computing the intersection NFA is to use the *Cartesian construct* for product automata. If there are m input NFA's each having n states, the product NFA will have n^m states. It therefore would be important to come up with good distributed algorithms for the problem.

Google introduced map-reduce as a parallel programming model [4] that can work over large clusters of commodity computers. Map-reduce provides a high-level framework for designing and implementing such parallelism. A growing number of papers deal with map-reduce algorithms for various problems, for instance related to graphs [12, 9, 3, 11], and related to relational joins [2, 6, 7].

In this paper we investigate the problem of implementing the Cartesian construct in map-reduce. We follow the optimization approach of Afrati et al. [1] and analyze the *replication rate* required for computing the product NFA. The replication rate corresponds intuitively to the total amount of communication between the processes in the cluster. We first derive a lower bound for the replication rate in the product computation. We then propose three algorithms for the product computation and analyze their behaviors, thereby obtaining upper bounds for the replication rate. Our study shows that in the case where the size of the alphabet for the NFA's is large and we have a large number of reducers available, an algorithm that distributes the transitions of the input NFA's based on their alphabet symbol achieves an optimal replication rate. For the cases where the alphabet size is smaller than the number of available reducers, a distribution based on both the alphabet symbol and states of the transitions works best. These conclusions are also supported by our experimental results.

The rest of this paper is organized as follows: Section 2 gives the necessary technical definitions, and in Section 3 we derive the lower bound for the replication rate. Section 4 presents and analyzes three concrete algorithms for the problem, and Section 5 describes the experimental results. Conclusions are drawn in the last section.

2. PRELIMINARIES

In this section we introduce the basic technical preliminaries and definitions. We assume familiarity with the map-reduce model (see e.g. [10]).

A *Nondeterministic Finite-state Automaton (NFA)* is a 5-tuple $A = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is a finite set of alphabet symbols, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $s \in Q$ is the start state, and $F \subseteq Q$ is a set of final states. By Σ^* we denote the set of all finite strings over Σ. Let $w = c_1 c_2 \dots c_n$ where $c_i \in \Sigma$ be a string in Σ^*. An *accepting computation path* of w in A is a sequence $(s, c_1, q_1)(q_1, c_2, q_2) \dots (q_{n-1}, c_n, f)$ of elements of δ, where s is the start state and $f \in F$. The *language* accepted by A, denoted $L(A)$, is the set of all strings in Σ^* for which there exists an accepting computation path in A. A language L is regular if and only if there exists an NFA A such that $L(A) = L$.

It is well known that regular languages are closed under intersection. In particular, given NFA's $A_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$, an NFA A, such that $L(A) = L(A_1) \cap L(A_2)$ can be computed by the *Cartesian construct* $A = A_1 \otimes A_2$, where

$$A_1 \otimes A_2 = (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2),$$

and

$$\delta = \{((p_1, p_2), c, (q_1, q_2)) : (p_1, c, q_1) \in \delta_1, (p_2, c, q_2) \in \delta_2\}.$$

The $\otimes$ operation clearly is associative, and can be generalized to a polyadic operator $A_1 \otimes \cdots \otimes A_m$. The Cartesian construct amends itself easily to the map-reduce framework by having the mappers emit transitions (p_i, c_i, q_i) from each NFA A_i, and the reducers output a transition $((p_1, \ldots, p_m), c, (q_1, \ldots, q_m))$ upon receiving inputs (p_i, c_i, q_i), where $c = c_1 = \cdots = c_m$. The crucial question is how to distribute the transitions (p_i, c_i, q_i) over the reducers. This is discussed in Section 4.

3. LOWER BOUND ON THE REPLICATION RATE

Recall that each mapper emits *key-value* pairs (K, V), where K determines the reducer that the pair is sent to. Each reducer receives and aggregates *key-value* lists of the form $(K, V_1, \ldots V_q)$, where the (K, V_i) pairs are emitted by the mappers. The largest list associated with one key is called the *reducer size*, and we will denote it by q. A small q-value ensures that the reducer can perform the aggregation in main memory, and also enables more parallelism. On the other hand, more parallelism usually increases the *replication rate*, which is the average number of key-value pairs that mappers create from one input. The replication rate is intended to model the *communication cost*, that is the total amount of information sent from the mappers to the reducers. The trade-off between reducer size q and replication rate r, is usually expressed through a function f, such that $r = f(q)$. The first task in designing a good map-reduce algorithm for a problem is to determine the function f, which gives us a lower bound of the replication rate r.

To start, we derive a tight upper bound, denoted $g(q)$, on the number of outputs that can be produced by a reducer of size q. We suppose that NFA A_i has $|\delta_i|/k$ transitions for each of the k alphabet symbols. To generate a transition for A, the reducer needs m transitions, one from each NFA A_i. The intersection NFA A has $\frac{|\delta_1| \times \cdots \times |\delta_m|}{k^m}$ transitions, for each alphabet symbol $c \in \Sigma$. As there are k alphabet symbols, the total number of transitions will be $k \times \frac{|\delta_1| \times \cdots \times |\delta_m|}{k^m} = \frac{|\delta_1| \times \cdots \times |\delta_m|}{k^{m-1}}$.

It is known that the product of the elements in a partition with a fixed summation is maximum when the blocks of the partition have equal size. We therefore assume that input data is evenly distributed, so each reducer receives q/m transitions from each NFA A_i. The proceeding gives us the following upper bound on the output of one reducer.

LEMMA 1. *In computing $A = A_1 \otimes \cdots \otimes A_m$ a reducer of size q can cover no more than $g(q) = (q/m)^m$ outputs.*

Using Lemma 1, and the total number of transitions in A, we can get a lower bound on the replication rate as a function of q. As shown in [1] the lower bound is given by the expression

$$\frac{q \times |O|}{g(q) \times |I|},$$

where $|I|$ is the size of input, and $|O|$ is the size of the output. The input size will be the sum of the size of the transition relation of all input NFA's, that is $|I| = |\delta_1| + \cdots + |\delta_m|$. As we saw above, the size of the output in terms of the number of transitions will be $|O| = \frac{|\delta_1| \times \cdots \times |\delta_m|}{k^{m-1}}$. This gives us the lower bound on replication rate for our problem as follows

PROPOSITION 1. *The replication rate r for the Cartesian construct $A = A_1 \otimes \cdots \otimes A_m$ is*

$$r \geq \frac{q \times \frac{|\delta_1| \times \cdots \times |\delta_m|}{k^{m-1}}}{(q/m)^m \times (|\delta_1| + \cdots + |\delta_m|)}.$$

4. ALGORITHMS FOR THE CARTESIAN CONSTRUCT

In this section we propose and analyze three different algorithms for computing $A = A_1 \otimes \cdots \otimes A_m$. Our algorithms compute A in one map-reduce round, as opposed to an $m-1$ round cascade $(\ldots(A_1 \otimes A_2) \otimes \ldots) \otimes A_m$. Since the Cartesian construct shares features with the multiway join problem, and the latter has been shown to work more efficiently when done in one round, as opposed to a cascade [2, 6], we only consider the one-round version in this paper.

We note that the main difference between the NFA intersection and the multiway join problem is that in the latter the only possibility for distributing the tuples is based on the value(s) of the join attribute(s) (corresponding to the alphabet symbols in Σ), whereas the NFA intersection problem we can also distribute the tuples of the transition relation based on the states they involve.

4.1 Mapping based on states

Suppose we have n^m reducers, where n is the maximum number of transitions in any of the input NFA's. In our first algorithm the mappers produce keys of the form $(i_1, i_2, \ldots, i_m)$. Let h be a hash-function with range $\{1, \ldots, n\}$. A transition (p_i, c_i, q_i) from NFA A_i is mapped as key-value pairs $(K, (p_i, c_i, q_i))$, where

$$K = (i_1, \ldots, i_{i-1}, h(p_i), i_{i+1}, \ldots, i_m).$$

for each $i_j \in \{1, \ldots, n\}$. In other words, each transition is sent to n^{m-1} reducers.

In this method, the input and output sizes remain unchanged. However, the function $g(q)$ will be affected by presence of transitions with different alphabet symbols inside a single reducer. This gives us a new upper bound on the number of outputs each reducer can produce, namely $g(q) = k\,(q/mk)^m$. We thus have

PROPOSITION 2. *The replication rate r in the state-based mapping scheme is*

$$r \leq \frac{q \times |\delta_1| \times \cdots \times |\delta_m|}{(q/m)^m \times (|\delta_1| + \cdots + |\delta_m|)}.$$

If n is the maximum number of transitions in any of the input NFA's, the upper bound on the replication rate becomes $r \leq (\frac{nm}{q})^{m-1}$.

By comparing propositions 1 and 2, we observe that the upper bound for the replication rate obtained by mapping based on states exceeds the theoretical lower bound by a factor of k^{m-1}. We conclude that the state-based mapping approach is best suited for situations where the alphabet size is small, e.g., when the alphabet is binary.

4.2 Mapping based on alphabet symbols

In our second algorithm, we have one reducer for each of the alphabet symbols. Thus, the number of reducers is equal to the alphabet size k. The mappers will send each transition (p, c, q) to the reducer corresponding the alphabet symbol c. More precisely, from transition (p_i, c, q_i) of NFA A_i the mapper will generate the key-value pair $(h(c), (p_i, c, q_i))$. Here h is a hash function with range $\{1, \ldots, k\}$. Thus each reducer will output transition $((p_1, \ldots, p_m), c, (q_1, \ldots, q_m))$, having received inputs (p_i, c, q_i) for $i = 1, \ldots, m$.

The total number of transitions sent to all reducers is $\sum_{i=1}^{m} |\delta_i|$ which we approximate by mn, assuming that each A_i has at most n transitions. The replication rate is 1, since every transition is mapped to exactly one reducer. This algorithm works well when the alphabet size k is large and the number of reducers is equal to the number of alphabet symbols. In summary:

PROPOSITION 3. *The replication rate in the alphabet-symbol based mapping scheme is 1, assuming that the number of reducers and alphabet symbols are the same.*

Obviously a replication rate of 1 is optimal. This matches the lower bound of Proposition 1, when observing that each reducer has to process $(nm)/k$ inputs, assuming that the alphabet symbols are uniformly distributed. Substituting $q = (nm)/k$ in the lower bound $(\frac{nm}{kq})^{m-1}$ of Proposition 1, gives $r \geq 1$.

4.3 Mapping based on both states and alphabet symbols

On one hand, if we map the transitions only based on the alphabet symbols, the algorithm does not allow for much parallelism if the alphabet Σ is small. On the other hand, as we have observed, if the transitions are mapped based on states only, the replication rate, and consequently the communication cost, will be sharply increased k^{m-1} times. We therefore consider a hybrid algorithm that maps transitions based on a combination of alphabet symbols and states. In the hybrid method we have a function h_s that hashes states into b_s buckets, and a function h_a that hashes the alphabet symbols into b_a buckets. A transition (p_i, c_i, q_i) from A_i is mapped to reducers $(i_1, \ldots, i_{i-1}, h_s(p_i), i_{i+1}, \ldots, i_m, h_a(c_i))$, for each $i_j \in \{1, \ldots, b_s\}$, and the total number of reducers will be $b_s^{m-1} \cdot b_a$.

To compute the replication rate in this method, we note the input and output sizes $|I|$ and $|O|$ remain unchanged. However, the function $g(q)$ will be affected by presence of transitions with different alphabet symbols inside a single reducer. We will now have $g(q) = \ell(q/m\ell)^m$, where ℓ is the average number of alphabet symbols received by a reducer, or equivalently, $\ell = k/b_a$. From this we can derive the replication rate.

PROPOSITION 4. *The replication rate r in the hybrid mapping scheme is*

$$r \leq \frac{q \times \frac{|\delta_1| \times \cdots \times |\delta_m|}{k^{m-1}}}{(q/m)^m \times (|\delta_1| + \cdots + |\delta_m|)} \times \ell^{m-1}.$$

Assuming that the maximum number of transitions in any of the input NFA's is n, we get $r \leq \left(\frac{nm\ell}{qk}\right)^{m-1}$.

Note that if $b_a = 1$ then $\ell = k$ and there is no hashing on alphabet symbols, and as it can be seen, the replication rate will be equal to the replication rate of the first mapping schema. On the other hand, if $b_a = k$, that is if we hash fully on alphabet symbols, then $\ell = 1$ and as it can be seen, the replication rate will be equal to the replication rate of the second mapping schema.

5. EXPERIMENTS

We conducted some experiments to validate the analysis of the previous section. We computed $A_1 \otimes A_2 \otimes A_3$, and varied the size of the NFA's and number of alphabet symbols. Our experiments were run on Hadoop on a 2-node, personal computer, cluster (8 cores per node running at 3.0 GHz and 24GB memory in total). The number of reducers in the experiments was set to 128. The desktops were running Scientific Linux operating system with kernel version 6.0. The NFA's were generated as labelled random graphs, along the lines of [13]. The total number of transitions were determined by the

transition density, that is, the ratio between the number of transitions and the number of states. In the data shown we used a transition density of 2.0.

In the experiments we compared the execution time obtained by hashing the input data based on states (Method I) and on both states and alphabet symbols (Method II).

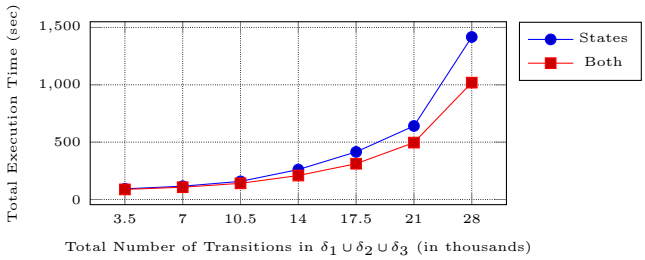


Figure 1: Processing times of two methods for the alphabet size $k = 16$

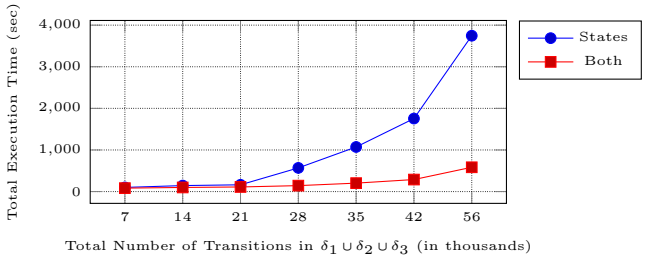


Figure 2: Processing times of two methods for the alphabet size $k = 64$

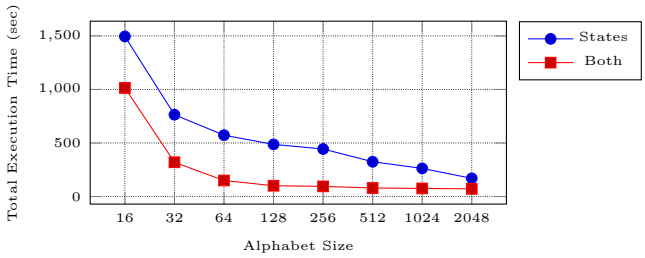


Figure 3: Processing times of two methods where total number of transitions are $28,000$

In Figure 1, we see the execution time for different data sizes with the alphabet size $k = 16$. Figure 2 shows the comparison of Method I and Method II, while the alphabet size $k = 64$. As expected, Method II is clearly more efficient. Figure 3 represents execution time of the two methods for various alphabet sizes when $|\delta_1| + |\delta_2| + |\delta_3| = 28,000$, The figure shows that as the size of alphabet increases, the execution time of both algorithms get closer to each other. This is due to the fact that once the the size of the alphabet exceeds the number of reducers (128), in Method II each reducer has to deal with several alphabet symbols, thus slowing down the computation inside the reducers.

6. CONCLUSIONS

In this paper we proposed and studied methods for computing a product automaton using Map-reduce. Our analysis and experimental results show that carefully optimizing the amount of inter-processor communication indeed pays off in improved processing time.

In future work we will investigate reducing the number of states in the product automaton, either by eliminating all or part of the useless states or by and determinizing and minimizing the automaton.

7. REFERENCES

[1] F. N. Afrati, A. D. Sarma, S. Salihoglu, and J. D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *PVLDB*, 6(4):277–288, 2013.

[2] F. N. Afrati and J. D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Trans. Knowl. Data Eng.*, 23(9):1282–1298, 2011.

[3] F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in map-reduce. In *19th WWW 2010*, pages 231–240. ACM, 2010.

[4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[5] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson/Addison Wesley, 2007.

[6] B. Kimmett, A. Thomo, and S. Venkatesh. Three-way joins on mapreduce: An experimental study. In *IISA 2014*, pages 227–232, 2014.

[7] I. K. Koumarelas, A. Naskos, and A. Gounaris. Binary theta-joins using mapreduce: Efficiency analysis and improvements. In *BeyondMR 2014*, pages 6–9, 2014.

[8] D. Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266, 1977.

[9] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA 2011*, pages 85–94. ACM, 2011.

[10] J. Leskovec, A. Rajaraman, and J. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014.

[11] G. D. F. Morales, A. Gionis, and M. Sozio. Social content matching in mapreduce. *PVLDB*, 4(7):460–469, 2011.

[12] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *20th WWW 2011*, pages 607–614. ACM, 2011.

[13] D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In *LPAR 2005*, pages 396–411, 2005.

Data (Co-)Processing on Heterogeneous Hardware (DAPHNE)

Witold Andrzejewski (Poznan University of Technology),
Sebastian Breß (TU Dortmund University),
Max Heimel (Technische Universität Berlin)

Declarative Query Processing in Imperative Managed Runtimes

Stratis Viglas
University of Edinburgh

ABSTRACT

The falling price of main memory has led to the development and growth of in-memory databases. At the same time, new advances in memory technology, like persistent memory, make it possible to have a truly universal storage model, accessed directly through the programming language in the context of a fully managed runtime. This environment is further enhanced by language-integrated query, which has picked up significant traction and has emerged as a generic, safe method of combining programming languages with databases with considerable software engineering benefits.

Short Bio

Stratis Viglas is a Reader (Associate Professor) in Database Systems in the School of Informatics at the University of Edinburgh, which he joined after receiving his PhD from the University of Wisconsin-Madison in 2003. He has made contributions to data stream processing, XML data management, query processing and optimization, and data management over flash memory. His current work involves integrating managed runtimes with database systems for main memory query processing through just-in-time compilation of SQL queries and incorporating technologies like heterogeneous multicore and persistent memory into the data processing stack.

Local vs. Global Optimization: Operator Placement Strategies in Heterogeneous Environments

Tomas Karnagel, Dirk Habich, Wolfgang Lehner

Database Technology Group
Technische Universität Dresden
Dresden, Germany

(tomas.karnagel, dirk.habich, wolfgang.lehner) @tu-dresden.de

ABSTRACT

In several parts of query optimization, like join enumeration or physical operator selection, there is always the question of how much optimization is needed and how large the performance benefits are. In particular, a decision for either global optimization (e.g., during query optimization) or local optimization (during query execution) has to be taken. In this way, heterogeneity in the hardware environment is adding a further optimization aspect while it is yet unknown, how much optimization is actually required for that aspect. Generally, several papers have shown that heterogeneous hardware environments can be used efficiently by applying operator placement for OLAP queries. However, whether it is better to apply this placement in a local or global optimization strategy is still an open question. To tackle this challenge, we examine both strategies for a column-store database system in this paper. Aside from describing local and global placement in detail, we conduct an exhaustive evaluation to draw some conclusions. For the global placement strategy, we also propose a novel approach to address the challenge of an exploding search space together with discussing well-known solutions for improving cardinality estimation.

1. INTRODUCTION

Column-store database systems have been established over the last years and have demonstrated that they massively benefit from high main memory capabilities and multi-core CPUs. As shown in several papers [1, 7, 10, 13], using such database principle, the speedup of query performance—in particular for OLAP scenarios—compared to classical row-based architectures is immense. Aside from high main memory capabilities and multi-core CPUs, hardware systems are more and more changing towards heterogeneity. That means, a multi-core CPU with large main memory is packed into one single hardware box together with one or more additional non-traditional computing units, e.g., graphic cards, Intel Xeon Phis, or FPGA cores. This heterogeneity trend is going to accelerate and database systems have to exploit this heterogeneity to fulfill increasing performance requirements from available and upcoming applications.

A significant number of research activities has already ported traditional database operators to different computing units like GPU [5, 4], FPGA [11], or many core processors [12]. To tackle the heterogeneity aspect, these ported operators are useful, whereas these operators were always executed on the corresponding computing unit, hoping to reduce the overall execution time. However, to efficiently utilize heterogeneous hardware environments and to reduce the overall query runtime in such environments, it is crucial to assign database operators to the appropriate computing unit for each query separately. This placement assignment has several influencing factors like execution behavior, data characteristics, and properties of available computing units [8].

In order to determine placement assignments, various decision models have been proposed, e.g. *HOP* [8] and *HyPE* [3]. These decision models use information about computing units together with monitored values of previous executions to calculate the estimated execution time in a cost function for each computing unit. A more static approach using instruction counts and execution cycles is also possible to estimate the runtime [5, 6]. Using one of these placement models, the resulting estimation can be deployed to assign an operator to the computing unit with the smallest estimated costs. The mentioned work in this field has proven or provide a high potential for heterogeneous execution. Nevertheless, it is yet unknown, how much optimization is actually required for this placement assignment.

Placement Strategies

In our previous work [8], we identified two strategies for column-store DBMS to support these operator-level placement assignments based on runtime estimations. Both strategies are shown in Figure 1 in the context of query optimization and execution. Both strategies have in common that, after an SQL query is translated into a query execution plan (QEP), a placement decision is made for each operator. In this paper, we assume the operators to be executed at a time with fully materialized intermediate results.

The first placement strategy (*local placement optimization*) conducts an estimation and placement step directly before the execution of each operator and the placement is done for each operator separately. Therefore, the estimation can work on the most recent information about data

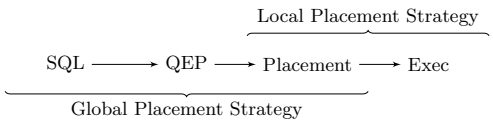


Figure 1: Heterogeneous operator placement strategies.

sizes, allowing an exact estimation of data transfers and execution. Additionally, only one operator is placed at the time, leaving a small search space of the amount of available compute units. However, this approach might be too greedy since the rest of the QEP is not considered in this local decision. In particular, data sharing between operators is hardly considered.

On the contrary, the second strategy (*global placement optimization*) decides the placement for all operators of a QEP before execution. In this case, global placement is done by considering all dependencies of the QEP. This approach yields a high potential for better performance compared to the *local placement optimization*, because data sharing between operators is explicitly encouraged to avoid costly data transfers. However, there is a price for optimizing the whole query for heterogeneous execution. The two main challenges are the huge search space of possible placements and the problem of uncertain or unknown intermediate result sizes.

Contribution

To tackle the issue of how much optimization is required for the heterogeneity aspect, we examine both placement strategies for a column-store database system in detail in this paper. Our main contributions are as follows:

- First, we briefly describe the local placement optimization strategy and present advantages and limitations of this approach (Section 2).
- Second, we introduce the global placement strategy with additional optimizations to tackle the mentioned challenges (Section 3).
- Third, we conduct an exhaustive evaluation to compare local and global placement optimization in an OpenCL based database system (Section 4).
- Finally, we summarize our findings in a property table illustrating the advantages and disadvantages of both approaches.

.

To the best of our knowledge, no one evaluated different query optimization strategies for heterogeneous environments in the past. However, different optimization approaches were mentioned in previous work: local query optimization was used by Breß et al. [2] and Karnagel et al [9] within an OpenCL based column store database system. He et al. [5] computed all possible solutions for separate subplans below a given number of operators and combined the result for the full plan dividing the search space into much smaller problems. However, this is only applicable for tree like query plans and might introduce a significant overhead for large queries.

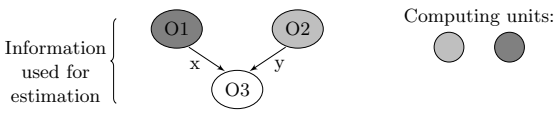


Figure 2: Local placement strategy.

2. LOCAL PLACEMENT STRATEGY

The strategy to integrate operator placement at the execution time of each operator, local optimization, is the most intuitive approach. Placement is decided right before the operator's execution, after previously executed operators have already finished. The input and output data is kept in the computing unit's memory until it will be needed on another computing unit. For local optimization, there are three questions that have to be considered:

1. How big is the input data?
2. Where is the input data placed at the moment?
3. How does the operator perform on the different computing units?

The approach is illustrated in Figure 2. The operators O1 and O2 produce the results x and y. These are stored on the computing units where the operators were executed, here illustrated with different colors. Placement and data size of each input for operator O3 is considered to calculate the transfer costs, if transfer is needed, for the hypothetical execution on each computing unit. The exact data input size is known for base columns as well as intermediate results, since previous operators have already finished their execution. For base columns, the data placement is either in main memory, or already on a compute unit's memory, if an other operator needed the column before. For intermediate results, the data is most likely stored on a computing unit's memory, where the result producing operator was executed. There is the possibility, that data was evicted from the computing unit's memory, if other operators needed additional memory space. However, this should be traceable and the actual memory location should be considered. The third question with respect to the estimated runtime should be answered by one of the prediction models presented in the introduction. Having the transfer time and the operator's execution time estimates, a decision can be made by picking the computing units with the minimal sum of all input transfers costs and execution time. This is the best decision from a local optimization point of view. The search space for this decision is limited to the number of computing units. The decision procedure is repeated for each operator in the order of execution. The result transfer is not considered for the producing plan operator since the data might be reused by the next operator on the same computing unit. If the result transfer is needed, it is added to the costs of the consuming operator instead of the producing one.

The strong advantage of the local placement strategy is its simplicity and easy implementation. The search space corresponds to the number of computing units per decision with one decision per plan operator. Additionally, this approach works on runtime information about data sizes and their placement. Furthermore, the decision is only local by

Op	Runtime		Placement Strategy	
	CU1	CU2	local	global
1	1.2s	0.1s	CU2+tr = 1.1s	CU1 = 1.2s
2	0.1s	1.2s	CU1+tr = 1.1s	CU1 = 0.1s
		Total:	2.2s	1.3s

Table 1: Local vs. global placement strategy. Data transfer (if needed) takes always 1s (tr). The initial data is stored on CU1. The operators are executed according to their ordering.

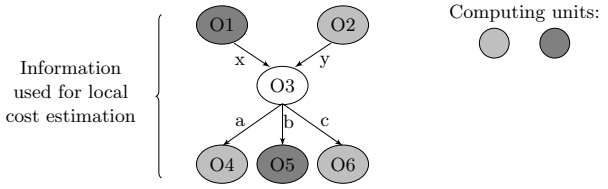


Figure 3: Global placement strategy.

trying to find the ideal execution unit for one single operator. This might not be optimal for the full plan, sacrificing performance through unnecessary data transfers.

3. GLOBAL PLACEMENT STRATEGY

Applying placement at compile time means making the placement decision globally during query optimization. This leads to new possibilities as well as new challenges. An example is shown in Table 1 to highlight the performance potential. The example includes two operators with estimated execution times for two computing units (CU1, CU2). The initial data resides on computing unit CU1 and every data transfer, if necessary, takes 1 second. The presented local strategy would choose CU2 for the first operator, since the run-time plus transfer-time is less than the execution time on CU1. In the second step, it chooses CU1 for the same reason. The total execution time is 2.2 seconds including transfers. For the global strategy, however, the total execution time is only 1.3 seconds since the placement can be globally optimized before execution. Besides the high potential, there are also additional challenges to consider. The two major challenges are (i) the exploding search space of global optimization and (ii) the unknown or uncertain data cardinalities of intermediate results.

3.1 Challenges

Data cardinalities are usually known for base relations but intermediate results are unknown and can only be estimated in the optimization step. However, the exact data cardinalities are crucial for calculating a good heterogeneous placement including correct transfer costs. Since this is a well-known problem in database research, we rely on other research results to provide realistic estimations for the intermediate result sizes.

To the best of our knowledge, the exploding search space for global placement optimization in heterogeneous hardware environment was not in focus of prior research. For a global optimization, every possible placement option has to be considered in order to find the best placement for the full plan. Being $\#cu$ the number of computing units and $\#op$ the number of database operators, then $\#cu^{\#op}$ describes the search space for this query plan. For example, a highly heterogeneous system with 10 computing units, executing a query with 100 operators would lead to 10^{100} possibilities, which is more than all possible 2-way join combinations for 50 joins! To avoid a much larger search space, we assume that, (i) the query execution plan is a DAG (directed acyclic graph) as usual in column-store database system and (ii) the DAG is fixed throughout our heterogeneous placement. That means, the heterogeneous placement do not have any influence on the structure of the DAG. There are approaches to cope with such a large search space in join enumeration. However, the general conditions are different for our heterogeneous placement approach. We identified three properties that define the large search space in heterogeneous execution.

1. The search space does not correlate with the actual runtime. This means, that a query with a large search space can be based on small relations and therefore can execute in a short time. In general, the runtime is highly dependent on the underlying data characteristics, whereas the effort to evaluate the search space stays the same.

2. The search space and the execution time scales with the number of operators.

3. With increasing number of computing units, the query execution time (ideally) reduces, since new computing units might be better suited for some tasks. However, the search space grows exponentially.

The first and the second issue are similar to join enumeration problem, while the third point is unique to heterogeneous execution. However, looking at the first point, a database system that needs to do the join enumeration for, e.g., 50 joins will reserve a fair amount of time for optimizing the order. In our case, dependent on the data sizes, the queries could execute in sub-seconds, leaving only a fraction of that time for efficient optimization.

3.2 Greedy-based Approach

To solve the presented challenges for global optimization, we choose a greedy-based search algorithm together with two approaches for further optimization. We rely on a greedy-based algorithm for several reasons. As mentioned earlier, the search space is too large for a complete search. Optimizing smaller sub-trees is not possible, since we focus on column stores having execution plans as DAGs instead of trees. This means the results of an operator can be used by multiple other operators, making it impossible to define isolated sub-trees. Moreover, a greedy approach makes small changes to improve the placement without considering every possibility.

For our greedy implementation, we start with a pre-set placement decision for every operator. This initial placement could assign the operators randomly to the computing units. Then, we iterate over each operator and evaluate the possible placement decisions locally for this operator. If the algorithm finds a better placement for this operator, we change the decision in the initial placement. The main difference to the local approach is that we already have a

Op	input transfer	Runtime CU1	CU2	Different Placements I	II	III	IV	V
1	1s	1s	5s	1	2	1	1	1
2	1s	1s	0.1s	1	2	2	1	2
3	5s	5s	0.1s	1	2	1	2	2
4	0.5s	1s	5s	1	2	1	1	1
	Total(inc. transfers):			8	11.2	13.1	8.6	3.7

Table 2: Placement cost example. The initial data is on CU1. If needed, the shown input transfer costs apply. The operators execute in order.

placement decision for the following operators, leading to a more informed decision concerning possible data sharing. Figure 3 illustrates this difference. Additional to Operator 1 and 2, the cost function knows the placement of the operators 4 to 6 and the data sizes a, b, and c, therefore being able to calculate inward and outward transfers. Including both kinds of transfers as well as estimates of execution times of each compute unit is leading to a more informed decision than in a runtime-based local optimization. After an optimization iteration over all operators, the changes made on one operator's placement, could influence placement of the previous ones as well. Therefore, the algorithm has to iterate over the operators as long as improvements can be found. When no single placement change of an operator improves the global estimation time, then the algorithm found a (local) optimum.

The above described greedy approach is fast and improves a pre-set starting placement iteratively. However, it is still a greedy approach, which finds a good but possibly not the best placement for the full plan. One reason for not finding the optimal placement is the occurrence of operator groups, that should be placed together. It could be possible that some operators are most beneficially placed together on one computing unit, so that data transfers between them are avoided. However, the best computing unit for the group might not be the best for the single computing unit, so an approach which can only change one placement at the time might not find the best solution. The problem is illustrated in Table 2. Dependent operators, transfer costs, and runtimes are shown. Varying input transfer times correspond to intermediate data sizes, e.g., Operator 2 could be a join with large result, so operator 3 has a high input transfer time. Local optimization would choose the pure CU1 placement (I). For global optimization, the result highly depends on the starting placement. If the starting placement is (I), then (III) and (IV) would be evaluated (besides others) but (I) would be chosen as placement with the minimal costs. With a starting placement of (IV) and assuming the algorithm starts from the top, our global strategy would also evaluate (V) and find it to be the best possible placement.

It is unknown how big these operator groups could be, so it would be a lot of effort to test all groups of two operators, three operators and so on. A more practical idea would be to change the pre-set starting placement and do multiple greedy runs. For example when testing random starting placements, there would be the possibility that some operators of a group are already assigned to the right computing unit, *pulling* the other operators as well. For that, the overall result could be improved by testing many different starting placements and picking the best plan placement according to our execution time estimation. Therefore, we implemented the greedy approach in a hardware-independent OpenCL version, that can test many different starting placements in parallel. This also addresses issue 3 from the previous section. With more computing units, the search space grows but there is also more computing power to evaluate more starting placements for a possibly better solution.

Search Space Reduction

In the previous part, we described our greedy approach and the problem of being dependent on the starting placement. We need to evaluate many different (random) placements, in order to find a good solution. This scales with the search space, meaning that we should test more starting placements with a higher search-space (e.g., for more plan operators). Since we can only evaluate a defined number of placements, we need to reduce the search space to improve the probability of finding a good placement.

We propose to reduce the search space by assigning operators fixed to one computing unit, if the greedy algorithm would pick this computing unit in every possible scenario. For example, Operator 1 and 4 in Table 2 will always be placed on CU1 even if all other operators are on CU2. We call these *strong placements*, where one computing unit is superior in the execution of one operator to an extent that the worst case data transfers are negligible. Since every greedy run for any starting placement would pick these placements, we do not have to consider them in the greedy algorithm as well as in selecting the starting placement. For Table 2, this would mean fixing the placement for Operator 1 and 4, reducing the search space for the other placement decisions from $2^4 = 16$ to $2^2 = 4$. Depending on the computing units and operators, this approach can reduce the search space significantly, even to the point of fixing the placement for the full plan.

The *strong placements* can be calculated by iterating over the plan once for each computing unit and evaluate if a single operator would be placed on another computing unit, even if all other operators are on the initial one. For example, a plan is initially set to CU1. Each operator is tested if a placement on CU2, CU3, and so on, is beneficial for the overall runtime while having all other operators on CU1. This has to be done for each computing unit. If, for example, one operator is always placed on the same computing unit, then this operator can be fixed to this computing unit as a strong placement. Calculating these strong placements introduces only a small overhead by having the potential to reduce the search space significantly.

Majority Voting

After determining the strong placements, the remaining open operator placements can be assigned randomly to the computing units as starting placements for the greedy approach. Here, we deploy the greedy algorithm for many starting placements in parallel, ideally even in parallel on different computing units. As a result, we get the improved placement from the greedy approach and the estimated costs of the full plan. According to the costs, we can choose the best placement for execution.

As an additional step, we look at the output placements and collect statistics on the operator placements. The statistics can be used to find tendencies of the placements. For

	System I		System II	
Vendor	AMD	AMD	Intel	Nvidia
Name	A10-5800K	HD7660D	i7-3960X	K20C
Type	CPU	GPU	CPU	GPU
Cores	4	384	6 (12 HT)	2496
Freq.(MHz)	3800	800	3300	706

Table 3: Heterogeneous test systems: AMD APU (CPU and integrated GPU) and a combination of Intel CPU and Nvidia GPU. The systems computing units are arranged to be balanced in their computational power.

example, if we run 1000 random greedy searches, 200 would pick CU1 for operator 1 and 800 would pick CU2 for the same operator, then we know that CU2 is probably more suited. Using the statistics for all operators, we apply a kind of *majority voting* by combining one common placement from all random runs. This placement is itself evaluated concerning runtime estimation as well as used for a starting placement for another single greedy evaluation.

With the *majority voting* approach, it is possible to combine many good placements to an even better one, which was not found by the greedy algorithm using the random starting placements. However, if the result of the majority voting is not as good as some other placements, the best placement is taken from the random runs.

3.3 Summary

Our approach for global optimization includes an informed greedy algorithm, search space reduction through strong placements, and the majority voting of random starting placements. Therefore, we are able to globally optimize a full QEP. Besides the advantage of global optimization, our global optimization has also limitations. The presented approach is still a greedy strategy which might only find a good solution but not the optimal one. Additionally a small overhead is added to query execution for optimization and reoptimization of the placements.

4. EVALUATION

To evaluate our local and global optimizing approaches, we implemented both in an established database system. For this, we chose Ocelot [7], an OpenCL based extension to the in-memory column store MonetDB [1]. To add heterogeneous hardware support to MonetDB, Heimel et al. implemented this hardware-oblivious extension that allows operators to be executed on most accelerators using the hardware abstraction language OpenCL. Most of the major CPU, GPU, and accelerator manufactures offer OpenCL support for their hardware. When we started, Ocelot did not include dynamic placement of plan operators but rather manual placement of whole queries. However, recent work was also done in this field by Breß et al. [2].

To support our two approaches, we added our self-learning decision model [8], which includes several benchmarks to evaluate data transfer bandwidths. We also included two placement decision units: (i) in the execution engine of the database and (ii) in the plan optimizer.

For the evaluation, we use the slightly altered TPC-H benchmark from Heimel et al. [7]. The benchmark queries

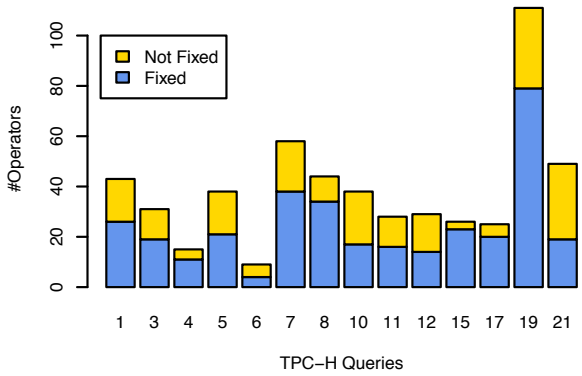


Figure 4: Reducing the search space by assigning strong placements fixed to one computing unit.

are altered to avoid string operations, which are not supported by the Ocelot operators, yet. This is also the reason why some queries were not used for our evaluation. All in all, we tested our approaches on a set of 14 queries from TPC-H.

We evaluated our approaches with two different hardware setups. The two systems are presented in detail in Table 3. Both test systems run with Ubuntu Linux. The first test system is based on an AMD APU with an on-die integrated GPU, which, however, does not support zero copy in our current Linux configuration. That means that data has to be transferred in order to be used by the GPU. The second test system includes an Intel CPU and a Nvidia discrete GPU. Here, memory also has to be transferred to the GPU, since it is attached by PCIe 2.0 and employs a separate GPU processor and GPU memory.

Please note, that heterogeneous placement is needed for any heterogeneous environment in order to utilize all computing units. Depending on the abilities of each computing unit and the *computational balance* between them, a query can be spread over all computing units or alternatively use only that computing unit, which fits best. So we expect for the placement decision, to be at least as good as the fastest computing unit for a query. Finding this fastest computing unit is also a benefit of using a dynamic placement approach. In most cases, it is also possible to improve the fastest single-computing-unit result by applying placement decisions on operator level. To show the effect of the placement decisions, we execute one operator at one time (operator-at-the-time execution model). We do not execute operators in parallel if they are placed on different computing units. Perceived speedups are purely achieved through the placement decisions.

4.1 Search Space Reduction

First, we want to show the effectiveness of our optimizations for the proposed global optimization approach. This is done on System I with the TPC-H benchmark using scale factor 5. First, we reduce the search space by finding strong placements. For TPC-H Query 1 for example, our prototype database system produces a plan with 43 operators, that can be executed on different computing units. For the system with 2 computing units, this results in a search space of:

$$2^{43} = 8,796,093,022,208 \text{ possibilites}$$

With our greedy approach, we do not need to search this

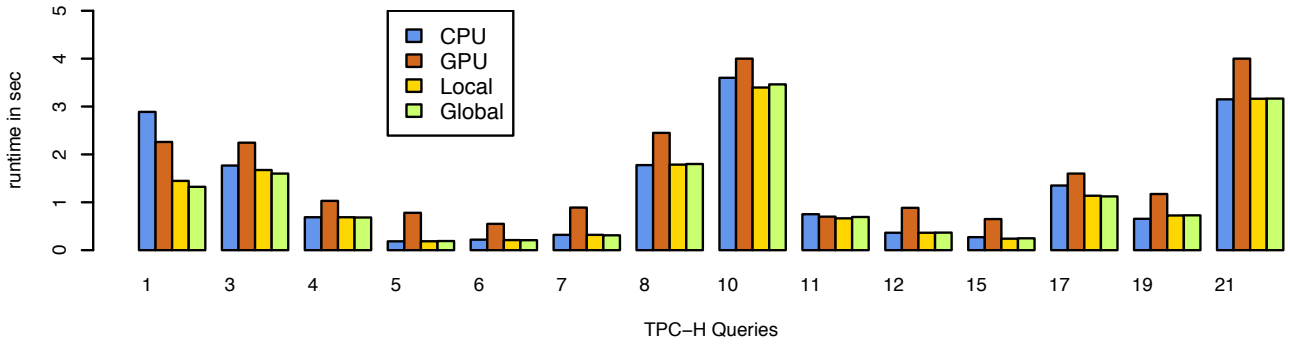


Figure 5: Performance results for TPC-H queries on test system I with SF 5.

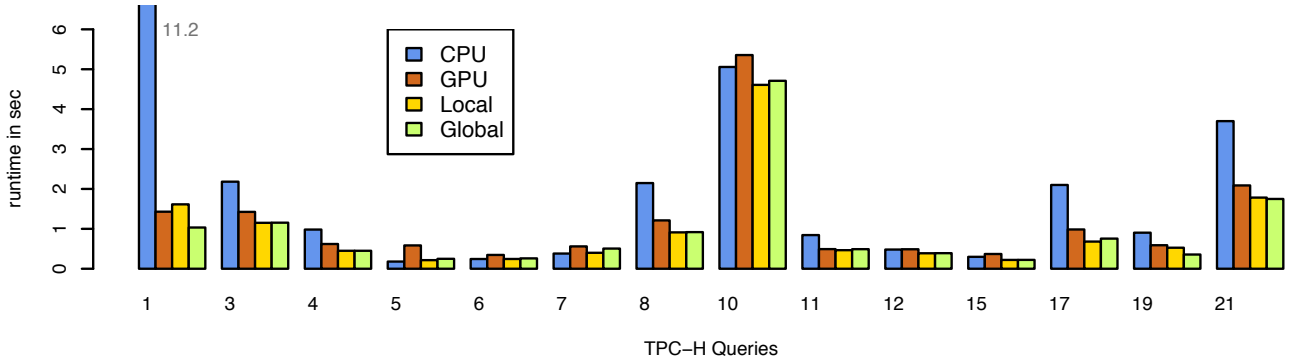


Figure 6: Performance results for TPC-H queries on test system II with SF 10.

high number of possibilities. However, since the algorithm is very dependent on the starting placement, the probability to pick a good starting placement by chance is very low. When we apply our search space reduction, we are able to assign 26 operators to computing units, that would always be placed this way in any greedy search. Removing these operators from the search, reduces the actual searching time as well as the search space for picking random starting placements. The search space for the 17 remaining operators is:

$$2^{17} = 131,072 \text{ posibilities}$$

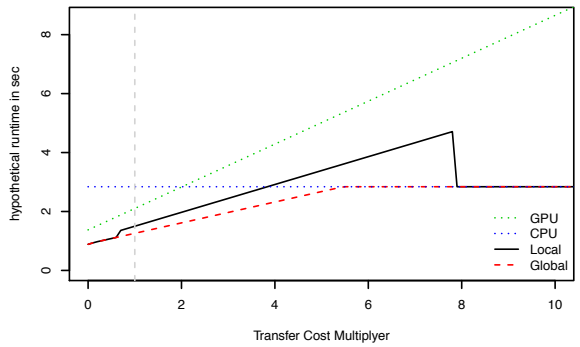
This is still too high to evaluate all possibilities in a fraction of the actual query execution, but it is much more likely to pick a good starting placement for the greedy search. The results for all our TPC-H queries is shown in Figure 4. Please note, that these results could be different for other data sizes (e.g., other scale factors) or in other hardware environments. For example, with a highly superior computing unit, most operators will be assigned as strong placements, while a perfectly balanced environment will have less strong placements.

Please note, that all operators that can be successfully fixed by our global optimization are also chosen in the local optimization, meaning that queries with many strong placements will not differ much between local and global placement decisions.

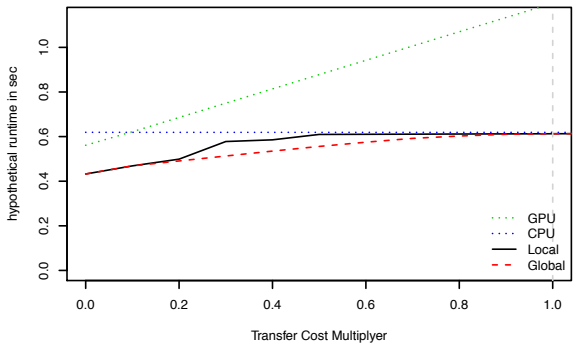
4.2 Greedy Search Performance

After reducing the search space by fixing strong placements, the goal is to evaluate as many starting placements as possible. For that we use our greedy algorithm in different implementations. The actual runtime of one greedy search is highly dependent on the amount of operators in the query plan. Not only one iteration over many operators takes longer, but one single change of an operator results in additional iterations over all operators, to evaluate if this change influence other decisions. The unfixed portion of operators in Figure 4 defines the variable search space. For Test System I, we have seen the naive, single threaded, search performance to be between 5 greedy runs per ms for query 19 (32 variable operators) up to 200 greedy runs per ms for query 6 (5 variable operators). Using OpenCL for the greedy search, we gain a speedup of up to 6x when execution on the CPU. This is to be expected for a 4 core system, since OpenCL also applies vectorization and code optimizations. For the GPU, a speedup of up to 3x can be seen, which indicates in this case that the CPU is more suited for the task. However, all computing units should be used in parallel to evaluate starting placements.

For the final evaluation, we decided to run 100 greedy searches, which takes in the worst case (Query 19) about 4 ms, when using the OpenCL implementation on the CPU. After the first searches, we get the estimated query runtime from the search results. Depending on this runtime, we can decide to do more greedy searches, if the query runtime is high, or to stop the search and start executing the plan, if the query runtime is low. A reevaluation is done every 100 search runs, since the estimated query runtime could improve during optimization. As a general rule, we propose spending about 1% of the total query runtime on optimizing the heterogeneous placement.



9(a) TPC-H Q1 with different transfer cost multiplyers.



9(b) TPC-H Q19 with different transfer cost multiplyers.

Figure 7: Placement performance comparison with varying transfer costs. The transfer bandwidths are taken from System I and multiplied with a transfer cost multiplier.

4.3 Evaluation Results

We compare our two optimization approaches on our set of TPC-H queries by running the queries first on the single computing units and afterwards, we use the gathered knowledge of the operator runtimes to execute the query heterogeneously with local or global optimization. For every query, the initial data is stored in the main memory, meaning that initially no data is cached on the computing units' memories. The results for the first test system are shown in Figure 5. As shown, for some queries the CPU is clearly better and for other queries the GPU is more suited. Heterogeneity-aware operator placement can improve the execution in most cases. In detail, global optimization is always better or equal in performance compared to local optimization. However, the difference is not significant. Further investigations have shown that global optimization finds sometimes the same or only a slightly different plan than local optimization. For the shown results, we used only about 1% of the query execution time for the global optimization. Testing with a higher percentage of optimization did not lead to better results. This shows, that our current global approach is suitable to find a good and possibly the best placement for the given query plan, however, the difference to local decisions is not as significant as having high impact on performance.

On the second test system, the results look similar. Here, the GPU is mostly better for full query execution. Local and global optimization show equally good or better results than the GPU. In some cases however, the local approach is slightly better than global optimization, which is caused by the optimization overhead. On the other side, for Query 1, local optimization is actually slower than the single GPU version, which is caused by its rather uninformed decision process. The local decision involves data transfers to a computing unit and the operators' execution. This makes sense from the execution-time perspective, however, from a global view, additional data transfers could be avoided by considering output transfers.

4.4 Evaluation with Changing Transfer Costs

To investigate effects caused by unnecessary data transfers in more detail, we conduct further experiments with theoretical data transfer properties. As a base line, we use System I, with the measured transfer bandwidth for each computing unit. Then, we introduce a multiplier (M) for the transfer costs, which allows us to adjust the theoretical transfer costs from zero ($M = 0$) to any multiple of the original transfer costs. The results are shown in Figure 7 for TPC-H Query 1 and 19. We can clearly see, that the estimated CPU-only performance is independent of the multiplier since no data needs to be transferred. For the GPU-only version, the initial data transfers of base columns and the final result transfers cause a linear scaling with the transfer costs. For no transfer costs ($M = 0$) local and global optimization always produce the same result, since both approaches solely decide the placement on the operator execution time and data sharing yields no benefit. With increasing transfer costs, the results differ because local optimization only considers input transfers and execution for an operator while global optimization considers execution, input and output transfer.

In Q1 (Figure 7(a)) the gap between the two strategies becomes large for $0.7 < M < 8$. The reason is one operator that is much faster on the GPU than on the CPU. As long as the input transfer costs are smaller than the execution speedup, the operator is placed on the GPU. However, output transfers are much higher and reduce the overall performance to be less than the CPU-only execution. For $M > 8$ the input transfers are too expensive and all operators are placed on the CPU. The global optimization is always better than or equal to the best single-computing-unit execution, being more reliable than local optimization. The effects for Q19 (Figure 7(b)) are similar, however, with a smaller gap between local and global optimization. For the remaining queries, the gaps were even smaller up to the point that, for some queries, local and global optimization chose the same placement for all values of M.

5. CONCLUSION

In this work, we have evaluated two operator placement strategies for heterogeneous hardware environments. The first, local placement optimization at execution time, is easy to integrate but limited on its optimization potential. The second, global placement optimization at compile time, introduces a large implementation effort, with the ability to find a more optimal plan. In this paper, we explained how to implement both strategies, including optimizations to re-

Property	Local Strategy	Global Strategy
1. Search space	+ small	- huge
2. Computational overhead	+ little	- some (can be defined)
3. Cardinalities	+ precisely known	- need to be estimated
4. Implementation	+ simple	- high implementation effort
5. Decision	- local (not fully informed)	+ global (informed)
6. Plan structure	- fixed	+ could be changed
7. Worst-case placement	- worse than single CU	+ best single CU

Table 4: Advantages and disadvantages of local and global placement strategy.

duce the search space and additional evaluations on the outcome of random placements. By applying our implementations and optimizations in an OpenCL-based database system within two test systems, we demonstrated that the global approach achieves better or similar performance than the local approach. However, the speedup is mostly not significant. Additionally, in our evaluation with theoretical transfer costs, we illustrated the effects of these costs and the worst-case performance we can expect from both strategies. While global optimization will always find a plan better than or similar to single-computing-unit execution, local optimization might choose a plan worse than the single-computing-unit execution.

Table 4 summarizes the advantages and disadvantages of both placement strategies. In this paper we presented ways to weaken the disadvantages of global optimization in Point 1 and 2. However, even with our approaches, global optimization achieves mostly a similar performance as local optimization on our test systems. On the other side, in our hypothetical tests, global optimization shows a reliably good performance compared to local optimization. Additionally, with global optimization, the placement decision could influence the physical and logical query plan structure. While this is not the focus of our paper, we would like to mention that changing the plan structure would only be possible with a global approach, where the structure might not be fixed, yet.

In the end, it depends on the use case which strategy is more suitable. From an implementation point of view, local optimization is easier and faster to implement. However, global optimization is more reliable to find a good operator placement as well as enabling plan changes. Especially the last point will be part of our future work.

6. ACKNOWLEDGMENTS

This work is partly funded by the German Research Foundation (DFG) within the Cluster of Excellence "Center for Advancing Electronics Dresden" and by the European Union together with the Free State of Saxony through the ESF young researcher group "IMData" 100098198. Parts of the evaluation hardware were generously provided by Dresden CUDA Center of Excellence.

7. REFERENCES

[1] P. A. Boncz, M. L. Kersten, and S. Manegold. Breaking the memory wall in monetdb. *Commun. ACM*, 51(12):77–85, Dec. 2008.

[2] S. Breß, M. Heimel, M. Saecker, B. Kocher, V. Markl, and G. Saake. Ocelot/hype: Optimized data processing on heterogeneous hardware. *PVLDB*, 7(13):1609–1612, 2014.

[3] S. Breß and G. Saake. Why it is time for a hype: A hybrid query processing engine for efficient gpu coprocessing in dbms. *Proc. VLDB Endow.*, 6(12):1398–1403, Aug. 2013.

[4] N. K. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha. Fast computation of database operations using graphics processors. SIGMOD '04, pages 215–226, New York, NY, USA, 2004. ACM.

[5] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, and P. V. Sander. Relational query coprocessing on graphics processors. *ACM Trans. Database Syst.*, 34(4):21:1–21:39, Dec. 2009.

[6] J. He, M. Lu, and B. He. Revisiting co-processing for hash joins on the coupled cpu-gpu architecture. *PVLDB*, 6(10):889–900, 2013.

[7] M. Heimel, M. Saecker, H. Pirk, S. Manegold, and V. Markl. Hardware-oblivious parallelism for in-memory column-stores. *PVLDB*, 6(9):709–720, 2013.

[8] T. Karnagel, D. Habich, B. Schlegel, and W. Lehner. Heterogeneity-aware operator placement in column-store dbms. *Datenbank-Spektrum*, 2014.

[9] T. Karnagel, M. Hille, M. Ludwig, D. Habich, W. Lehner, M. Heimel, and V. Markl. Demonstrating efficient query processing in heterogeneous environments. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 693–696, New York, NY, USA, 2014. ACM.

[10] S. Manegold, P. A. Boncz, and M. L. Kersten. Optimizing database architecture for the new bottleneck: Memory access. *The VLDB Journal*, 9(3):231–246, Dec. 2000.

[11] R. Mueller, J. Teubner, and G. Alonso. Streams on wires: a query compiler for fpgas. *Proc. VLDB Endow.*, 2(1):229–240, Aug. 2009.

[12] B. Schlegel, T. Karnagel, T. Kiefer, and W. Lehner. Scalable frequent itemset mining on many-core processors. In *Proceedings of the Ninth International Workshop on Data Management on New Hardware*, DaMoN '13, pages 3:1–3:8, New York, NY, USA, 2013. ACM.

[13] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: a column-oriented dbms. VLDB '05, pages 553–564, 2005.

Massively Parallel Analysis of Similarity Matrices on Heterogeneous Hardware

Tobias Rawald
Humboldt-Universität zu Berlin
and
Helmholtz Centre Potsdam - GFZ German Research Centre for Geosciences
trawald@gfz-potsdam.de

Mike Sips
Helmholtz Centre Potsdam - GFZ German Research Centre for Geosciences
sips@gfz-potsdam.de

Norbert Marwan
Potsdam Institute for Climate Impact Research
marwan@pik-potsdam.de

Ulf Leser
Humboldt-Universität zu Berlin
leser@informatik.hu-berlin.de

ABSTRACT

We conduct a study that investigates the performance characteristics of a set of parallel implementations of the recurrence quantification analysis (RQA) using OpenCL. Being an important tool in climate impact and medical research, a central aspect of RQA is the construction of a binary matrix that captures the similarities of multi-dimensional vectors. Based on this matrix, quantitative measures are derived. Starting with a baseline implementation, we diversify its properties along four dimensions: the representation of input data, the materialisation of the similarity matrix, the representation of similarity values and the recycling of intermediate results. We evaluate the performance of five implementations by varying the input parameter assignments, the hardware platform employed for execution and the default OpenCL compiler optimisations status. We come to the conclusion that the performance of conducting RQA highly depends on the selected implementation as well as the combination of these variables under investigation. Differences in runtime of up to one order of magnitude are observed, emphasising the importance of performance studies as presented here.

Categories and Subject Descriptors

C.1.4 [**Processor Architectures**]: Parallel Architectures; G.1.0 [**Numerical Analysis**]: General—*Parallel Algorithms*

Keywords

Similarity Matrix, Parallel Algorithm, Heterogeneous Hardware, Recurrence Quantification Analysis

1. INTRODUCTION

Recurrence quantification analysis (RQA) is a statistical method to quantify the recurrent behaviour of dynamic systems, captured in one or more time series [11]. It has proven its potential in a variety of applications, such as the investigation of the climate system [12] and the early detection of epileptic states [3].

RQA is based on extracting multi-dimensional vectors from time series; each vector corresponds to a reconstructed state of the system at a point in time. To identify recurrences, these vectors are compared regarding their mutual similarities. The results of the comparisons are stored within a binary similarity matrix.

Matrix elements referring to pairs of vectors considered to be similar form vertically and diagonally connected sequences. Using frequency distributions of those lines, RQA derives quantitative measures. They allow to draw conclusions concerning the dynamics of the system under investigation [11].

Focussing on very long time series, in [13] we introduced coarse-grained parallelisation strategies to the problem of RQA. We presented an approach that divides the similarity matrix into multiple sub matrices, computing intermediate results for each sub matrix. This allows to process several sub matrices concurrently. Within a final step, the intermediate results are recombined into a global RQA result.

Even though our approach is independent of the concrete implementation, in [13] we compare a non-parallel version of RQA to a prototype of our approach based on OpenCL, which performs parts of the computation in a massively parallel manner. Exploiting the parallel computing capabilities of modern GPU processors, we achieved drastic performance improvements.

However, executing the prototype on different hardware platforms, we discovered that the relative performance improvements vary. Hence, in this publication we conduct a study that exemplarily examines a selection of factors influencing the overall performance characteristics of RQA.

We provide five implementations, which differ concerning input data representation, similarity matrix materialisation, similarity value representation and intermediate results recycling. Given a specific implementation, we investigate

Time Series:

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}
0.0	0.7	1.0	0.7	0.0	-0.7	-1.0	-0.7	0.0	0.7	1.0	0.7	0.0

$m = 2$ (Embedding Dimension)
$t = 2$ (Time Delay)

Extracted Vectors:

s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
0.0	0.7	1.0	0.7	0.0	-0.7	-1.0	-0.7	0.0	0.7	1.0
1.0	0.7	0.0	-0.7	-1.0	-0.7	0.0	0.7	1.0	0.7	0.0

Figure 1: Vector Extraction. Given a time series capturing the sine function at multiples of $\pi/4$ starting at 0, consisting of thirteen data points. Applying the parameter values $m = 2$ and $t = 2$, eleven vectors are extracted.

the influence of the RQA input parameter assignments, the hardware platform used for execution and whether default OpenCL compiler optimisations are enabled.

The results of our experiments show, that the performance of each implementation highly depends on the combination of hardware platform, default OpenCL compiler optimisations status as well as RQA input parameter assignments. Providing general guidelines, we support the selection of an implementation given a specific RQA scenario as well as computing environment (see Sect. 5.2). Recognising the fact that the exploration space covered is limited, we see this study as a first effort to address the performance comparison of parallel RQA implementations.

We believe that our work, apart from providing highly interest into the nature of RQA, is also relevant for other application areas that face similar problems, including nearest neighbour search.

2. OVERVIEW OF RECURRENCE QUANTIFICATION ANALYSIS

Recurrence quantification analysis is a method in the context of time series analysis [11]. It is based on:

1. extracting multi-dimensional vectors from a set of time series,
2. creating a similarity matrix by calculating pairwise vector similarities, and
3. quantifying small-scale structures within the similarity matrix.

There are several approaches for conducting each of these steps. For the sake of clarity, in this paper we consider performing RQA with the following properties: We are given a single time series consisting of floating point numbers; each value refers to a measurement of an output variable, e.g., the air temperature, of a dynamic system, e.g., the Earth's climate, at a specific point in time. To extract the multi-dimensional vectors, the so called *time delay* method is applied, building on the two parameters *embedding dimension* (m) and *time delay* (t). Starting at the first element of the time series, vectors of size m with the temporal offset t are extracted (see Fig. 1).

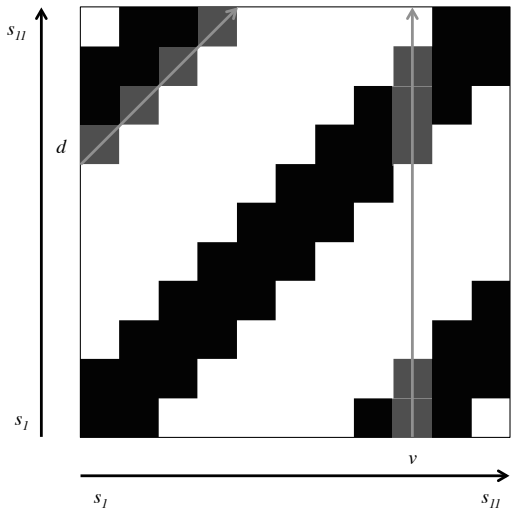


Figure 2: Thresholded Recurrence Plot. Referring to the example from the previous figure, the eleven vectors extracted are compared regarding their mutual similarities. Concerning the vector comparisons, the Euclidean norm is applied, using a similarity threshold of 1.0. The column v contains two lines; one of length 2 and one of length 3. The diagonal d comprises a line of length 4.

To compare those vectors concerning their mutual similarity, a metric such as the Euclidean norm is applied. By introducing a threshold condition regarding the vector similarities, all matrix elements fulfilling the condition are assigned the value 1 (recurrence point), whereas pairs of non-similar vectors are assigned the value 0. A visual representation of this matrix is referred to as thresholded recurrence plot (see Fig. 2). Recurrence points, encoded using the colour black, form vertical and diagonal lines, which are captured in corresponding histograms of line lengths. Based on these histograms, quantitive measures are calculated, including for example the average vertical line length.

3. PARALLEL RQA ALGORITHM

To enable a systematic analysis, we divide the problem of conducting RQA into three operators:

I The creation of the binary similarity matrix. (*create_matrix*)

II The detection of vertical lines within the similarity matrix. (*detect_vertical_lines*)

III The detection of diagonal lines within the similarity matrix. (*detect_diagonal_lines*)

We refine these operators into atomic units of computation:

I The computation of the similarity of a *single pair of multi-dimensional vectors.*

II The inspection of a *single column* of the similarity matrix concerning vertical lines.

III The inspection of a *single diagonal* of the similarity matrix concerning diagonal lines.

Having extracted N multi-dimensional vectors, the maximum degree of parallelism varies between N^2 (I), N (II) and $2N-1$ (III).

Performing similar operations on different data objects, each atomic unit is fully independent of any other unit regarding the execution of a single operator. However, there exist interdependencies between atomic units belonging to different operators: Prior to the detection of lines within a single column or diagonal, the corresponding vector similarities have to be computed.

The structure presented above allows us to perform RQA in a parallel manner. Although subdividing the problem into multiple operators, we mainly focus on the cumulative performance of all operators, regarding the evaluation.

4. EXPERIMENTAL SETUP

4.1 Implementation Strategies

Building on the OpenCL framework, we consider a computing environment that consists of a *host device* and a single *computing device*. The code executed on the host device is written in *Python* 2.7, utilising the package *PyOpenCL*. The atomic units of computation described in Sect. 3 are mapped to OpenCL kernels, implemented in *OpenCL C*.

We provide five RQA implementations, which differ along the following dimensions:

- Input Data Representation,
- Similarity Matrix Materialisation,
- Similarity Value Representation, and
- Intermediate Results Recycling.

In the following, we introduce each dimension and motivate the corresponding values. Regarding the evaluation, we include only a subset of possible value combinations. Nevertheless, we ensure that each value is featured within at least one implementation. Tab. 1 gives an overview of the individual properties of each implementation considered (see Impl. *A*–*E*).

Input Data Representation

Conducting RQA, multi-dimensional vectors are extracted from a time series. Regarding their representation within the memory of the computing device, the set of vectors may either be stored row-wise or column-wise. Choosing a *Row-Store* layout, all components of a single vector are stored consecutively. This requires to reorganise the data given by the input time series.

However, having to perform read-only operations on the vector data, a *Column-Store* layout [16] may be advantageous. Applying this approach, all values belonging to the same vector component are stored contiguously. Since segments of the input time series represent those columns, it can be transferred to the memory of the computing device without having to perform reorganisations.

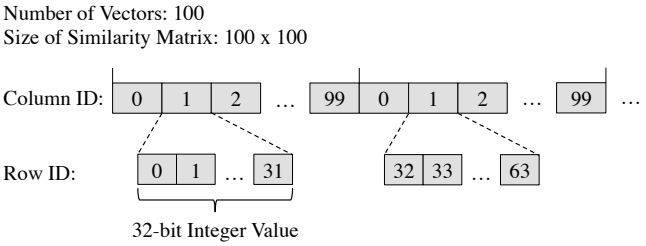


Figure 3: Bitwise Similarity Value Representation. The 32 bits of an integer value refer to a single column. Integer values stored contiguously refer to different columns. Each bit within an integer value refers to a different row of the similarity matrix.

Similarity Matrix Materialisation

The vectors extracted from the time series are compared regarding to their mutual similarities. The resulting binary similarity values are used as input for the detection of vertical and diagonal lines. The corresponding similarity matrix may be stored within the memory of the computing device (*Yes*). This requires that the size of this memory is sufficiently large enough.

Avoiding this restriction, the similarity values may be computed on-the-fly by transferring the computations to the operators for detecting vertical and diagonal lines (*No*). Previous work has shown that the computation of the pairwise similarities requires extensive computing [4]. We are interested, if there are conditions where computing similarity values outperforms writing them to and reading them from the memory.

Similarity Value Representation

Since device memory is a limited resource, the similarity matrix shall be represented in the most efficient manner. Using the bit-compression approach [14], a single bit is used to encode the binary result of a similarity comparison (*Bit*). A schematic illustration of the underlying memory layout is depicted in Fig. 3.

Considering the detection of lines, this approach allows to process up to 32 similarity values of a single column without having to read additional data from the memory. In addition, it ensures that similarity values belonging to different columns are read using a single read instruction.

Nevertheless, this compression approach may introduce a computing overhead, having negative effects on the overall performance. Hence, we compare it to representing a similarity value using the smallest data object addressable (*Byte*).

Intermediate Results Recycling

To avoid matrix materialisation, similarity values may be computed on-the-fly during the line detection process, as explained earlier. Assuming that the execution model adheres to operator-at-a-time, similarity values computed within one line detection operator may be reused later on. Applying this concept of recycling [7], performance improvements may be exposed.

Omitting the *create_matrix* operator, we integrate the materialisation of the similarity values in *detect_vertical_lines* and reuse the results during the detection of diagonal lines

Table 1: Implementation Comparison.

Dimension	Value	Impl. *A*	Impl. *B*	Impl. *C*	Impl. *D*	Impl. *E*
Input Data Representation	Row-Store	✓				
	Column-Store		✓	✓	✓	✓
Similarity Matrix Materisalisation	Yes	✓	✓		✓	✓
	No			✓		
Similarity Value Representation	Byte	✓	✓			✓
	Bit				✓	
Intermediate Results Recycling	Yes					✓
	No	✓	✓	✓	✓	

(*Yes*). Here, the challenge is that the maximum degree of parallelism for detecting vertical lines is significantly smaller than creating the similarity matrix individually (*No*). Thus, our goal is to reveal whether there are conditions under which the positive impact of eliminating one operator is large enough to overcome this limitation.

4.2 Hardware Platforms

We evaluate each implementation using three computing devices. Each device is part of a system that runs on a 64-bit version of *openSUSE*. This includes an *Intel Core i7-3820* CPU running at up to 3.8 GHz, which is supplied with 16 GB of random access memory.

In addition, we employ an *NVIDIA GeForce GTX 690* graphics card, equipped with two GPU processors running at up to 1.019 GHz; each processor is supplied with 2 GB of memory. In the context of our evaluation, only one of those processors is used. The underlying system has version 331.49 of the NVIDIA graphics driver installed.

Adding diversity regarding the GPU architectures, we employ an *AMD Radeon HD 7470* GPU, equipped with a single processor running at up to 0.775 GHz. It is supplied with 0.5 GB of memory. The underlying system has version 14.9 of the AMD Catalyst driver installed.

4.3 Parameter Space

Given the three hardware platforms, we identified the following factors additionally influencing the performance characteristics:

- the parameters steering the properties of the similarity matrix, including:
  - the time series,
  - the embedding dimension,
  - the time delay,
  - the similarity measure, and
  - the similarity threshold, as well as
- the default OpenCL compiler optimisations.

To restrict the exploration space, we reduce the number of degrees of freedom addressed within the evaluation to two. This includes varying the embedding dimension between 1 and 32. Moreover, we observe the impact of disabling the default OpenCL compiler optimisations using the compiler flag *-cl-opt-disable*. We consider evaluating the impact of those optimisations as highly relevant, since they are vendor specific and may affect the computing results, e.g., the default activation of relaxed math operations on the NVIDIA GPU.

We employ a time series capturing the sine function, similar to Fig. 1, consisting of 10,000 data points. We choose this rather short length since we have to ensure that the resulting similarity matrix fits into the memory of all computing devices applied.

Regarding the similarity comparisons, we select the Euclidean norm in combination with a threshold of 1.0. Initial experiments have shown that the time delay parameter does not have considerable influence on the performance. Hence, we set this parameter to 2.

5. EVALUATION

5.1 Procedure

Concerning the evaluation, we consider an experiment to be a combination of:

- hardware platform,
- implementation,
- embedding dimension, and
- default OpenCL compiler optimisations status.

To reduce the impact of outliers, we conduct each experiment five times. For the purpose of measuring the runtime behaviour of the implementations, we rely on *profiling events* as part of the OpenCL API, collecting information about the average runtime of the three operators. Furthermore, we use the *sprofile* [1] command line tool to retrieve extended performance information provided by the AMD GPU.

5.2 General Guidelines

The cumulative runtime results are depicted in Fig. 4, having the default OpenCL compiler optimisations disabled, and Fig. 5, having them enabled.

As expected, increasing the dimensionality of the vectors, the runtime increases as well. Enabling the default compiler optimisations has a positive impact on the cumulative runtime, independent of the implementation as well as the hardware platform employed. Considering the GPU devices, implementation *A*, using a row-wise layout for storing the multi-dimensional vectors, benefits the least. Whereas the relative difference in runtime between *A* and the other implementations is narrow considering the CPU, it widens more

drastically regarding the GPU devices. Hence, considering GPU devices, a row-wise layout should be avoided.

Compared to the other implementations, B shows well-balanced performance characteristics, relying on the column-wise memory layout. Applying an embedding dimension of 32, it is among the two fastest implementations independent of the hardware platform applied. Eliminating the similarity matrix materialisation, implementation C delivers performance improvements considering small embedding dimensions, as expected.

The usage of the bit-representation in implementation D proves to be reasonable for larger embedding dimensions. The corresponding runtime curves start at a higher plateau, but have the smallest slope, independent of hardware platform and default compiler optimisations status. Diminishing the compression overhead with increasing dimensionality, the curves of D converge towards the corresponding curves of B.

Recycling intermediate results, as employed in implementation E, does not present runtime benefits across all hardware platforms. Considering the CPU, it is the fastest implementation, for nearly all embedding dimension values. Regarding the GPU devices, E delivers runtime improvements for vectors having small dimensionality, but is eventually outperformed by implementation B and D.

Considering a given hardware platform, time series as well as RQA input parameter assignments, we propose employing an implementation that comprises the following features:

- column-wise input data representation,
- materialisation of the similarity matrix,
- byte representation of the similarity values, and
- usage of a separate *create_matrix* operator.

Although this combination does not deliver the best performance under all circumstances, it appears to be a reasonable choice based on the evaluation results.

5.3 Detailed Performance Analysis

We present selected details on the impact of using different implementation strategies. The runtime results as well as the performance counter values listed below refer to an embedding dimension of 32.

Input Data Representation

Comparing the hardware platforms applied, the row-store layout for representing the vectors has the least worst impact considering the CPU. Having the default compiler optimisations disabled, the *create_matrix* operator of implementation A (0.79s) is as nearly as fast as the same operator of B (0.75s). Enabling the optimisations, creating the matrix in A (0.44s) consumes twice as much runtime as in B (0.22s).

Additionally, the impact of changing the access pattern to the device memory is illustrated by the cache hit rate produced on the AMD GPU. Disabling the compiler optimisations, the *create_matrix* operator of A has a rate of 23.39%, whereas executing the same operator of B results in a rate of 91.36%.

Similarity Matrix Materialisation

Not materialising the similarity matrix presents advantages concerning the cumulative runtime using small embedding dimensions. Regarding the NVIDIA GPU, the break-even point of implementation B and C is a dimensionality of 3.

Experiencing a drastic increase in fetch operations, the ratio between the amount of *arithmetical logical unit* (ALU) instructions performed by the AMD GPU in comparison to the number of *fetch unit* instructions decreases; from 17.97 (B) to 2.25 (C) regarding the detection of vertical lines, having the default optimisations enabled.

Similarity Value Representation

Encoding similarity values using a single bit leads to an increase in ALU instructions for all three operators, reflecting the corresponding computing overhead. However, the custom layout presented in Sect. 4 improves the memory access. Considering the AMD GPU, this results in an increased cache hit rate for detecting diagonal lines; from 3.26% (B) to 21.52% (D), having the default compiler optimisations enabled.

Intermediate Results Recycling

Focussing on the execution on the CPU, the reuse of similarity values in E is advantageous compared to any other implementation. Enabling the default OpenCL compiler optimisations, implementation E (0.31s) outperforms its direct successor B (0.37s), regarding the cumulative runtime.

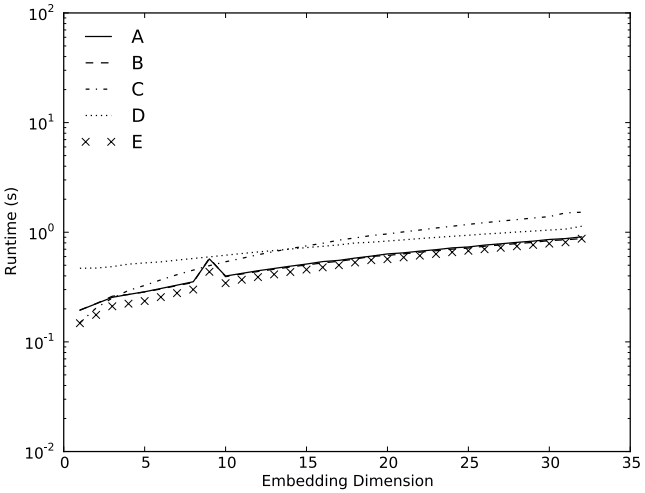
6. RELATED WORK

A number of RQA implementations are available, posing restrictions concerning the size of the similarity matrices that can be processed [10, 17]. The *Commandline Recurrence Plots* (CRP) software allows to analyse time series of arbitrary size [9]. However, it relies on computing the RQA measures using a single CPU thread. For an overview of free RQA software, we refer to [2].

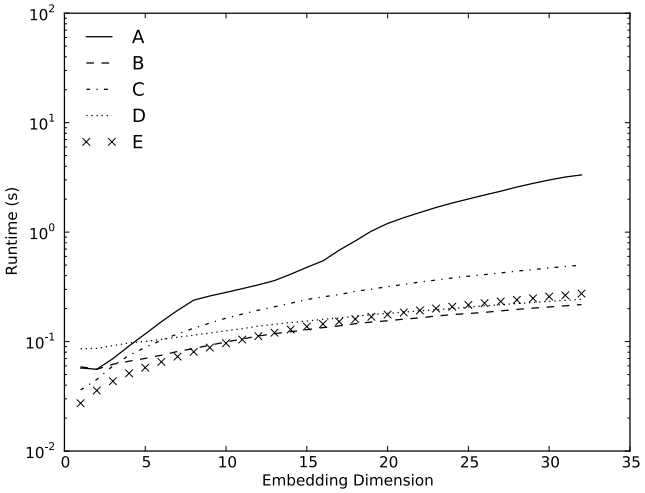
In [15] prior efforts to bring RQA to the GPU are described, comprising several limitations that hamper the analysis of long time series. This includes being restricted to similarity matrices that fit into the memory of the GPU device. Relying on the concepts of *Divide & Recombine* [6], our approach presented in [13] allows to process similarity matrices of arbitrary size. We demonstrated the capabilities of our approach for a specific RQA scenario from climate impact research. Examining a time series consisting of over one million data points, we were able to reduce the runtime from over six hours, using the CRP software, to almost five minutes, using an OpenCL implementation of our approach running on two GPUs.

Considerable efforts have been made to accelerate database operations. Exploiting the computing capabilities of general-purpose graphics cards, in [5] several parallel implementations for database operations, such as semi-linear query, are presented. The conclusion is that depending on the operation investigated, GPUs enable drastic performance improvements.

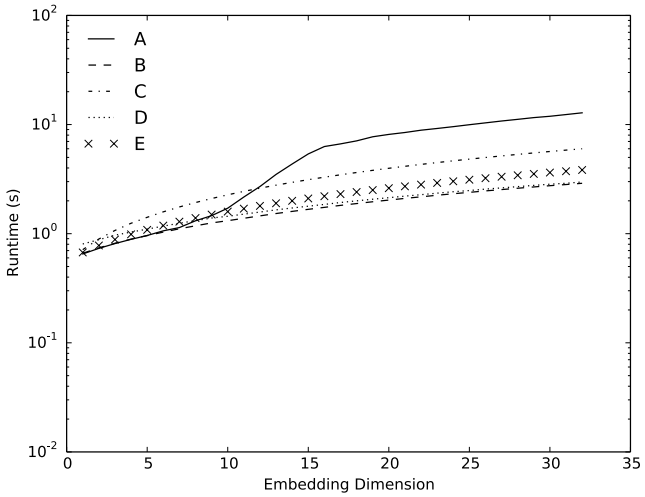
A prominent database operation similar to RQA is the k-nearest neighbour search (kNN). Within both techniques, comparing a set of objects regarding their mutual similarities is a key aspect. Adapting kNN processing to many-core systems, a large amount of similarity comparisons is performed concurrently. Experimental results illustrate that executing a parallelised version of the algorithm on the GPU is two orders of magnitudes faster than performing the search on the CPU [4].



(a) Intel Core i7-3820

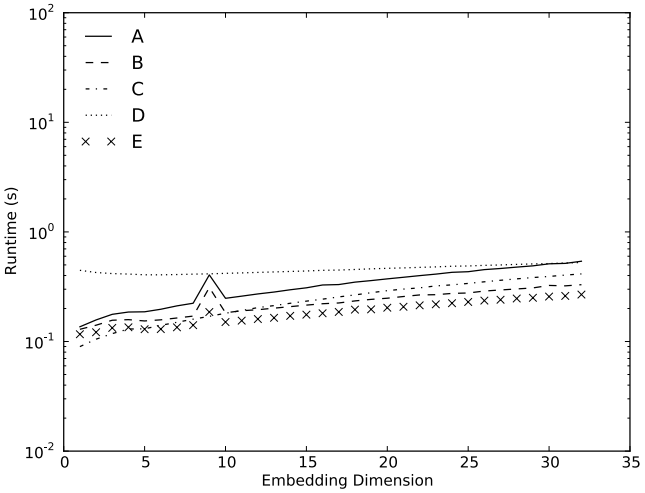


(b) NVIDIA GeForce GTX 690

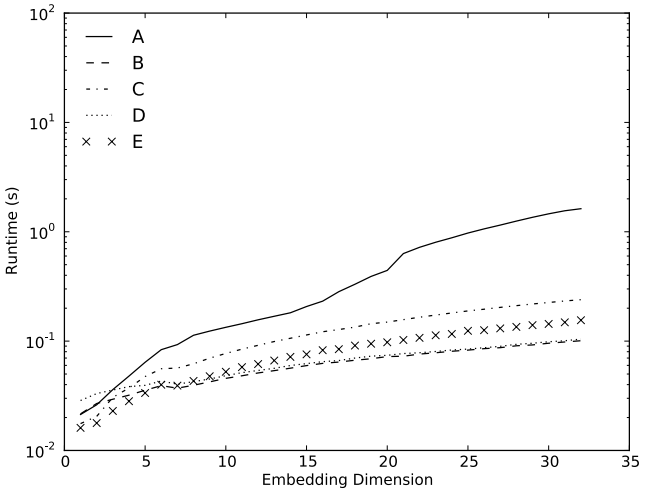


(c) AMD Radeon HD 7470

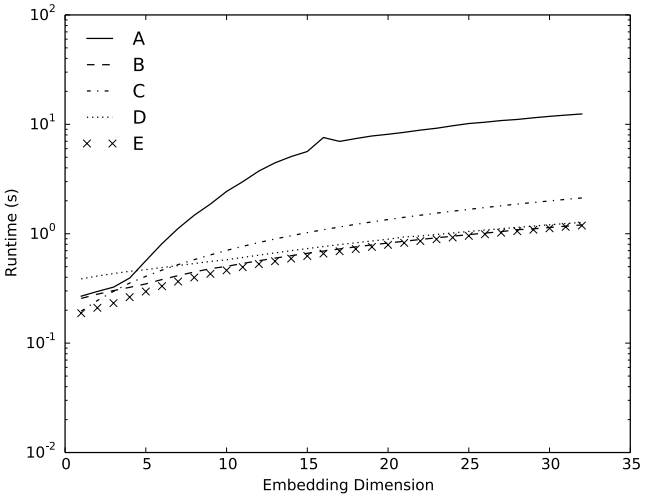
Figure 4: Disabling Default OpenCL Compiler Optimisations. Cumulative runtime of executing the operators *create_matrix*, *detect_vertical_lines* and *detect_diagonal_lines*.



(a) Intel Core i7-3820



(b) NVIDIA GeForce GTX 690



(c) AMD Radeon HD 7470

Figure 5: Enabling Default OpenCL Compiler Optimisations. Cumulative runtime of executing the operators *create_matrix*, *detect_vertical_lines* and *detect_diagonal_lines*.

Previous work focussed on employing a set of optimisations to gain runtime improvements on a specific device. To the best of our knowledge, we provide the first structured approach to analyse the performance characteristics of parallel RQA implementations. In this regard, we benefit from using the OpenCL framework for heterogeneous computing [8], which allows us to execute identical code on a variety of hardware platforms.

7. CONCLUSION

We present a structured approach to evaluate the performance of five parallel implementations analysing binary similarity matrices in the context of RQA. Assessing the performance of each implementation, we vary their characteristics along four dimensions, including the representation of input data, the materialisation of the similarity matrix, the representation of the similarity values as well as the recycling of intermediate results.

Building on the OpenCL framework, we investigate the influence of the hardware platform used for execution, input parameter assignments and default OpenCL compiler optimisations enabled on the performance. We examine the runtime behaviour as well as additional indicators, e.g., the cache hit rate. We come to the conclusion, that an implementation using column-wise input data representation in combination with similarity matrix materialisation provides reasonable performance, regarding a given RQA scenario. Subsuming, we see our study as a first effort towards a comprehensive analysis of parallel RQA implementations.

8. ACKNOWLEDGEMENTS

This work is supported by grants from the Deutsche Forschungsgemeinschaft, Graduiertenkolleg METRIK (GRK 1324).

9. REFERENCES

[1] Advanced Micro Devices, Inc. APP Profiler Settings. `http://developer.amd.com/tools-and-sdks/archive/amd-app-profiler/user-guide/app-profiler-settings/`, 2014.

[2] J. Belaire-franch and D. Contreras. Recurrence plots in nonlinear time series analysis: Free software. *Journal of Statistical Software*, 2002.

[3] K. C. Chua, V. Chandran, U. R. Acharya, and C. M. Lim. Computer-based analysis of cardiac state using entropies, recurrence plots and Poincare geometry. *Journal of Medical Engineering & Technology*, 32(4):263–272, 2008.

[4] V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using GPU. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–6, 2008.

[5] N. K. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha. Fast Computation of Database Operations Using Graphics Processors. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 215–226, New York, NY, USA, 2004. ACM.

[6] S. Guha, R. Hafen, J. Rounds, J. Xia, J. Li, B. Xi, and W. S. Cleveland. Large complex data: divide and recombine (D&R) with RHIPE. *Stat*, 1(1):53–67, 2012.

[7] M. G. Ivanova, M. L. Kersten, N. J. Nes, and R. A. Gonçalves. An Architecture for Recycling Intermediates in a Column-store. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 309–320, New York, NY, USA, 2009. ACM.

[8] Khronos Group. OpenCL 1.1 Specification. `http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf`, Sept. 2010.

[9] N. Marwan. Commandline Recurrence Plots, Version 1.13z. `http://tocsy.pik-potsdam.de/commandline-rp.php`, 2006.

[10] N. Marwan. CRP Toolbox, Version 5.17. `http://tocsy.pik-potsdam.de/CRPtoolbox`, 2013. platform independent (for Matlab).

[11] N. Marwan, M. C. Romano, M. Thiel, and J. Kurths. Recurrence Plots for the Analysis of Complex Systems. *Physics Reports*, 438(5–6):237–329, 2007.

[12] D. I. Ponyavin and N. V. Zolotova. Cross Recurrence Plots Analysis of the North-South Sunspot Activities. volume 2004, pages 141–142, 2005.

[13] T. Rawald, M. Sips, N. Marwan, and D. Dransch. Fast Computation of Recurrences in Long Time Series. In *Translational Recurrences. From Mathematical Theory to Real-World Applications*, volume 103 of *Springer Proceedings in Mathematics & Statistics*, pages 17–29. Springer International Publishing, 2014.

[14] M. A. Roth and S. J. Van Horn. Database Compression. *SIGMOD Rec.*, 22(3):31–39, Sept. 1993.

[15] T. Rybak. Using GPU to Improve Performance of Calculating Recurrence Plot. `http://www.wi.pb.edu.pl/pliki/nauka/zeszyty/z6/Rybak-full.pdf`, 2010.

[16] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: A Column-oriented DBMS. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 553–564. VLDB Endowment, 2005.

[17] C. L. Webber Jr. RQA Software, Version 14.1. `http://homepages.luc.edu/~cwebber`, 2013. only for DOS.

Energy Data Management (EnDM)

Torben Bach Pedersen (Aalborg University),
Wolfgang Lehner (TU Dresden)

Enhancing energy awareness through the analysis of thermal energy consumption

Andrea Acquaviva[†], Daniele Apiletti[†], Antonio Attanasio[† §], Elena Baralis[†], Federico Boni Castagnetti[‡], Tania Cerquitelli[†], Silvia Chiusano[†], Enrico Macii[†], Dario Martellacci[‡], Edoardo Patti[†]

[†] Dipartimento di Automatica e Informatica, Politecnico di Torino, ITALY

[§] Istituto Superiore Mario Boella, Torino, ITALY – [‡] IREN Energia Torino, ITALY

[†] {name.surname}@polito.it – [§]attanasio@ismb.it – [‡]{name.surname}@gruppoiren.it

ABSTRACT

Energy efficiency by means of reduction in wasteful energy consumption is a growing policy priority for many countries. Innovative systems should be designed to continuously monitor a smart city environment and provide all stakeholders the tools to improve energy efficiency. This paper presents the EDEN platform, designed to collect and analyze thermal energy consumption of residential and public building heating systems. EDEN is being deployed in a major Italian city and collects energy consumption measurements through an extensive smart metering grid involving thousands of buildings. EDEN also collects and analyzes indoor climate conditions, and user feedbacks, such as their thermal comfort perception, by means of an ad-hoc social network. Collected data are further enriched with temporal and spatial information at different abstraction levels and meteorological data available as an open source data set. Several technical Key Performance Indicators (KPIs) have been defined to inform users on their building thermal energy consumption, while user-friendly KPIs present energy savings or over-consumptions in an informative fashion.

1. INTRODUCTION

In the last few years, the interest in urban data computing is continuously growing both in the industrial and research domains, as well as in the Public Administration. Industries are attracted by the business opportunities arising from the design, implementation, and exploitation of novel technologies and applications to effectively support all the crucial aspects of Smart Cities management. Researchers, instead, are interested in the challenging issues coming from the application of innovative data management and mining techniques to new and more complex fields. Innovative systems should be designed to continuously monitor a smart city environment and suggest new ways to improve the quality of life within an urban environment, for both citizens and the Public Administration. A complete overview of the key challenges of urban computing from the computer scientists perspective is presented in [25]. Among the large variety of applications available in the context of smart cities, this paper focuses on energy consumption, and specifically on thermal energy consumption in buildings during the winter period. The goal is to improve energy infrastructures and reduce energy consumption, and the associated costs, by suggesting energy-saving strategies to users and by providing better information to the different people involved in the energy management roles.

Energy efficiency is a growing policy priority for many countries around the world, as governments seek to reduce wasteful energy consumption and encourage the use of renewable sources. The International Energy Agency (IEA) has estimated that in terms of primary energy consumption, buildings represent roughly 40% of total final energy consumption in most countries. The amount of this energy used for heating and cooling systems is about 60% in the residential sector and 45% in the service one [12].

Important research activities have been carried out to use database management systems and exploratory data mining techniques in the field of storage and analysis of energy data to evaluate the efficiency of buildings. The proliferation of sensor networks for monitoring indoor and outdoor environmental parameters [16, 19] has brought to the facility managers huge archives of measures with temporal and spatial references. Research contributions on these large data volumes have been carried out for: (i) supporting data visualization and warning notification [17, 20, 24]; (ii) efficient storing and retrieval operations based on NoSQL databases [19, 23]; (iii) discovering anomalous behaviors using clustering algorithms [6, 24], Support Vector Machines (SVM) [9] and outlier detection [21]; (iv) characterizing consumption profiles among different users [2, 9, 20]; identifying the main factors that increase energy consumption (e.g., floors and room orientation [10], location [9, 14]).

In this paper we describe the Energy Data ENgagment platform, EDEN, designed to monitor and analyze thermal energy consumption of heating systems for enhancing user energy awareness. EDEN collects data from smart meters deployed in thousands of buildings in Turin, a major Italian city. EDEN also collects and analyzes indoor climate conditions by means of temperature sensors installed in a subset of the monitored buildings. Thermal comfort perception and user feedbacks on indoor climate conditions are also collected by means of an ad-hoc social network. Collected data are further enriched with temporal and spatial

information at different abstraction levels, and meteorological data available as an open source data set. Several technical and user-friendly Key Performance Indicators (KPIs) are defined within EDEN targeting different users. A *technical KPI* informs users on their building thermal energy consumptions, while a *user-friendly KPI* explains monetary savings or overconsumption by converting its value into the price of commonly purchased goods. EDEN is designed, developed and experimented within the context of a publicly-funded research project, including both academic and industrial partners that contribute to make it a live platform, with actual deployment and real data.

This paper is organized as follows. Section 2 discusses our vision towards enhancing energy awareness through the Energy Data ENgagment platform. Section 3 describes the main building blocks of the proposed system. For some blocks, we describe our first implementation to show both the feasibility and high potential of the proposed approach. Section 4 reports a preliminary analysis of thermal energy consumption for 2 school buildings and 6 residential buildings located in Turin. Section 5 draws conclusions and presents future developments of this work.

2. PLATFORM OVERVIEW

Figure 1 shows the overall architecture of the EDEN system. In this study we focus on an instance of EDEN tailored to an indoor heating monitoring system. However, the EDEN architecture can be easily tailored to different indoor monitoring contexts, such as electric cooling, and outdoor monitoring applications as well. It includes three main components, named *Data Platform*, *Publication Platform*, and *Social Platform*, briefly described below and detailed in the following sections.

EDEN is designed for the collection, storage, modeling, and analysis of a large amount of heterogeneous data to provide different levels of relevant knowledge. The aim is to make people aware of their energy and thermal consumptions, as well as encouraging them to pursue energy saving strategies. Collected data include energy consumption logs provided by thermal smart meters and indoor climate conditions monitored through indoor temperature sensors. In addition, data on the user thermal comfort perception of indoor climate conditions and user feedbacks are gathered through an ad-hoc social network. Heterogeneity in terms of formats, timings and sampling periods, and sources presented a challenge to the designers, also considering the changes over time of this factors, determined by contexts (e.g., smart meters update) or design improvements. To this aim, EDEN exploits a non-relational schema-free data warehouse, which allows coping with frequent changes in data formats without technological issues. This component will be detailed in Section 3.3.

Energy consumption data are collected by means of a large number of smart meters (4,000 as of December 2014) deployed in Turin (Italy) by IREN [13] to monitor thermal energy for district heating. IREN is a multi-utility company listed on the Italian Stock Exchange and operates in the sectors of electricity, thermal energy for district heating, gas, management of integrated water services as well as the collection and disposal of waste.

Data on energy consumption and on monitored indoor climate conditions, collected through sensors and smart meters, are stored in the *Data Platform* component. These data are enriched with spatial and temporal information at different granularity levels as well as with various meteorological conditions. The enriched dataset is stored in a datawarehouse and is managed by the *Publication Platform* component. Specifically, an informative dashboard is generated based on a selection of Key Performance Indicators (KPIs) to produce useful feedbacks to the different users and suggests ready-to-implement energy efficient actions or strategies. Mainly, the following two classes of KPIs have been proposed. (i) *Technical KPIs* allow informing users on the thermal energy consumption of their building, but also comparing the consumption between buildings in the same neighborhood, also in different time periods. Comparison can be performed under similar meteorological conditions. (ii) *Informative and user-friendly KPIs* present the results of the analysis on energy savings and overconsumption in an informative fashion, using simple and easily understandable comparisons according to the user profile. For example, let us consider the energy consumption of a secondary school, and suppose that we would like to improve students' energy awareness. An informative and user-friendly KPI can provide the school energy savings in terms of energy needed for heating the gym for a given number of days. Alternatively, it can explain the possible monetary savings in terms of commonly purchased goods (e.g. average number of pizzas that could have been purchased by saving on energy consumption).

The *Social Platform* component is a digital and social platform which will be developed as a social network where users can share their feedbacks and their perceptions of indoor thermal comfort (e.g. too hot, too cold, or comfortable). Furthermore, it provides visibility of both technical and informative KPIs. The aim is ehnancing energy awareness and stimulate sustainable behaviors to optimize energy consumption.

The EDEN platform also includes the knowledge extraction block for discovering interesting associations among thermal energy consumptions, indoor climate conditions, meteorological conditions, and user perception of indoor thermal comfort in the form of association rules [1]. Association rules represent a powerful exploratory data mining approach able to discover interesting and hidden correlations in the data.

Finally, a subset of interesting and open data (e.g., KPI values) will be published in the *Smart Data Platform* to improve both individual and collective energy awareness. The Smart Data Platform exploited in EDEN is the Yucca Smart Data Net [18] developed by the Piedmont Region (Italy).

3. PLATFORM COMPONENTS

In this section we describe the main components of the proposed EDEN platform, which are currently under development.

3.1 Data platform

Remote measurements of energy consumption are collected by IREN [13], an Italian energy-provider company, by means of gateway boxes installed in monitored buildings. Each gateway includes a GPRS modem with an embedded programmable ARM CPU. An ad-hoc software has been developed to execute the following activities: sensor management, GPRS communication, remote software update, data collection scheduling, and collected data sending to a remote server.

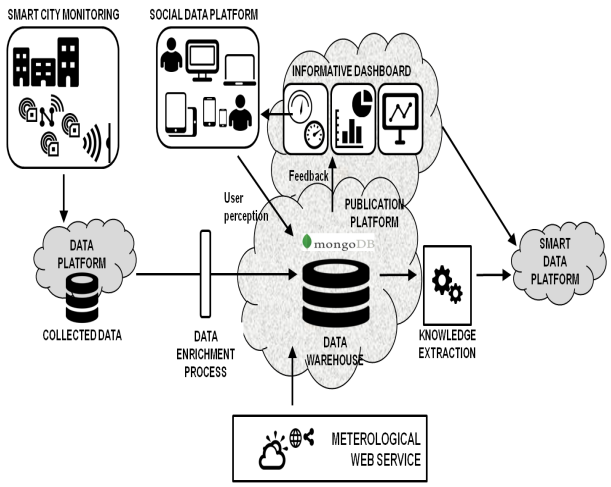


Figure 1: The EDEN system architecture.

Each gateway has in charge the management of all the sensors deployed in its building. Thermal energy is measured under different aspects, such as instantaneous power, cumulative energy consumption, water flow and corresponding temperatures. Furthermore, gateways also collect indoor temperature and the status of the heating system.

A cloud architecture is used for storing and processing all the monitored data. As of December 2014, there are about 4 thousands monitored buildings, each generating about 2,000 data frames per day. Thus, EDEN needs to manage a growing base of at least 8 million data frames per day. The gateways send the data frame to the cloud architecture, where a firewall first authenticates the data sender and then assigns each data frame to one of four dispatchers to guarantee the system reliability. Each dispatcher delivers the frame to a cluster of computers including different processing servers where data are stored in an HDFS distributed file system. The dispatcher is able to recognize if the process server has stored the frame correctly and in that case it sends the ACK to the gateway which can send the next data frame.

Each processing server elaborates the received data and stores the result in an Oracle database. The logical model of the database includes the following three tables: (i)The *Building* table contains the main features characterizing each building such as address and volume; (ii) the *Sensor* table stores the list of sensors located in each building and the main characteristics for each sensor (e.g., unit of measure, description, sensor type and model, etc.); (ii) the *History* table stores the collected measurements for all sensors. On average, every 5 minutes a data frame is received from each building. Then, corresponding data are stored in many records, with one record for each measurement value.

To efficiently perform the management of a large volume of collected data, different strategies have been adopted (e.g., data sharding, distributed map-reduces, and data replication).

3.2 Data integration and enrichment

Data collected through the smart meters are aggregated and enriched with additional contextual information acquired from external open data sources. More specifically, to analyze the *temporal distribution* of thermal energy consumption, the following time granularities are considered: day, month, 2-month, 3-month, 6-month time periods. Moreover, each day is classified as holiday or not, and the measurement time is aggregated into the corresponding *daily time slot* (morning, afternoon, evening, or night).

In Italy, heating systems are operated only from October 15th to April 14th, hence times periods outside this range were not considered. In addition, since the heating systems under monitoring within EDEN are operated at fixed time slots, each aggregation (morning, afternoon, evening) includes only the time slots when the system is actually on (e.g., morning from 6:00a.m. to 11:59a.m, afternoon from 12:00p.m to 6:59p.m., evening from 7:00p.m to 10:00p.m.).

To analyze the *spatial distribution* of thermal energy consumption, different space granularities are also considered beyond the building addresses. In addition, each *address* is mapped to the corresponding geographical *coordinates* (longitude and latitude degrees), *neighborhood* and *city district* including that neighborhood. While the address is an information recorded for the monitored building, the geographical coordinates and both the neighborhood and district names corresponding to the address are added as additional contextual features to the repository. We exploited the Google Maps APIs [11] for geocoding street addresses. Furthermore, topological information about neighborhood names and districts are integrated in the repository as well. The latter have been retrieved from [22]. Topologies are used to graphically analyze the most significant spatial trends in thermal energy consumption data and were encoded in GeoJSON, which is a standard format for encoding a variety of geographic data structures.

The above data were also enriched with meteorological information collected from the web. Specifically, historical meteorological data were taken from the Weather Underground web service, which gathers data from Personal Weather Stations (PWS) registered by users. For the city of Turin more than twenty PWS are distributed throughout the territory and about 4 of them are directly located inside the area considered in this study. The decision to use data from PWS is motivated by the fact that they reflect with high accuracy the real conditions registered in their neighborhood, as opposed to other services that provide estimated values with respect to a wider area. Although the measurement frequency can be easily set by the user for each PWS (and can vary over time), the average value for the ones we considered was about 5 minutes. Data were collected for the period going from October 2012 to April 2013. More specifically, each measurement includes the air temperature (expressed in degree Celsius), the relative humidity (percentage), the precipitation level (mm), the wind speed (km/h) and the sea level atmospheric pressure (hPa). The date and time of each measurement is also included.

3.3 Data warehouse

While the data collection from smart meters exploits an Oracle database, due to the fixed and constant nature of those measurements, enriched data is much more variable and heterogeneous, and its analysis requires a different technological solution. To this aim, enriched data are modeled into a document-oriented distributed data warehouse providing rich queries, full indexing, data replication, horizontal scalability and a flexible aggregation framework, including a distributed map-reduce engine. The current database em-

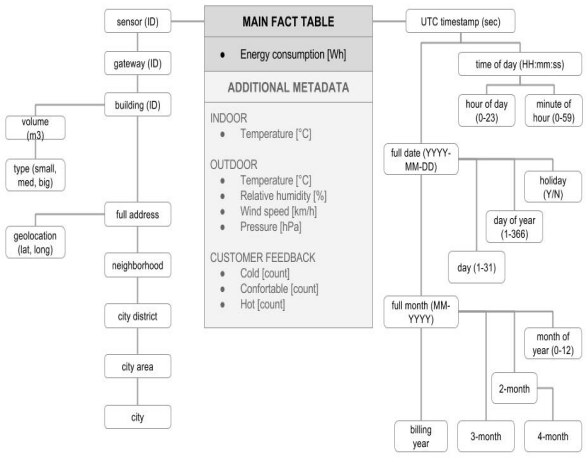


Figure 2: The EDEN data warehouse design.

powering EDEN analytics inside the Publication Platform is MongoDB [7], and to our purpose it is actually exploited as a data warehouse: periodically, sensor-collected data and social-platform data are enriched, integrated and loaded into a MongoDB collection.

Following best practices in data warehouse design, data are de-normalized and redundant information is added to each record (document) to speedup read performance by avoiding join operations (which are not sopported by MongoDB), and resulting in fast querying and KPI computation. The model design aims at providing a human-readable document format, hence the choice of long, self-descriptive field names, with sub-documents for each separate aspect of the record, from user feedbacks to geo-location, through smart meter measurements and other contextual informations. Such structured choice helps in coping with heterogeneity, but presents a main drawback in disk space usage: each field name is re-written within each document, together with all the redundant information that enrich the measurement. However, the low cost of disk space nowadays makes it an acceptable issue, also considering that no image or video data are currently included.

In Figure 2 the data warehouse conceptual model is presented: the fact table consists of a main measure, the energy consumption in a 5-minute time period, and some additional metadata coming from indoor sensors, outdoor PWSs, and the social data platform collecting customer feedbacks. Two hierarchies are defined: a time-related hierarchy and a place-related one. The former provides many different blends of time spans, from minutes to months and years. The latter starts from the physical sensors inside each monitored building and builds up to the whole city, with the building volume and the geolocation coordinates as related features included in the document.

Some metadata, in particular weather data and customer feedbacks, may require some additional pre-processing during the data loading phase because of different time spans: e.g., a customer feedback given at a certain point in time may be considered valid for a longer period than the specific 5-minute of a single data warehouse document, and weather data may be unavailable for a specific point in space. The solution adopted in EDEN supposes that customer feedback in terms of indoor environment comfort has a temporal validity of 30 minutes, which is distributed from 15 minutes before the feedback is provided and 15 minutes after. Hence, a customer reporting a very cold indoor environment at midnight, is associated with 5-minute documents from 23:45 (included) to 00:15 (excluded). Weather data associated with a specific building and address are computed as a distance-based weighted mean of the values provided by the three nearest PWSs. The weight is inversely proportional to the distance from the PWS to the building location, hence three equally distant PWSs would have the same weight in determining the outdoor values of a given building.

In the following, a sample MongoDB document from the designed data warehouse is provided. Subdocuments have been extensively used to group similar fields together. Some fields deem special attention:

- The customer feedback fields that identify too cold, too hot and comfortable indoor environments are the count of the collected feedbacks in the 30-minute time span as previously described.
- The customer comments are a list of text strings provided as status description on the social data platform; this allows us to exploit text mining techniques to associate keywords to measurement values, by building upon the text search features of MongoDB. This issues will be addressed as a future development of the current implementation.
- The billing period spans over two different years: October-November is the first 2-month (billing and operational) period and so the December-January 2-month period spans two calendar years, hence the choice to be verbose and use values such as '2-2014-2015'.

```
{
    _id: ObjectId(...),
    energy_consumption: 0.12,
    indoor: {
        temperature: 21.2
},
    outdoor: {
        temperature: 15.6,
        relative_humidity: 70.0,
        wind_speed: 5.0
    },
    feedback: {
        cold: 2,
        comfortable: 12,
        hot: 1,
        comments: ["nice sunny winter day",...]
    },
    place: {
        sensor:  {id: 123456, model:"..."},
        gateway: {id: 234567, model:"..."},
        building: {
            id: 345678,
            volume: 1234,
            type: "med"
        },
        address: {
            full: "corso Castelfidardo 39, 10129, ...",
            street_name: "Castelfidardo",
            street_number: "39",
```

```
            coordinates: [7.6600778, 45.0632518],
            ...
        },
        neighborhood: "Crocetta",
        city_district: "Circoscrizione I",
        city: "Torino"
    },
    time: {
        UTC_timestamp: 1419266446.0,
        day: {
            time: "16:40:46",
            minute: 40,
            hour: 16,
            slot: "afternoon"
        },
        date: {
            full: "2014-12-22",
            day: 22,
            day_of_year: 356,
            holiday: "N"
        }
        month: "12-2014",
        month_of_year: 12,
        2month: "2-2014-2015",
        3month: "1-2014-2015",
        4month: "1-2014-2015",
        billing_year: "2014-2015"
    }
}
```

Finally, the data model design addresses horizontal scalability and replication choices.

Horizontal scalability is obtained by exploiting data sharding, i.e., storing documents across multiple distributed machines by dividing the collection and distributing its data over multiple servers, or shards. As the size of the data increases, EDEN only needs to add more machines to scale and support the demand of a higher number of read and write operations. Each shard processes fewer operations as the cluster grows, and the amount of data that each server needs to store is reduced.

MongoDB provides automatic sharding and the key design choice is the attribute whose values partition the collection documents, i.e., the shard key. In EDEN the sharding is performed using a hash-based partitioning on the value of the building ID field. The shard key choice is motivated by KPIs that are typically computed by grouping measurements per building, and the number of buildings grows with the expansion of the EDEN framework, hence it is a natural scaling indicator. Hash-based partitioning has been chosen over the range-based partitioning approach to ensure that data are evenly distributed across the machines in the cluster, since no range queries are performed on the building ID.

Replication is obtained by exploiting MongoDB replica sets to provide redundancy and high availability. With multiple copies of data on different servers, replication avoids data loss from a single server failure. Currently, in EDEN each replica set consists of a primary server, a secondary server and an arbiter. All writes go to the primary server, while the secondary server can be exploited to increase the read capacity at the cost of possible inconsistency. However, this is not an issue in EDEN since KPIs for the dashboards can wait to be updated after the secondary has caught up the updates from the primary, which usually happens within seconds.

3.4 KPIs definition

The EDEN system performs the KPI analysis tailored to different users to gain insights on the integrated data. In Business Intelligence, the analysis of Key Performance Indicators (KPIs) is an established methodology [15]. KPIs help organizations define and measure progress towards organizational goals by monitoring the most significant achievements. In our context, KPIs are quantitative indicators of thermal energy consumptions. To apply KPI analyses to data coming from a real scenario, we defined *technical KPIs* and *informative and user-friendly KPIs*. The aim of KPI generation is to produce useful feedbacks to enhance energy awareness for different types of users. We identified four different operational roles representing users of the EDEN system: (i) the *Energy Manager* is responsible for the energy services provided. He/She needs to access summary and high-level information in order to grasp the overall picture of the energy situation of the city district under observation. He/She requires dashboards showing KPIs at a higher level of granularity (e.g., city district). (ii) The *Energy Analyst* is an expert in energy consumption. He/She is interested in analyzing the complete streams of collected data to observe and understand the observed phenomenon, analyze the different components and identify possible causes. He/She needs to inspect a significant volume of data to understand the anomaly. (iii) The *Consumer* represents the building condos administrators or the public administration (as in the case of public schools), whose interest is to assess the efficiency of the heating system, as well as to get a feeling of virtuous behaviors that should/could lead to energy savings while maintaining the desired level of indoor confort. He/She only needs to visualize a few indicators, possibly presented in a clear and intuitive way. (iv) The *Users living in the building* are interested in mantaining indoor wellness and understand how their behaviors affect energy consumption and they can achieve a significant reduction of their energy expenditure. Presented data should be informative and, at the same time, easy to understand.

For users living in the building we define two user-friendly KPIs that measure virtuous behaviors (i.e. energy savings) in terms of (i) energy needed for heating the given building for a given number of days, or (ii) kilograms of bread or number of pizzas that can be purchased with the savings.

The technical KPIs aims at evaluating the energy consumption at different levels: from the single building to the neighborhood, and from hours and days to months. In EDEN four technical KPIs have been identified.

- Building KPI. Average energy consumption indicator of the building per unit of volume, i.e., total energy consumption of the building divided by the building total volume. This KPI can be also normalized according to the degree days and to the known indoor temperature.
- Neighborhood KPI. Average energy consumption indicator of the buildings in the same neighborhood per unit of volume.
- Building-type KPI. Average energy consumption indicator of the buildings of the same type and in the same neighborhood per unit of volume.

- Climate KPI. Average energy consumption indicator of the buildings of the same type and in the same neighborhood per unit of volume, considering only energy consumption during specific outdoor conditions (temperature range).

These KPIs are computed on different time scales, in particular: hourly, for each daily time slots, daily, monthly, and on N-month periods.

Rich queries, indexing and map-reduces are the data warehouse features exploited to compute KPIs. Specifically, fields frequently used by KPI queries such as building IDs are indexed, and map reduces are exploited to perform KPI computation. Let consider a simple KPI such as the first of the list, and for the sake of simplicity, suppose the temporal scope and normalizations are removed (their implications will be discussed later). The equivalent SQL query to extract the Building KPI would be as follows.

```
select sum(energy_consumption)/building_volume
from fact_table, dimension_table1, ...
where <join fact and dimension tables>
group by building_id, building_volume
```

In EDEN such KPI is computed by exploiting map, reduce and finalize functions of MongoDB, as follows. The map function determines the key and value pairs emitted by each processed document: the key is similar to the group by SQL clause, and in this case it corresponds to the building ID, whereas the value is a more complex object, since to compute an average we need to carry over both operands, the consumption sum and the building volume. Hence, we put these two values into the value object returned (emitted) by the map function.

```
function() {
    key = this.place.building.id;
    value = {
        ec: this.energy_consumption,
        vol: this.place.building.volume
    };
    emit(key, value);
}
```

The reduce function receives a list of values from the map functions having the same key, hence we have a list of objects containing the energy consumption (*ec*) and the building volume (*vol*), and we need to sum all the *ec* values of the list. The building volume is the same for all values, since they refer to the same building (the building id is the map reduce key).

```
function(key, values) {
    reduced_value = {
        ec: 0,
        vol: values[0].vol,
    };
    for (var i=0; i<values.length; i++) {
        reduced_value.ec += values[i].ec;
    }
    return reduced_value;
}
```

After the reduce phase we have a list of value objects, one for each building id (key), containing the total energy consumption and the building volume. The finalize function adds to each object in this list the average value, which is the final result and corresponds to the desired KPI.

```
function (key, value) {
    value.ec_vol = value.ec/value.vol;
    return value;
};
```

The provided example is computed over the whole collection and return total cumulative results since the beginning of the data collection. The temporal scope can be introduced by exploiting two approaches: (i) a specific query filtering undesired time periods can be passed to the map reduce MongoDB command, thus limiting the computation to a specific time span, or (ii) a more complex key can be used involving compound building ID and time periods. The latter is particularly useful to save pre-aggregated results in a collection similarly to materialized views. For instance a simple compound map-reduce key such as the concatenation of the building ID and the date (YYYY-MM-DD) of the measurement would automatically provide day-level aggregations and would require a small change in the map function only. In EDEN then, monthly KPIs are computed directly by querying the daily KPIs collections, hence building a tree of map-reduces that are fed by lower-level lesser-aggregated results and feed higher-level map-reduces in the tree.

Current advantages of the map-reduce KPI approach include a natively distributed computation, that allows horizontal scaling and load balancing among the nodes of the MongoDB cluster. We are currently analyzing further improvements on the EDEN KPI computation framework that include incremental map-reduces, which are an obvious approach due to the nature of the data loading, and the exploitation of the MongoDB aggregation framework. Furthermore, the ability to add new fields to the documents allow us to easily implement new KPI computations as they are required, even if the database does not natively support join operations. Indeed, the actual join is performed as a preprocessing step during the data enriching phase.

Finally, MongoDB also provides native support for geospatial querying, that is exploited in EDEN to compute KPIs involving the neighborhood besides the administrative boundaries. For instance, to query all the measurements associated with buildings within a given distance from a specific point in space, the following snippet of code can be added to an existent query.

```
{
   'address.coordinates': {
      $geoWithin: {
        $center: [ [7.6600778, 45.0632518], 0.01]
      }
   }
}
```

This limits the results to the measurements in a radius of approximately 0.01 degrees (roughly 1 km) from the point at the given longitude and latitude coordinates.

3.5 Smart data platform

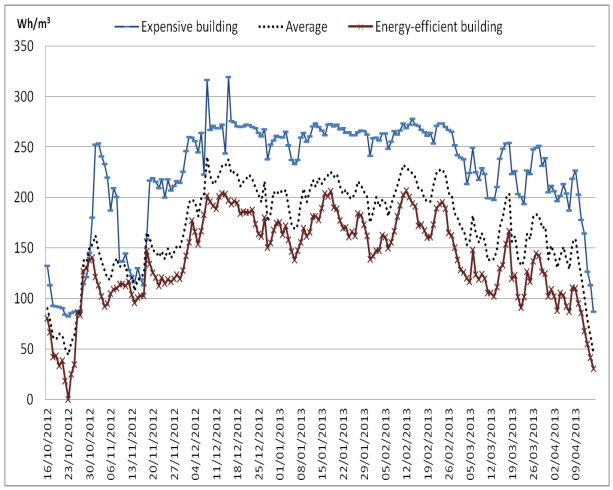


Figure 3: Residential buildings: Daily energy consumption per unit of volume (Wh/m^3) .

The EDEN system will publish a subset of collected data and results of the analysis in the Yucca Smart Data Platform (SDP) [18]. Specifically, a portion of the data showed to users through the informative dashboard, a subset of user's feedbacks and indoor thermal comfort perception data, and interesting knowledge items extracted from the enriched data collection. The Yucca SDP is a Big Data store developed and maintained by CSI Piemonte [8]. Based on the Open Data paradigm, it gives individuals and organizations the opportunity to publicly share their data under a license that allows anyone to freely use them. It enables the interconnection of geographically distributed applications, social networks, objects and systems. The Yucca SDP supports different protocols to receive and send data, such as HTTP, MQTT, RTSP, WebSocket and OData REST APIs. It also provides some basic functionalities of data enrichment, aggregation, filtering, pattern matching and windowing.

4. EXPERIMENTAL RESULTS

We performed a preliminary analysis of energy consumption on a real dataset using the EDEN platform. We considered 2 school buildings and 6 residential buildings, all located in two neighboring districts in Turin, within a circular area of 1 km of radius. Values were measured roughly every 5 minutes. The full time period depends on the availability of measurements for each building. For the residential buildings, measures are available from 2012 to 2014. To consider a complete winter period we analyzed the period from October 15^{th}, 2012 to April 14^{th}, 2013. For the first school (named school A), instead, the time period is from November 28^{th}, 2013 to April 30^{th}, 2014. For the second school (named school B), it is from October 1^{st}, 2012 to March 14^{th}, 2013.

Firstly, the daily energy consumption per unit of volume (Wh/m^3) has been computed for each residential building, together with the daily average consumption among all buildings. Figure 3 shows the average consumption profile, and the profiles of an expensive building and an efficient one.

Since the time periods available for the two school buildings are different, also in duration, a further processing has been performed to compare their energy efficiency: the consumption has been normalized with respect to the total degree days measured for the same time length. This measure represents the different external temperatures that influence the daily energetic demand for heating. We computed the total degree days as the sum of all the positive differences between a reference indoor temperature (i.e., 20 °Celsius) and the average daily temperature taken from the ARPA weather archives [3]. Results are reported in Table 1. As shown in Table 1, the daily energy consumption in school B is much greater than in school A, with a difference of about 254 kWh. However, a higher value of average degree days can also be observed (12.37 °C of school B versus 10.97 °C of school A). The last row in Table 1 shows the energy consumption per unit of volume divided by the total degree days. The total consumption normalized with respect to the degree days is still higher, but the difference is much smaller. In fact, if we had 1690 degree days for school B (like in school A), the total energy consumption per unit of volume unit would have been only 31.04 Wh/(m^3×°C) × 1690 °C= 52458 Wh/m^3, rather than 63336 Wh/m^3, which is much closer to the 50373 Wh/m^3 of school A.

5. CONCLUSIONS AND FUTURE WORKS

This paper presented a preliminary implementation of the EDEN platform to enhance energy awareness. As of December 2014, IREN has installed thousands of thermal smart meters in buildings in Turin, a major Italian city. EDEN components and design choices that led to the Data Platform and the Publication Platform have been discussed, with the aim of efficiently collect and analyze data on energy consumption. The Data Platform collects all the monitored data, while the Publication Platform includes pre-processed data enriched with spatial and temporal information at different abstraction levels, as well as meteorological data available in open source datasets. We also designed and implemented different technical and user-friendly KPIs to provide informative dashboards targeting different users.

We are currently implementing an ad-hoc social platform where users are proactively engaged in the act of generating data related to their perception of thermal comfort, as well as useful feedbacks on thermal energy consumption of the buildings where they live or work. The social platform will also show to users both technical and user-friendly KPIs on energy consumptions (savings or overconsumption) in an informative fashion.

Since the collected data easily scale towards very large datasets, the problem of discovering interesting and hidden correlations for these huge data collections becomes challenging. We are currently designing an innovative scalable algorithmtailored to enriched data managed by EDEN to efficiently perform the association rule mining on a huge energy consumption dataset [5, 4].

6. ACKNOWLEDGMENTS

The research leading to these results has partially received funding from the Piedmont Region under the POR FESR 2007/2013 n. 281-79 (EDEN Project).

The authors wish to thank colleagues and other partners involved in the EDEN project (i.e., CSP Innovazione nel ICT, COMMITWORLD S.r.l., Capetti Elettronica s.r.l., Consorzio TOP-IX, Experientia, Sisvel Technology s.r.l.) for their advices and fruitful discussions.

	School A	School B
Volume [m^3]	4480	4480
Time period	11/28/2013 – 04/30/2014	10/01/2012 – 14/03/2013
Total energy consumption per unit of volume [Whm^3]	50,373	63,336
Daily energy consumption [Wh]	1,465,390	1,719,658
Daily consumption per unit of volume [Wh/m^3]	327.10	383.85
Average degree days [°C]	10.97	12.37
Total degree days (in the given time period) [°C]	1690	2040.4
Total normalized consumption [$Wh/(m^3 \times °C)$]	29.81	31.04

Table 1: School buildings: Energy consumption normalized per unit of volume and degree days

7. REFERENCES

[1] R. Agrawal, T. Imielinski, and Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD 1993*, pages 207–216, 1993.

[2] O. Ardakanian, N. Koochakzadeh, R. P. Singh, L. Golab, and S. Keshav. Computing electricity consumption profiles from household smart meter data. In *EDBT/ICDT Workshops'14*, pages 140–147, 2014.

[3] ARPA. Piedmont Region. *Regional Agency for the Protection of the Environment. Available at http://www.arpa.piemonte.it/english-version Last access: December 2014.*

[4] E. Baralis, T. Cerquitelli, S. Chiusano, and A. Grand. P-mine: Parallel itemset mining on large datasets. In *Workshops Proceedings of the 29th IEEE International Conference on Data Engineering*, pages 266–271, 2013.

[5] E. Baralis, T. Cerquitelli, S. Chiusano, and A. Grand. Scalable out-of-core itemset mining. *Inf. Sci.*, 293:146–162, 2015.

[6] E. Baralis, T. Cerquitelli, and V. D'Elia. Modeling a sensor network by means of clustering. In *18th International Workshop on Database and Expert Systems Applications*, pages 177–181, 2007.

[7] K. Chodorow and M. Dirolf. *MongoDB: the definitive guide.* O'Reilly Media, 2010.

[8] CSI website. *http://www.csipiemonte.it/. Last access on December 2014.*

[9] S. Depuru, L. Wang, V. Devabhaktuni, and P. Nelapati. A hybrid neural network model and encoding technique for enhanced classification of energy consumption data. In *Power and Energy Society General Meeting*, pages 1–8, 2011.

[10] C. Filippín and S. F. Larsen. Analysis of energy consumption patterns in multi-family housing in a moderate cold climate. *Energy Policy*, 37(9):3489 – 3501, 2009.

[11] Google Maps. *Available at http:/maps.google.it Last access: December 2014.*

[12] IEA. Energy efficiency indicators. 2014.

[13] IREN website. *http://www.gruppoiren.it/index.asp. Last access on December 2014.*

[14] S. R. Iyer, V. Sarangan, A. Vasan, and A. Sivasubramaniam. Watts in the basket?: Energy analysis of a retail chain. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, pages 4:1–4:8. ACM, 2013.

[15] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling.* John Wiley & Sons, Inc., 2nd edition, 2002.

[16] D. Kriksciuniene, T. Pitner, A. Kucera, and V. Sakalauskas. Sensor network analytics for intelligent facility management. In *Proceedings of the 6th International Conference on Intelligent Interactive Multimedia Systems and Services*, pages 212 –221, Amsterdam, 2013. IOS Press.

[17] D. Kriksciuniene, T. Pitner, A. Kucera, and V. Sakalauskas. Data analysis in the intelligent building environment. *International Journal of Computer Science and Applications*, Volume 11, 2014.

[18] Piedmont Region. ITALY. *SMARTDATANET. Available at http://www.smartdatanet.it/ Last access: December 2014.*

[19] L. Pitt, P. Green, and B. Lennox. A sensor network for predicting and maintaining occupant comfort. In *Environmental Energy and Structural Monitoring Systems*, pages 1–6, 2013.

[20] M. Santamouris, G. Mihalakakou, P. Patargias, N. Gaitani, K. Sfakianaki, M. Papaglastra, C. Pavlou, P. Doukas, E. Primikiri, V. Geros, M. Assimakopoulos, R. Mitoula, and S. Zerefos. Using intelligent clustering techniques to classify the energy performance of school buildings. *Energy and Buildings*, 39(1):45 – 51, 2007.

[21] J. E. Seem. Using intelligent data analysis to detect abnormal energy consumption in buildings. *Energy and Buildings*, 39(1):52 – 58, 2007.

[22] Turin GeoPortal. *Available at http://www.comune.torino.it/geoportale/ Last access: December 2014.*

[23] J. van der Veen, B. van der Waaij, and R. Meijer. Sensor data storage performance: SQL or NoSQL, physical or virtual. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 431–438, June 2012.

[24] D. Wijayasekara, O. Linda, M. Manic, and C. Rieger. Mining building energy management system data using fuzzy anomaly detection and linguistic descriptions. *Industrial Informatics, IEEE Transactions on*, 10(3):1829–1840, Aug 2014.

[25] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. Urban computing: Concepts, methodologies, and applications. *ACM Trans. Intell. Syst. Technol.*, 5(3):38:1–38:55, Sept. 2014.

Hybrid Multidimensional Design for Heterogeneous Data Supported by Ontological Analysis: an Application Case in the Brazilian Electric System Operation

João Moreira[1], Kelli Cordeiro[2], Maria Luiza M. Campos[2], Marcos Borges[2]

[1] University of Twente, Services / Cyber-security / Safety (SCS) Group, Netherlands
[2] Federal University of Rio de Janeiro (UFRJ), Knowledge Engineering (GRECO) Group, Brazil
[1] j.luizrebelomoreira@utwente.nl
[2] {kelli,mluiza,mborges}@ppgi.ufrj.br

ABSTRACT

An issue in operating a national electric system is how the corporate image of an Independent System Operator (ISO) can be impacted by disturbances in the system and their related news publications from specialized press. To deal with it, a solution was developed in the context of the Brazilian Electric System National Operator (ONS): an analytical system for disturbances analysis integrating both structured and unstructured data sources. It considers both the daily news publications about the electric sector from ONS clippings website and the details of operational disturbances from the company data marts. We introduce here an adaptation of the hybrid multidimensional (MD) design method, considering heterogeneous data sources during business analysis and design phases. Most important, we illustrate how ontological analysis can enhance the semantic expressiveness of the MD modeling activity through a semi-automatic derivation process. The analytical potential is evidenced by a real scenario case study.

Keywords

Multidimensional design, unstructured, ontology, disturbances.

1. INTRODUCTION

Treating events of the electric sector through the support of database (DB) systems is a critical activity in the operation of multi-owned energy transmission, such as collecting and analyzing disturbances occurrences. Usually, an ISO company is responsible for this activity. In Brazil, ONS is in charge of monitoring the national electric system. A Decision Support System (DSS) based on Business Intelligence / Data Warehousing (BI/DW) architecture [7], coined Disturbances BI, uses structured data for disturbances analysis. Disturbances are most noticed by the population when blackouts occur. Because of their negative consequences, Brazilian press often publishes news on the subject, citing ONS, which may influence its corporate image. News publications regarding the electric sector are collected and made available daily at the clippings website. However, there is still the need of an analytical environment to support decision makers on analyzing the impact of disturbances on ONS institutional image. To achieve this goal, a solution to integrate structured data sources to unstructured data from the clippings in a BI/DW architecture has been a main requirement.

The problem addressed in this paper is the lack of a methodology for BI/DW solutions that considers both types of data sources. Indeed, there are a number of systems and solutions that extract information from text and integrate with existing DBs, but it is still missing a well-defined process to determine how this type of application can be used in a corporative context. We propose an approach for adapting Moss's BI/DW lifecycle methodology [11] so to consider heterogeneous data, addressing semantic problems during the MD design, such as ambiguity and low semantic expressiveness. In this paper a systematic approach is described, extending the hybrid MD design method by considering reverse engineering from text corpora during the source-driven activity. Furthermore, we guide how ontological analysis can be applied as a base for a semi-automatic process for MD schemas derivation, from a well-founded domain ontology that represents information in the data sources.

We also include the application of our approach in the case study of disturbances and news publications joint analysis for ONS corporate image. In the analysis-driven design activity we consulted ONS official glossary, domain engineers and the Common Information Model (CIM). In parallel, during the source-driven design activity, the transactional master DB of ONS and the Disturbances BI solution played the role of structured data sources. Corpora of news publications from clippings website were collected and analyzed by the DW designer as an unstructured data source. Afterwards, the disturbance domain ontology was built considering the scope of the original schemas. It was developed using ontological analysis based on a foundational ontology [4], a high-level category system for a solid grounding of conceptual modeling. The domain ontology had its semantic enriched during the verification and validation activity, where conceptual assertions were made to increase the model quality. Then, a semi-automatic process for MD structures derivation supports the designer in delivering the final MD schema for temporal analysis. The DB design and data cube construction phases were performed so joint analysis examples over the final data cube are presented through reports in an OLAP tool. This paper presents the continuation of the approach introduced in [10], covering the adaptation to consider heterogeneous data in the MD design methodology, supported by the architecture we introduced in [9]. Moreover, the study case is detailed and the ontological approach is depicted.

2. EVENTS IN ELECTRIC SYSTEMS

Reliable and sustainable electric systems depend on the ability of monitoring and responding, or even predicting, occurrences in the electric sector. The treatment of events in the transmission grid is a crucial activity, like responding to the shutdown of transmission lines or other equipment, i.e. electrical disturbances. Such treatment is made possible by solutions that handle large amount of data, providing high quality information for decision makers in adequate time. BI/DW architecture is a consolidated and usual way to deliver information originated in structured data sources.

2.1 BI/DW Solution for Disturbances Analysis

ONS is a non-profit ISO, unique in Brazil, performing its duties under the supervision and regulation of the national electric energy agency. Its mission is to operate the Integrated National System (SIN), a large hydrothermal system responsible for 97% of the Brazilian electricity supply. It has a strong predominance of hydroelectric plants with multiple owners and their facilities and equipment, such as power plants, transmission lines and power transformers. Equipment in SIN is subject to faults and failures of various natures, causing forced shutdowns of one or more devices. This can interrupt the power supply to consumers depending on the resulting load cut level. These events are known as electric disturbances and may be caused by atmospheric electric discharges, floods, fires or even human failures. ONS official glossary defines an electric disturbance as "an occurrence in SIN characterized by forced shutdown of one or more of its components, which cause any of the following consequences: loss of load, shutdown of the system components, equipment damage or violation of operating limits."

The processes to fulfill the coordination and the control of SIN operation are based on technical procedures, rules and criteria defined in normative documents. Information systems were developed by ONS, e.g. Disturbances Integrated System (SIPER), to support the registration of disturbances as abnormalities, undesirable events or unsatisfactory performance. They are integrated through ONS master DB, which stores the core transactional data from the electric system. Analytical processes of disturbances are supported by a BI/DW solution, coined Disturbances BI. It consolidates data from transactional systems and a historical DB. The integration is made through a conventional ETL process over structured data, being available in a disturbances data cube. The users can navigate and generate reports over the data cube through OLAP tools.

2.2 Impact of Disturbances on ONS Image

The corporate image is the way the organization is perceived by society, tending to be classified as positive or negative, varying in intensity and depending on variables such as opportunities, threats and competences. ONS provides a daily summary of news related to the electricity sector in its intranet home page, the clippings website. Its main purpose is to provide to ONS collaborators news publications from the specialized press, quoting the organization when it is mentioned. The result of a disturbance in the system can lead to power supply cuts, popularly known as blackouts. This situation has a direct relation to the load cut level measure of disturbances fact in Disturbances BI. The negative consequences of a blackout to the population are numerous, generating large financial losses in all sectors of the economy. The press gives great focus to the subject, often citing ONS when such situation occurs, which may influence its corporate image. Among ONS main concerns in the electric security domain, the analyses of faults caused by disturbances in the system and their impact on users' lives, reflected in the media, is much relevant for decisions related to the corporate marketing. Current information systems present the information of disturbances and news about SIN independently. Hard manual work is necessary for a joint analysis over large amounts of historic data, often making it impossible to reach the desired results. Therefore, an analytical information system for joint exploration of disturbances and their impact in news can address this need. Existing DSS solutions were mapped: Disturbance BI and clippings website. They represent operational data sources captured in business case assessment.

3. APPROACH PROPOSAL

To address the methodological support needed for a disturbances and clippings integration solution we propose adaptations of Moss's methodology [11] to consider heterogeneous data. This BI/DW lifecycle resembles Kimball's [7] and Malinowski's [8] approaches, but it adds a metadata repository. Even being called a lifecycle, it lacks operations and decommissioning phases after deployment. However, it presents a balanced approach, considering complexity and practice. Each activity is part of a specific phase, as depicted in Figure 1 (left). Business analysis and design phases are considered the most important activities because they guide the BI/DW solution development. The efficacy of MD modeling is directly related to future costs in maintaining the BI/DW solution. It can be increased by avoiding conceptual mistakes through the application of a common understanding formalization [16]. Here we focus in the MD design activity as illustrated in Figure 1 (right). Our method is based on [8], but we consider an ontological approach.

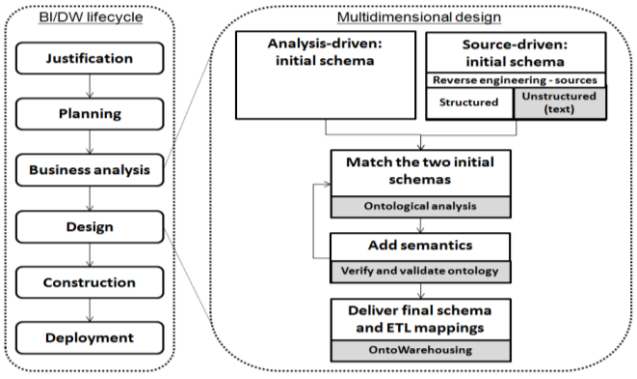


Figure 1. Hybrid multidimensional design activity adapted.

3.1 Hybrid Design for Heterogeneous Data

In our approach we consider unstructured data sources as text and ontological analysis to increase the semantic expressiveness of the MD modeling activity. The semantic expressiveness (or semantic power) is the quality of how precise a model is on representing the reality [4]. It considers both supply-driven and analysis-driven strategies running in parallel for deriving the initial schemas. In the analysis-driven schema derivation, the designer can use the domain knowledge from domain experts, existing procedures, glossaries, taxonomies or other terminological standards. In the supply-driven one, both structured and unstructured data sources can be analyzed. Then, the matching of the schemas sketches, i.e. their merging, is supported by ontological analysis [4], representing the business concepts in a domain ontology, categorized by top-level categories. Then, the domain ontology is verified and validated, increasing its quality in a cyclical way. Afterwards, rules defined as in a prior work, coined OntoWarehousing [10], can be applied to derive possible MD structures, used by DW designers in the final

definitions of MD schemas. Adaptations of the activities to cope with unstructured data and ontological engineering are described below.

3.1.1 Analysis-Driven Design

Each domain concept should be correctly named, uniquely identified and validated by business people who will access the data. Therefore, ontological analysis can be applied to support common understanding. A domain ontology can be sketched based on interviews with the main stakeholders and business official vocabularies, such as glossaries and standards. The ontology should be independent of technologies, not being influenced by any type of software or hardware. Current business processes should be understood, so the behavior of the concepts is mapped to the ontology, e.g. their creation or modification.

3.1.2 Source-Driven Design

In our approach we divided the reverse engineering in two main activities: from transactional DBs (usual), as structured sources, and, from textual sources. The result artifact from this activity is a sketch of the domain ontology from the point of view of the data sources. In both reengineering processes, making annotations about the origins of the data is crucial for the ETL design. Reverse engineering from structured data sources is widely addressed by supply-driven related works, e.g. AMDO [15]. It checks functional dependencies among tables by verifying relationships, cardinalities and constraints. Then, MD structures can be derived automatically based on a set of heuristics. It can capture important business rules and policies that may not be gathered during interview sessions. Some CASE tools implement this capability.

Reverse engineering from text generates representations of entities and their relations from unstructured data sources. This can be made manually or automatically. In both cases a set of text corpora is selected with support of business experts. Then, its content is analyzed. Automatic approaches consider Natural Language Processing (NLP) and Information Retrieval (IR) techniques applied to the corpora, resulting in suggestions of models. Entity and relations recognition techniques play an important role on ontology generation. Tools that implement these techniques are based on lexical methods, such as orthographic correction, stop word elimination, tokening, synonymous resolution, stemming, morphological classification and some type of semantic categorization from business terms. In our approach we do not choose one specific technique or tool. Instead, we guide the designer to first check the existing NLP and IR solutions.

3.1.3 Domain Ontology for Initial Schemas

The inputs for this phase are the model sketches from prior activities. Common concepts found in these representations should be matched or associated, by annotating their data structures origins. This information will be necessary to build the linkages between the structured and the unstructured universes for designing the ETL process. After matching all entities, annotating their origins, the consolidated domain ontology should be built. We propose the use of the OntoWarehousing [10] ontological approach to increase the semantic expressiveness of the MD design. It presents a systematic and semi-automatic derivation process to suggest MD structures from categories of a foundational ontology, a high-level category system that represents concepts such as endurants (things that are in time) and perdurants (events or things that happen in time) [3] – refer to section 5 for a more detailed explanation. The output of this activity is the consolidated well-founded domain ontology.

3.1.4 Add Semantics: Verify and Validate

In this phase the designer analyzes the foundational constructs and checks if the entities and relations from the model are semantically consistent, also verifying business rules violations. Domain experts can support the quality improvement of the domain ontology, ensuring that the model is semantically correct, covering the main entities involved in the business requirements, avoiding ambiguity. This is a cyclical process: when the designer finds an inconsistency, he fixes the model and validates it again. Verifying and validating (V&V) ontologies with many concepts can be unfeasible for humans because of their size and complexity. Thus, a common practice is to choose ontology sub-domains, validate each separately and merge them. The resulting artifact is the valid well-founded domain ontology.

3.1.5 Deliver Final Schema: Derivation Process

The final MD schemas are designed based on the well-founded domain ontology and other existing MD schemas. In common methodologies [7,8] this task depends purely on informal guidelines for MD designer decisions. It depends on tacit knowledge, being error prone. In OntoWarehousing [10], we defined a set of mapping rules to derive possible MD structures from the well-founded domain ontology. The MD designer can use this method to increase its assertiveness. The mechanism to derive MD concepts begins by reading the domain ontology and looking for the foundational ontology categories. Once they are found, it executes the mapping rules and presents to the modeler the possible MD structures inferred. Thereafter, the derived MD concepts are conformed to other MD schemas, providing new analytical possibilities. At last, the designer defines the final MD schemas with data sources annotations as comments in natural language to serve as specification for the ETL processes. Other ontological approaches for MD design can be used in parallel, combining the final MD concepts produced such as in [16].

4. APPLICATION CASE

To handle disturbances and clipping s integration we have applied our approach in the construction of a BI/DW solution. Business analysis, design and construction phases were performed and some analysis examples over the final data cube could be made. To support the BI/DW lifecycle, Enterprise Architect (EA: http://www.sparxsystems.com.au/) CASE tool was chosen, which provides a full-set of capabilities for requirements formalization, conceptual models design and behavioral aspects representations. In addition, OntoUML Lightweight Editor (OLED: https://code.google.com/p/ontouml-lightweight-editor/) software and its plug-in to EA supported the V&V process.

4.1 Business analysis

4.1.1 Analysis-Driven Design

The analysis driven design was supported by ONS domain experts, the official glossary, and the CIM IEC 61970 (http://www.iec.ch/smartgrid/standards/). ONS official glossary contains the definitions of the main terms used in the electric sector. It serves as main business concepts source for common understanding among ONS and other agents. It helped in asserting the initial domain representations. When a specific term was not encountered in the glossary or there was ambiguity, the domain experts were consulted, mostly power systems engineers. They asserted specific rules, such as the part-whole relation between a disturbance and a forced shutdown, where a forced shutdown is part of one unique disturbance and it is existentially dependent of the disturbance. The CIM IEC 61970 is an international standard built by the electric power industry and it was adopted to support

information systems interoperation and common concepts agreement. Particularly, the main part of this standard was chosen, the IEC-61970 for energy management. It brings the representations of core concepts of electric power transmission and distribution domain, such as equipment (e.g. power transformer) and its sets (equipment containers) as power system resources. As a practical advantage, this standard is available and extensible in the EA tool as a UML class structural package.

4.1.2 Source-Driven Design

The involved data sources were listed as: the SIPER cut-off of ONS master DB, an entities mapping between the master DB and CIM models, Disturbances BI and clippings website (news publications). The physical data model was used to check tables, attributes, relationship integrities and constraints that implement the domain behavior. This type of information was included when representing the company concept. The entities mapping specification between the master DB and CIM describes the equivalence between the data structures from ONS master DB and the classes and relations of CIM meta-model. This document was previously built and used for the development of an ETL process, which extracts data from ONS master DB, transforms and load it in a CIM file representation. Thus, the domain could be designed in English terms, reusing the existing knowledge. The available ETL processes of the Disturbances BI were analyzed. We checked the ETL process to load the fact disturbance, which has associations to dimensions such as owner agent, source equipment, cause, begin/end time, among others. The clippings website was checked and the news publications sub-domain modeled, as textual information source. At first, a web crawler was built to download news publications from January 2011 to March 2013. A textual ETL process from a prior work [9] was applied in these corpora selected. It resulted on a data repository, named terminological DB, which stores the terms and their lexical and semantic categories, supported by IR.

4.1.3 Domain Ontology for Initial Schemas

The domain ontology was built in the EA tool supported by OLED plug-in. As starting point, the SIN domain package was composed by five sub-domains: companies, facilities, equipment and geographical region (structural aspects); and disturbances and news publications (dynamic aspects). The most important relation to link disturbances and news publications was defined through the temporal formal relation "before" at the conceptual level, meaning that a disturbance that occurs before a news publication can be somehow related to it. Disturbance and news publication are both classified as complex events, inheriting a series of properties, such as their beginning and ending time points, composition by other events, etc. There is a practical implication in this representation that was found during the construction phase regarding how long a disturbance occurred before a publication about it. For instance, if a disturbance occurred in 2010 and some news are published in 2013, if even this relation respects the "before" relation, it is most unlikely that they are related. Therefore, we stated a threshold of ten days based in prior experimentation [9]. Initial analysis evidenced the increase in publications after a severe disturbance and a decrease on subsequent days, reaching the publications average in ten days.

4.1.4 Add Semantics: Verify and Validate

This activity was supported by OLED software. After the first time designing the main concepts in the domain ontology (in EA tool), we exported it as an XMI file and imported into the OLED tool. During the import process, the tool provides a selection of classes and relations that the user would like to validate. Cut offs were made for V&V each part of the domain ontology. We could validate the domain ontology by examples and counterexamples, simulating instances of classes and their relations through the visual capability of Alloy analyzer provided in OLED. Moreover, OCL check statements were written as business rules representations. The result of this activity was the well-founded domain ontology considering disturbances and news publications.

4.1.5 Deliver Final Schema: Derivation Process

The final MD schema was designed based on the well-founded domain ontology. The OntoWarehousing approach was applied to discover MD structures, implemented through a prototype, which was executed in the domain ontology (refer to [9]). The MD schema to analyze the "before" relation could be designed by the derived MD structures from the proposed rules and conciliated with the existing disturbances MD schema. Moreover, from the axiomatization of the temporal operator "before", the constraint for the WHERE clause of the SQL to load the fact at the end of the ETL process was derived. Each event is considered a data structure (e.g. table, view, procedure) having the columns of start and end time points as date/time fields, where "before" is represented when the end of the first event is lower than the begin of the second event. As a result, the domain ontology, the MD schema specification, the requirements document and a high-level design of the ETL process were produced.

4.2 Construction and Deployment

A DB was physically created reflecting the MD schema specification. It supported the ETL process construction based on the ETL design and the domain ontology. It considered an ETL integration architecture coping with textual ETL, termed JointOLAP [9]. It uses IR and NLP techniques for the extraction process from text files and loads the terminological DB. The high-level data flow design is illustrated in Figure 3, having each activity supported by a set of tools. It begins with parallel activities: the conventional ETL process execution of disturbances Operational Data Store (ODS) and the textual ETL downloading news publications through a web crawler. Then, JointOLAP is performed in these documents, populating the terminological DB with all news articles content. It checks patterns in headers (e.g. title, publication date and press company), structuring this information in the DB.

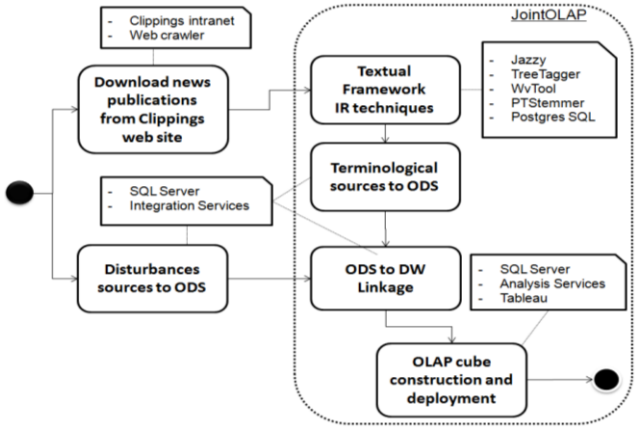


Figure 3. High-level ETL process workflow design.

The framework applies orthographic correction, case sensitive elimination, tokenizing and morphological classification, stop word and punctuation elimination, synonymous resolution and stemming. It indexes the terms, their stems, morphological

classification (e.g. verb, adjective, preposition and adverb). A list of business terms was created based on ONS official glossary and the terms of news articles were marked. An ETL process was created to execute the linkage activity, integrating data between disturbances and terminological ODSs and loading the final MD schema. It was used to build a data cube, making data accessible by OLAP tools.

The impact of blackouts on ONS corporate image analysis was supported through the OLAP tool connected to the data cube. The navigation was made possible through the dimensions and their hierarchies, enabling the exploration of the measures within the fact and possible aggregations with drill-down and roll-up operations, as well as graphics and reports generation. An analysis example is the number of terms published in news by the load cut level measure of the related disturbances. Average terms occurrences by disturbances is 5,720. Analyzing this measure by the severity of the disturbances makes it possible to verify a direct relation with the number of news publications. The more severe are the disturbances, more terms are published. In average the "blackout" term is, considering synonymous, the 27th most common term, but when load cut level is greater than 99MW it jumps to 1st. When it is lower than 49MW it becomes the 52nd of the blackout terms. News publications by disturbance month in 2011 revealed a significant variance of terms published after the disturbances occurred in February, which caused an enormous blackout in northeast. The number of publications increases considerably in March and April, then, it decreases back again to the standard baseline. These analyses are evidences that the expressivity enrichment of the MD design is a differential of our proposal, having the relation "before" connecting disturbances and news publications. The counting of mentions to certain words emphasized the press terminology when severe disturbances happened, addressing the main requirement of the solution.

5. RELATED WORK

Energy data management is a knowledge area that addresses the techniques for collecting, storing and analyzing huge amounts of data from the energy sector through IT solutions. Common definitions of data and information concepts by ontological approaches are still open topics [14]. Ontologies may be applied for the representation of portions of reality to understand, communicate and reason about the domain. In software engineering it is commonly built as UML class diagrams. In artificial intelligence it is commonly built as semantic networks and in DB area as ER diagrams. All these models seek to represent entities, relationships, properties, rules and restrictions of the involved domain. It can be considered formal when it is machine-processable, enabling automated reasoning by the semantics described in formal logic [4]. An example of an ontological solution for real-time data sources integration is the smart grid domain ontology introduced in [2]. It presents representations of event types, such as electrical appliances, weathers, storages and generators.

To fulfill analysis requirements in a BI/DW project, the MD and ETL design activities are supported by ontological solutions addressing the lack of semantic expressivity in MD models [1,12]. We introduced OntoWarehousing approach [10] where ontological analysis is applied in MD design based on formal ontology discipline. Analogous to formal logic, which contemplates logic formal structures, such as truth, validity and consistence [4], formal ontology is founded on mathematical disciplines of mereology (part-wholes), theory of dependence and topology and principles of identity and unity. In our approach we used the Unified Foundational Ontology (UFO) [3] to enrich the domain representation. It is a high-level category system, a top-level ontology, which presents these philosophical concepts interpreted, describing the most general concepts, such as space, time, matter, object, event and action, concepts independent of a domain or a particular problem. In OntoWarehousing, the domain ontology is semantically enriched by these top-level formalizations, e.g. domain concepts classified as events, participations, temporal relations, roles, among others. These high-level categories are used during the derivation process, which is based on a set of mapping rules from UFO categories to MD structures. The idea of such interpretation mapping from a foundational ontology to MD concepts was first discussed in [12].

A survey [1] summarized semantic web technologies (e.g. RDF and OWL) applied in BI/DW, discussing advantages, disadvantages and cases. Description logic can be used to assist data aggregation processing by reasoning services over rules. To enforce the semantics in MD design, Romero et al. [15] proposed the AMDO approach for conceptual modeling in BI/DW solutions based on end-user requirements elicitation and hybrid MD design. It uses a supply-driven mechanism where a set of rules formalized in first order logic derive MD structures (facts, measures, dimensions, hierarchies and attributes). The GEM approach [16] operationalizes the whole process, automating the identification of potential MD concepts by analyzing the domain ontology and the semantic annotations represented in OWL-DL. The ORE module [6] evolves GEM considering the complexity of frequent changes in MD design, integrating each new analytical requirement. These tools are mature enough, but they still lack some common understanding, which can be provided by a foundational ontology.

The need of considering unstructured data in BI/DW solutions is fundamental for business analytics. Even so, most of BI/DW methodologies are based on structured data. Analyzing and exploring data from heterogeneous natures, jointly, can enhance the potential of analytical applications offered to decision makers [5]. Several works are being proposed to consider the unstructured data sources by applying IR and NLP techniques as listed in [13]. We introduced the architecture JointOLAP [9] as a solution for joint exploration. It takes advantage of semantic treatment mechanisms for the unstructured content.

6. CONCLUSIONS AND FUTURE WORK

We introduced our approach as an adaptation of Moss's BI/DW methodology to consider heterogeneous data by making specific changes in the hybrid MD design activity. It takes advantage of IR and NLP techniques during the source-driven analysis phase to derive complementary analytical elements and associate data from structured and unstructured sources. In addition, we increased the MD design semantics by applying ontological engineering supported by a foundational ontology. A case study regarding ONS joint analysis of distribution consumption energy affected by press publications was described. The MD schema was derived considering the "before" temporal formal relation between disturbances and news publications. This case study is a work-in-progress, being considered as an original research and an industrial-strength solution for energy data management. Its main contribution is the integrated OLAP specification for ONS corporate image impact analyses built based on our approach. Indeed, the correlation between disturbances and news publications is not surprising. However, our case study could materialize this relation and its exploration using real data.

Lessons learned from our approach application on hybrid MD design activity include: (i) unstructured data sources proved to be essential information for MD conceptual design; (ii) ontological engineering seems to be an adequate method to improve knowledge acquisition and its design through a well-founded domain ontology; (iii) we believe this method may increase the productivity in business analysis and design phases of BI/DW projects. Some limitations are: (i) the derived ETL process did not considered implementation issues such as surrogate keys treatment and indexing management; (ii) to simplify, we considered a 1:1 relation between terms and categories, restricting the terms classification; (iii) the reverse engineering from text can be unfeasible depending on the amount of data; and (iv) the choice of MD concepts for the resulting MD schemas continues to be a tacit activity depending on the designer's decisions. Future work includes: (i) to enhance the textual ETL for news publications with new text treatment techniques, considering distributional models; (ii) to apply sentiment analysis techniques to discover the polarity of the sentiment around the events (e.g. positive, negative and neutral); (iii) to predict how quickly after an event the sentiment for or against ONS changes in the news and also to incorporate sentiment from crowd sources; (iv) we believe that entity recognition and relation extraction activities can consider categories of a foundational ontology; (v) regarding the involved tools, we believe that the prototype should be developed as an extension of OLED integrated to GEM/ORE.

7. ACKNOWLEDGMENTS

Our thanks to CAPES PhD scholarship (process BEX 1046/14-4).

8. REFERENCES

[1] Berlanga, R., Romero, O., Simitsis, A., Nebot, V., Pedersen, T. B., Abelló, A., Aramburu, M. J. 2011. Semantic Web Technologies for Business Intelligence. *BI Applications and the Web - Models, Systems, and Technologies*. pp. 310-339.

[2] Gillani, S., Laforest, F. e Picard, G. 2014. A Generic Ontology for Prosumer-Oriented Smart Grid. *3rd workshop on Energy Data Management (EnDM) - EDBT*.

[3] Guizzardi, G., Wagner, G., Falbo, R. A., Guizzardi, R. S. S., Almeida, J. P. A. 2013. Towards Ontological Foundations for the Conceptual Modeling of Events. *Conceptual Modeling, Lecture Notes in Computer Science.* pp. 327-341.

[4] Guizzardi, G.; Lopes, M.; Baião, F.; Falbo, R. 2010. On the importance of truly ontological representation languages. *Journal of Info. Systems Modeling and Design (IJISMD).*

[5] Inmon, W. H., Strauss, D. e Neushloss, G. 2008. *DW 2.0 - The Architecture for the Next Generation of DW*.

[6] Jovanovic, P., Romero, O., Simitsis, A., Abelló, A., Mayorova. 2014. A requirement-driven approach to the design and evolution of data warehouses. *Information Systems 44, 94-119*.

[7] Kimball, R. e Ross, M. 2013. *The Data Warehouse Toolkit: The definitve Guide to Dimensional Modeling.* Wiley.

[8] Malinowski, E. e Zimányi, E. 2009. *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications.* Springer.

[9] Moreira, J., Cordeiro, K. F. e Campos, M. L. M. 2013. JointOLAP – Sistema de Informação para Exploração Conjunta de Dados Estruturados e Textuais - Um estudo de caso no setor elétrico. *SBSI*.

[10] Moreira, J., Cordeiro, K. F., Campos, M. L. M., Borges, M.. 2014. OntoWarehousing – Multidimensional Design Supported by a Foundational Ontology: A Temporal Perspective. *16th International Conference on Data Warehousing and Knowledge Discovery (DaWaK/DEXA).*

[11] Moss, L. T. 2003 Business Intelligence Roadmap - The Complete Project Lifecycle for Decision-Support Applications. 1st. s.l. : *Addison-Wesley Professional*.

[12] Pardillo, J. and Mazón, J. N. 2011. Using Ontologies for the Design of Data Warehouses. *International Journal of Database Management Systems (IJDMS).* May, pp. 73–87.

[13] Park, B. K. and Song, I. Y. 2011. Toward Total Business Intelligence Incorporating Structured and Unstructured Data. *Proceedings of the 2nd International Workshop on Business intelligence and the WEB (BEWEB).* pp. 12-19.

[14] Pedersen, T. B. 2014. Energy Data Management: Where Are We Headed? *Panel paper, 3rd workshop on Energy Data Management (ENDM) - EDBT*.

[15] Romero, O. e Abelló, A. 2010. A framework for multidimensional design of data warehouses from ontologies. *Data & Knowledge Engineering Journal. Elsevier Science Publishers B. V.* Vol. 69, pp. 1138-1157.

[16] Romero, O., Simitsis, A. e Abelló, A. 2011. GEM: Requirement-Driven Generation of ETL and Multidimensional Conceptual Designs. *13th International Conference on Data Warehousing and Knowledge Discovery (DaWaK).* August, pp. 80-95

Measuring and Comparing Energy Flexibilities

Emmanouil Valsomatzis, Katja Hose, Torben Bach Pedersen, Laurynas Šikšnys
{evalsoma, khose, tbp, siksnys}@cs.aau.dk
Department of Computer Science, Aalborg University

ABSTRACT

Flexibility in energy supply and demand becomes more and more important with increasing Renewable Energy Sources (RES) production and the emergence of the Smart Grid. So-called *prosumers*, i.e., entities that produce and/or consume energy, can offer their inherent flexibilities through so-called demand response and thus help stabilize the energy markets. Thus, prosumer flexibility becomes valuable and the ongoing Danish project TotalFlex [1] explores the use of prosumer flexibility in the energy market using the concept of a flex-offer [2], which captures energy flexibilities in time and/or amount explicitly. However, in order to manage and price the flexibilities of flex-offers effectively, we must first be able to measure these flexibilities and compare them to each other. In this paper, we propose a number of possible flexibility definitions for flex-offers. We consider flexibility induced by time and amount individually, and by their combination. To this end, we introduce several flexibility measures that take into account the combined effect of time and energy on flex-offer flexibility and discuss their respective pros and cons through a number of realistic examples.

Keywords

Energy Flexibility, Flex-offers, Flexibility Measures

1. INTRODUCTION

A common challenging goal is to increase the use of energy produced by renewable energy sources (RES), such as wind and solar and at the same time reduce the CO_2 emissions. However, RES are characterized by fluctuating energy production and increased use of RES can lead to peaks (and valleys) in energy production and thus create congestion problems (or shortages) in the electric grid [5]. On the other hand, new devices such as heat pumps, increase the demand of energy and will lead to undesirable consumption peaks and a need for load shedding.

In this new energy scenery, the forthcoming Smart Grid [4] uses advanced information and communication infrastructures to activate the concept of demand side management (DSM) [6, 8]. According to DSM, the individual energy prosumers (producers and consumers) have a prominent role in the energy market due to their inherent flexibility. Flexibility can be used to mainly let the energy demand follow the energy supply and adjust the energy requirement according to energy production. The TotalFlex project explores the effect of prosumer flexibility on the energy market by introducing a new commodity using the *flex-offer* [2] concept that captures flexibilities in operating times and energy amounts of devices, as presented in the following use case.

Flex-offer use-case example. An electrical vehicle (EV) is plugged in and ready for charging at 23:00. Its battery is totally empty and it needs 3 hours to be charged. Moreover, its owner is satisfied with a minimum charging of 60% because this is sufficient enough for his needs tomorrow, e.g., going to work. Thus, we can see a flexibility regarding the energy demand of the EV due to the energy range satisfaction (60%−100%). Furthermore, the owner wants the car to be charged by 6:00 the latest, where he/she leaves home. As the battery requires 3 hours of charging, it should start being charged at 3:00 the latest. Therefore, we can also see a flexibility regarding the starting time range (23:00-3:00) of recharging the EV. The energy supplier is notified about the EV owner's energy requirement as well as the associated flexibilities in time and amount in the form of a flex-offer. Utilizing the flex-offer, the charging of the battery is scheduled (the starting time and energy demand for operating are assigned) at 1:00 because wind production will increase at that time. Furthermore, in order to ensure the owner's participation and to take advantage of the EV flexibility, the owner is offered lower energy tariff prices.

Flexibility, harnessed from many prosumers (using flex-offers) and handled according to the use-case example above, brings many advantages to society as well as to the actors participating in the energy market. Specifically, the utilization of RES is substantially increased and CO_2 emissions are reduced. Individual energy demands from prosumers are met and lower energy tariffs are offered. Marginal costs are reduced for Balanced Responsible Parties (BRPs) who trade energy. Congestion problems of Distributed System Operators (DSOs) can be handled without costly upgrades of physical grid infrastructures.

However, in order to take flexibility into consideration, we need to be able to measure *how much flexibility* is offered and

identify the *kind of flexibility* offered. Only with a proper flexibility measure, different flexibility offerings can be compared together. Focusing on the use-case of flex-offers and flexibility represented by these, we now present two scenarios where measuring flexibility is particularly useful.

Scenario Nr. 1 Flex-offers must be scheduled at some point in time to be able to satisfy the prosumers' energy needs, as described in the use case example above. Flex-offer scheduling problem [13], being similar to the unit commitment problem [9], is highly complex [12], when considering a large number of flex-offers, issued for a variety of appliances such as EVs, heat-pumps, dish washers, and smart refrigerators. To reduce the complexity of scheduling, flex-offer aggregation [15] plays a crucial role by trying to reduce the number of flex-offers while retaining as much as possible of their flexibility. In addition, the TotalFlex project is further utilizing the aggregation not only to reduce the number of the flex-offers, but also to partially handle the balancing task as well [14]. For all the aggregation techniques, it is essential to quantify and then to minimize flexibility losses, and therefore a flexibility measure is needed.

Scenario Nr. 2 Consider an energy market where flex-offers are traded. It is infeasible to trade flex-offers from individual prosumers directly in the market due to their small energy amounts. It is desirable for a BRP or for any other participating actor (e.g., an Aggregator) to first aggregate flex-offers from individual prosumers (e.g., household appliances) into "larger"' *aggregated flex-offers* (e.g., at the district level) before entering the market. Consequently, only large aggregated flex-offers are allowed to be traded in the market, and, when traded, used, e.g., by a BRP to ensure balance between the physically dispatched energy and energy traded in the energy spot-market, thus avoiding imbalance penalties. In this scenario, it is preferable for aggregated flex-offers to retain as much flexibility as possible in order to obtain a better value in the energy market when they are traded. Thus a flexibility measure to quantify flexibility of various flex-offers traded as commodities is needed.

In this paper, we employ the existing flex-offer definition [15] capturing flexibilities regarding time and energy amount. We assume that a flex-offer is already generated and it captures the energy and associated flexibility of a single prosumer unit (e.g., an EV). Our goal, is to express the flexibility, in time, amount, and both time and amount, with a single flexibility measure that can be applied on a single flex-offer or on a set of flex-offers. Therefore, we introduce 8 possible flexibility measures that can be used to quantify flexibilities of flex-offers and to compare flex-offers together in terms of their flexibilities. These include so-called *time*, *energy*, *product*, *vector*, *time-series*, *assignments*, *absolute area-based*, and *relative area-based* flexibility measures, which treat time and energy amount either as independent or dependent flex-offer dimensions. We discuss their advantages and disadvantages using illustrative real-world based examples. Our proposed flexibility definitions can be used not only for the valuing of flex-offers, but also for evaluation of flex-offer aggregation techniques and their algorithmic implementation. In fact, depending on the application needs, the flexibility of a flex-offer can be measured using one or more of the proposed measures, each with their advantage.

The remainder of the paper is structured as follows. In Section 3, we introduce and propose different flexibility definitions. We discuss in Section 4 about the use-case of the introduced definitions mentioning their pros and cons. We refer to related work in Section 5, and we conclude and mention our future work in Section 6.

2. PRELIMINARIES

In this paper, we consider the dimensions of *time* and *energy*, where time has the domain of natural numbers including zero ($\mathbb{N}_0$) and energy has the domain of integers ($\mathbb{Z}$). These assumptions are without loss of generality as we can achieve any desired finer granularity/precision of time and energy by simply multiplying their values with the desirable coefficient. Based on [15], we define a flex-offer according to Definition 1.

Definition 1. *A* flex-offer f *is a 2-tuple* $f=([t_{es}, t_{ls}], \langle s^{(1)}, \ldots, s^{(s)} \rangle)$. *The first element of the tuple denotes the start time flexibility interval where* $t_{es} \in \mathbb{N}_0$ *and* $t_{ls} \in \mathbb{N}_0$ *are the* earliest start time *and* latest start time*, respectively. The second element is a sequence of* s *consecutive slices that represents the energy profile. Each* slice $s^{(i)}$ *is an energy range* $[a_{min}, a_{max}]$*, where* $a_{min} \in \mathbb{Z}$ *and* $a_{max} \in \mathbb{Z}$*. The duration of slices is 1 time unit.*

A flex-offer also has a total minimum c_{min} and a maximum c_{max} energy constraint. The minimum constraint is smaller than or equal to the maximum one and they are lower and upper bounded by the sum of all the minimums and the sum of all the maximums of energy of the slices, respectively. If all the energy values of a flex-offer are positive then the flex-offer represents energy consumption (positive flex-offer), e.g., a dishwasher. If all the energy values of a flex-offer are negative then the flex-offer represents energy production (negative flex-offer), e.g. a solar panel. If the energy values of a flex-offer are both positive and negative then the flex-offer represents both energy consumption and production (mixed flex-offer), e.g., a "vehicle-to-grid".

A flex-offer f can be instantiated into a so-called *assignment* of f, f_a, is a time series defining the starting time and the exact energy amounts satisfying all flex-offer constraints.

Definition 2. *An* assignment f_a *of a flex-offer* $f = ([t_{es}, t_{ls}], \langle s^{(1)}, \ldots, s^{(s)} \rangle)$ *is a time series* $\{f_a\}_{t=t_{start}}^{t_{start}+s} = \langle v^{(1)}, \ldots, v^{(s)} \rangle$ *such that:*

- $t_{es} \leq t_{start} \leq t_{ls}$
- $\forall i = 1..s : s^{(i)}.a_{min} \leq v^{(i)} \leq s^{(i)}.a_{max}$
- $c_{min} \leq \sum_{i=1}^{s} v^{(i)} \leq c_{max}$

A (valid) flex-offer assignment satisfies the constraints of a flex-offer. Specifically, for each slice of the flex-offer, the assignment has a corresponding energy value which must be within the corresponding slice energy range of the flex-offer. In addition, the sum of the energy values of a flex-offer assignment must be within the total minimum and the total maximum energy constraints of the flex-offer. Furthermore, the first non-zero energy value of the assignment that defines

the actual starting time of the flex-offer must be within the start time flexibility interval of the flex-offer. A single flex-offer (typically) has several flex-offer assignments. We use the set $L(f)$ to define all (valid) flex-offer assignments. For instance, Figure 1 illustrates a flex-offer with four slices $f = ([1,6], \langle [1,3], [2,4], [0,5], [0,3] \rangle)$. One valid assignment of f is $f_{a1} \in L(f)$ such that $\{f_{a1}\}_{t=2}^{5} = \langle 2,3,1,2 \rangle$, shown as bold lines in Figure 1.

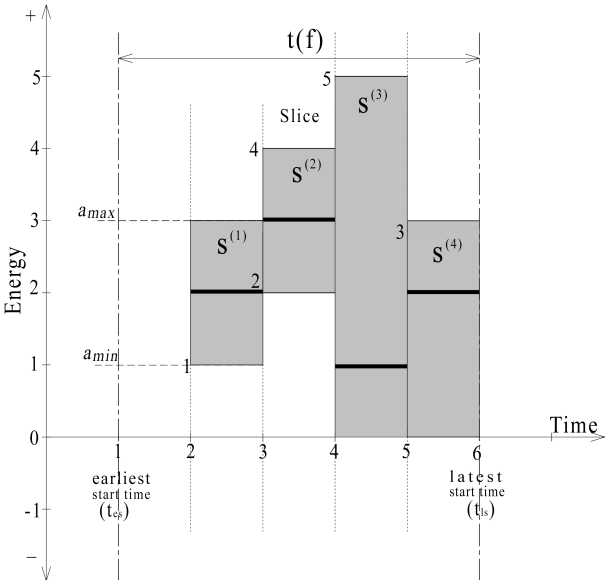


Figure 1: Illustration of a flex-offer f

3. FLEXIBILITY DEFINITIONS AND MEASURES

We now introduce different flexibility definitions and measures associated with a flex-offer.

3.1 Time and energy flexibility

There are two different types of flexibilities associated with a flex-offer, either derived by the starting time interval or by the energy ranges of the slices.

Based on the flexibility definitions introduced in [15], we consider the *time flexibility* $tf(f)$ of a flex-offer f to be the difference between the latest and the earliest start time of f, measured in time units, i.e., $tf(f) = f.t_{ls} - f.t_{es}$.

Example 1. *The flex-offer f in Figure 1 has t_{ls}=6 and t_{es}=1, thus time flexibility is: $tf(f) = 6 - 1 = 5$.*

Moreover, since the total maximum and the total minimum energy constraints impose the allowed energy range of a flex-offer, we also define *energy flexibility* of a flex-offer f to be the difference between the total maximum and the total minimum energy constraints, i.e., $ef(f) = c\_max(f) - c\_min(f)$

Example 2. *The flex-offer f in Figure 1 has the sum of maximum slice values equal to* 15 *and the sum of minimum slice values equal to* 3*. Given that, $c\_max(f)$=15, $c\_min(f)$=3, and the energy flexibility of f is $ef(f)$=15−3=12.*

3.2 Combined flexibility measures

As seen above, quantifying either time or energy flexibilities on their own is rather straightforward. It is more tricky to consider them in combination. Therefore, we now define and discuss several alternative measures for this.

Product flexibility. The existing definition of *total flexibility* [15] originally specified the total (joint) flexibility of a flex-offer f as the product of the time flexibility and the sum of the energy flexibilities of all the slices. However, as we have additionally introduced the total energy constraints of a flex-offer, we define the *product flexibility* of a flex-offer as follows:

Definition 3. *The* product flexibility *$product\_flexibility(f)$ of a flex-offer f is the product of the time flexibility and the energy flexibility of f, i.e., $product\_flexibility(f) = tf(f) \cdot ef(f)$.*

Example 3. *The flex-offer f in Figure 1 has product flexibility $product\_flexibility(f) = 5 \cdot 12 = 60$.*

Vector flexibility. Since a flex-offer is characterized by both time and energy we define the flexibility of a flex-offer to be a vector where time and energy flexibilities are the vector components.

Definition 4. *The vector flexibility $vector\_flexibility(f)$ of a flex-offer f is a vector v with 2 components. The first component of the vector is the time flexibility of f, and the second component is the energy flexibility, i.e., $v = \langle tf(f), ef(f) \rangle$.*

The total flexibility is then intuitively given by the "length" of the vector, computed using a given norm. Possible relevant norms in our two dimensions include Manhattan (L^{1-norm}) and Euclidean norm (L^{2-norm}).

Example 4. *The flex-offer f in Figure 1 has vector flexibility $vector\_flexibility(f) = \langle 5, 10 \rangle$, and we can compute its length as either $\|vector\_flexibility(f)\|_1$=5+10=15 or $\|vector\_flexibility(f)\|_2$=$\sqrt{(5^2 + 10^2)}$=11.180.*

Time-series flexibility. A flex-offer allows multiple assignments, each expressing a possible instantiation of the flex-offer. Since every assignment of a flex-offer is a time series, the difference between two assignments is also a time series. We consider the two most dissimilar time series (assignments), *minimum* and *maximum*, defined as follows:

Definition 5. *The minimum assignment $f_a^{min}(f)$ of a flex-offer $f = ([t_{es}, t_{ls}], \langle s^{(1)}, \ldots, s^{(s)} \rangle)$ is the assignment with the first energy value positioned at the earliest starting time of f and energy values equal to the minimum slice values of f, i.e., $f_a^{min}(f) = t$, where $\{t\}_{t=t_{es}}^{t_{es}+s} = \langle f.s^{(1)}.a_{min}, \ldots, f.s^{(s)}.a_{min} \rangle$.*

Definition 6. *The maximum assignment $f_a^{max}(f)$ of a flex-offer $f = ([t_{es}, t_{ls}], \langle s^{(1)}, \ldots, s^{(s)} \rangle)$ is the assignment with the first energy value positioned at the latest starting time of f and energy values equal to the maximum slice values of f, i.e., $f_a^{max}(f) = t$, where $\{t\}_{t=t_{ls}}^{t_{ls}+s} = \langle f.s^{(1)}.a_{max}, \ldots, f.s^{(s)}.a_{max} \rangle$.*

Using minimum and maximum assignments, we define *series flexibility* as follows:

Definition 7. *The time series flexibility, series_flexibility(f), of a flex-offer f is the difference the maximum and the minimum assignments of f (time series), i.e., series_flexibility(f)=$f_a^{max}(f)$-$f_a^{min}(f)$.*

Since we use two dimensions, we again propose the Manhattan and Euclidean norms to quantify the difference between two assignments.

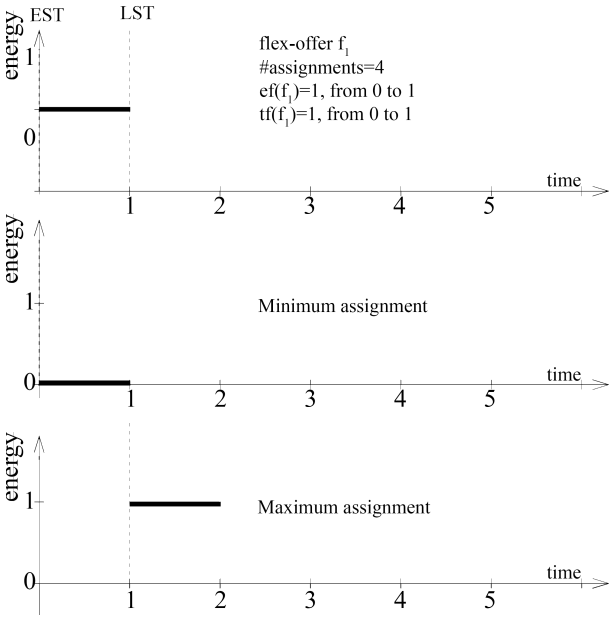


Figure 2: Time series definition example with $ef(f_1) = 1$ and $tf(f_1) = 1$

Example 5. *Figure 2 illustrates a flex-offer f_1 with 1 slice, earliest start time = 0, and latest start time = 1, $f_1 = ([0, 1], \langle[0, 1]\rangle)$, $c_{min}(f_1) = 0$, and $c_{max}(f_1) = 1$.*

Flex-offer f_1 has 4 assignments, and the following minimum and maximum assignments: $\{f_{1a}^{min}(f_1)\}_{t=0}^{1} = \langle 0, 0\rangle$, $\{f_{1a}^{max}(f_1)\}_{t=0}^{1} = \langle 0, 1\rangle$. Let the difference between $f_{1a}^{max}(f_1)$ and $f_{1a}^{min}(f_1)$ be f_{d1} so that f_{d1}=$f_{1a}^{max}(f_1)$-$f_{1a}^{min}(f_1)$. In this example $\{f_{d1}\}_{t=0}^{1} = \langle 0, 1\rangle$, L^{1-norm}, $|\{f_{d1}\}_{t=1}^{1}|_1 = 1$, and L^{2-norm}, $|\{f_{d1}\}_{t=1}^{1}|_2 = 1$. According to both L^{1-norm} and L^{2-norm}, series_flexibility(f_1)=1.

Assignment flexibility. As mentioned in Section 2, a flex-offer allows a number of possible assignments. The number of possible assignments directly depends on time and energy flexibility and is the number of the combinations between all the allowed amount and time values of all its slices. Therefore, we use the number of possible assignments as a combined measure induced by both time and amount flexibility.

Definition 8. *We define* assignment flexibility, *assignment_flexibility(f), of a flex-offer $f = ([t_{es}, t_{ls}], \langle s^{(1)}, \ldots, s^{(s)}\rangle)$ to be the number of all possible assignments of f, i .e., assignment_flexibility(f)=*

$$=(t_{ls}-t_{es}+1)\cdot\prod_{i=1}^{s}(s^{(i)}.a_{max}-s^{(i)}.a_{min}+1).$$

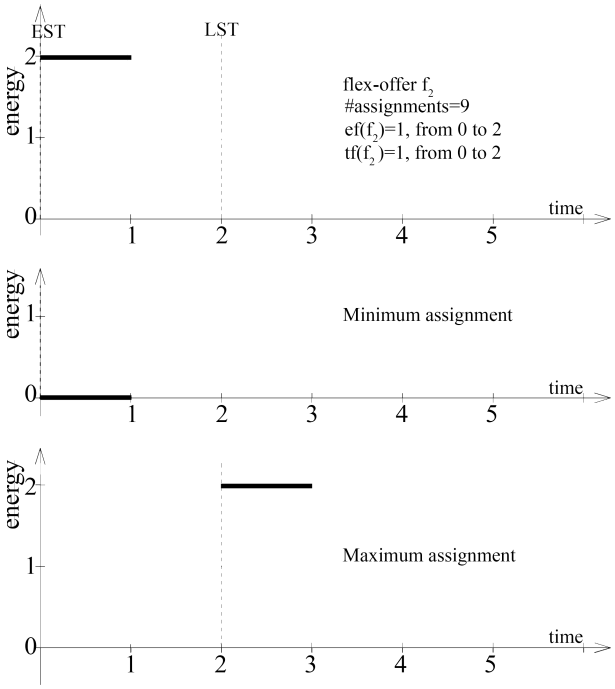


Figure 3: Number of assignments example with $ef(f_2) = 2$ and $tf(f_2) = 2$

Example 6. *Flex-offer $f_2 = ([0, 2], \langle[0, 2]\rangle)$ in Figure 3 has t_{ls}−t_{es}+1=3 and since it has one slice $s^{(1)}.a_{max}$−$s^{(1)}.a_{min}$+1=3. Thus, f_2 has 9 assignments in total.*

Absolute area-based flexibility. Absolute area-based flexibility is based on the area that all flex-offer assignments jointly cover, considering all of their possible values of start time and energy. As a basis for calculating this area, we consider a two-dimensional (time and energy) grid $G = \mathbb{N}_0 \times \mathbb{Z} = \{(t, e) : t \in time, e \in energy\}$, in which the x axis corresponds to discretized time and the y axis to discretized energy. Cells of the grid are identified by their lower left coordinates. For instance, the cell with identifier $(0, 0)$ has the following corner coordinates: $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$.

First, we define the area of a single flex-offer assignment.

Definition 9. *The area of an assignment f_a of a flex-offer f, denoted as $area(f_a)$, is the set of cells that falls between the f_a energy values and the X-axis of the grid.*

Example 7. *Given an* assignment *of flex-offer f_3, $\{f_{3a}\}_{t=1}^{3} = \langle 2, 1, 3\rangle$ the area is as follows: $area(\{f_{3a}\}_{t=1}^{3}) = \{(1, 0), (1, 1), (2, 0), (3, 0), (3, 1), (3, 2)\}$, which is represented by the hatched cells in Figure 4.*

This area represents the total assigned amount of a single flex-offer. However, multiple assignments with different areas are possible for a flex-offer. The total coverage of all these assignment areas gives us the area of the flex-offer flexibility. This joint area expresses all the possible amounts at all the possible time instances that a flex-offer

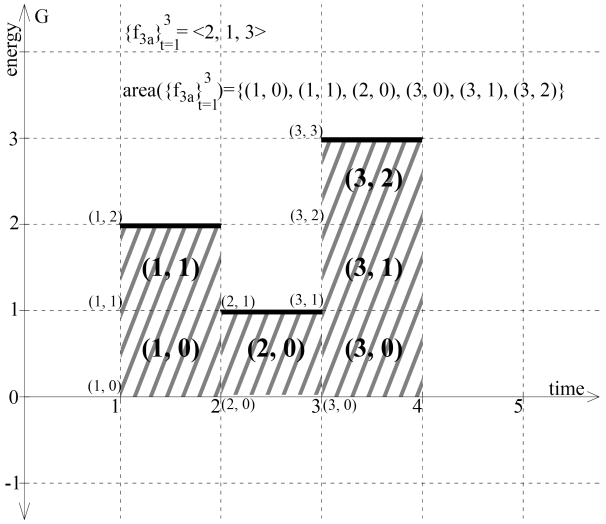


Figure 4: Area of the assignment $\{f_{3a}\}_{t=1}^{3} = \langle 2, 1, 3\rangle$

could have. Furthermore, we are interested in the size (a numerical value) of this area of flexibility. To specify this, we additionally take into account the minimum total energy constraint *c_min*, which is applicable to all assignments and is thus considered inflexible.

Definition 10. *The* absolute area-based flexibility *of a flex-offer f is the difference between the size of the total area covered by all the assignments of f and the total minimum constraint of f: absolute_area_flexibility=$|\bigcup_{as\_f \in L(f)} area(as\_f)| - c\_min(f)$*

Example 8. *Figure 5 illustrates the flex-offer $f_4 = ([0,4], \langle [2,2] \rangle)$, $c_{min}(f_4)$=2, and $c_{max}(f_4)$=2. Flex-offer f_4 has 5 different assignments and each one covers an area of two cells, see Figure 5. Flex-offer f_4 has absolute_area_flexibility(f_4)=10−2=8.*

Example 9. *Figure 6 illustrates the flex-offer $f_5 = ([0,4], \langle [1,1], [2,2] \rangle)$, $c_{min}(f_5)$=3, and $c_{max}(f_5)$=3. Flex-offer f_5 has 5 different assignments and each one covers an area of three cells, see Figure 6. Flex-offer f_5 has absolute_area_flexibility(f_5)=10−2=8.*

Relative area-based flexibility. For most of the presented flexibility measures (incl., absolute area-based flexibility), the value of the flexibility depends on the actual amounts specified in the flex-offer. However, in cases when we need to compare flex-offers of different sizes in terms of amount, we need a size-independent measure. For these cases, we propose a *relative area-based flexibility.*

Definition 11. *The relative area-based flexibility of a flex-offer f is equal to the absolute flexibility divided by the average of the energy total constraints of f:*

$$relative\_area\_flexibility(f) = \frac{2 * absolute\_area\_flexibility(f)}{|c\_min(f)| + |c\_max(f)|}, \quad |c\_min(f)| + |c\_max(f)| \neq 0$$

Example 10. *Flex-offer $f_4 = ([0,4], \langle [2,2] \rangle)$, $c_{min}(f_4)$=2, $c_{max}(f_4)$=2, shown in Figure 5, has relative_area_flexibility(f_4)=$\frac{2*8}{|2|+|2|}$=4. Flex-offer $f_5 = ([0,4], \langle [1,1], [2,2] \rangle)$, $c_{min}(f_5)$=3, $c_{max}(f_5)$=3, shown in Figure 6, has relative_area_flexibility(f_5)=$\frac{2*8}{|3|+|3|}$=16/6.*

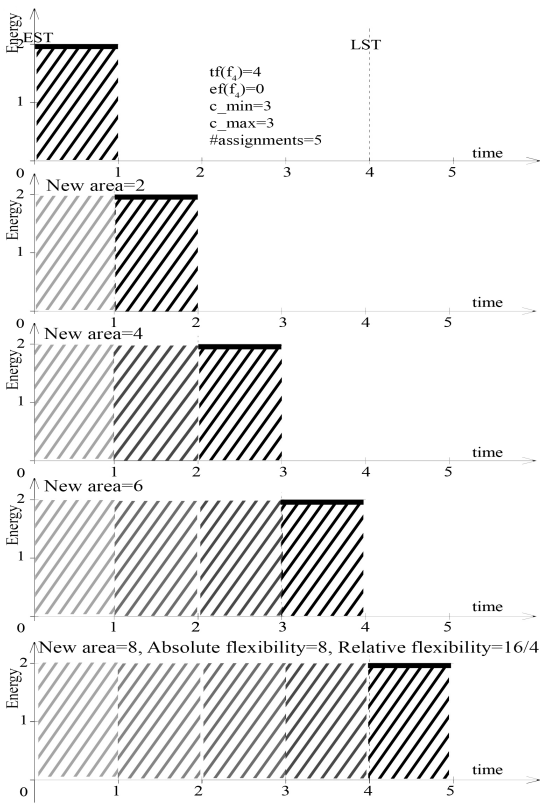


Figure 5: Absolute and relative area-based flexibility of the flex-offer f_4

4. DISCUSSION

In this section, we discuss the pros and cons of the proposed flexibility measures, and scenarios in which we can use each of these measures.

Product flexibility. The product flexibility measure, defined in Definition 3, is only applicable in cases when a flex-offer f has positive time and energy flexibilities, i.e., $tf(f) > 0$ and $ef(f) > 0$. In cases, when either the time or the amount flexibility is equal to zero, the value of the product flexibility is also equal to zero. As the flex-offer is still flexible in the other dimension (time or energy), this measure is not particularly accurate.

Example 11. *Flex-offer f_x=($[2,8], \langle [5,5] \rangle$) has $tf(f)$=6, $ef(f_x)$=0, and product_flexibility(f_x) = $6 \cdot 0 = 0$. Moreover, two flex-offers f_x=($[1,3], \langle [1,5] \rangle$) and f_y=($[1,3], \langle [101,105] \rangle$) have equal product flexibility values, i.e., product_flexibility(f_x)=product_flexibility(f_y)=8, even if the minimum energy requirement of f_y is more than 100 times greater than the minimum energy requirement of f_x.*

Furthermore, product flexibility does not take into account individual slice energy requirements. It relies only on total energy requirements (c_{min} and c_{max}). Nevertheless, Defi-

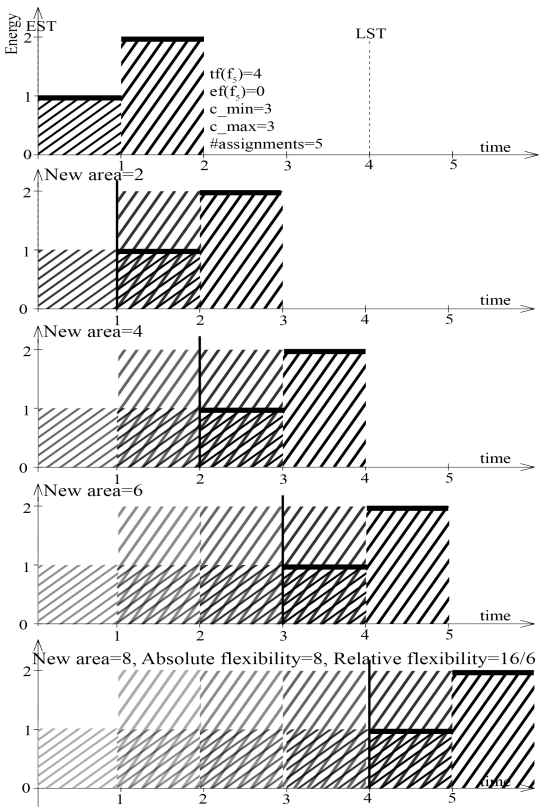


Figure 6: Absolute and relative area-based flexibility of the flex-offer f_5

nition 3 can still be applicable in scenarios where the flex-offer represents production, consumption, or both, as long as there are no mixed flex-offers. Additionally, it can be generalized for sets of flex-offers. To compare two or more sets of flex-offers, we should sum the product flexibilities of the flex-offers in each set.

Vector flexibility. Vector flexibility measure, as defined in Definition 4, can be applicable to either individual flex-offers or sets of flex-offers, like the product flexibility. However, unlike the product flexibility, it can capture the flexibility in cases where either time or energy flexibility of a flex-offer is equal to zero. Furthermore, it is independent of the sign of the energy values of the slices of a flex-offer. In particular, it can express flexibility of flex-offers that represent either energy production, consumption, or both. Like the product flexibility, it does not take into account individual slice energy requirements, solely relying on total energy requirements (c_{min} and c_{max}). Lastly, vector flexibility does not take into account the actual values of energy ("size of the flex-offer"), but, instead, captures only the difference between energy bounds.

Example 12. *The two flex-offers* f_x=$([1,3],\langle[1,5]\rangle)$ *and* f_y=$([1,3],\langle[101,105]\rangle)$ *from Example 11 have the same vector flexibility irrespectively of the used norm, even if the minimum energy requirement of* f_y *is more than* 100 *times greater than the minimum energy requirement of* f_x*. Specifically,* $\|vector\_flexibility(f_x)\|_1$=$\|vector\_flexibility(f_y)\|_1$=6 *according to the Manhattan norm, and* $\|vector\_flexibility(f_x)\|$=$\|vector\_flexibility(f_y)\|_2$=4.472 *according to the Euclidean norm.*

Time-series flexibility. Norms such as Manhattan and Euclidean, applicable with time-series flexibility (see Definition 7), do not take into account the temporal structure of the time series [7] and thus cannot capture the joint effect of time and energy flexibilities. As a result even if time-series captures both time and energy, the norms applied on a difference between time-series can capture only energy flexibility. However, the time-series definition can be applied on positive, negative, and mixed flex-offers, as well as on sets of flex-offers – by computing the sum of time-series flexibilities of the flex-offers in the set.

Example 13. *As mentioned in Example 5 , flex-offer* $f_1 = ([0,1],\langle[0,1]\rangle)$*,* $c_{min}(f_1) = 0$*, and* $c_{max}(f_1) = 1$ *results in time series* $\{f_{d1}\}_{t=0}^{1} = \langle 0,1\rangle$*, and its norms are as follows:* L^{1-norm}*,* $|\{f_{d1}\}_{t=1}^{1}|_1 = 1$*, and* L^{2-norm}*,* $|\{f_{d1}\}_{t=1}^{1}|_2 = 1$*. However, another flex-offer* $f_1' = ([0,10],\langle[0,1]\rangle)$*,* $c_{min}(f_1') = 0$*, and* $c_{max}(f_1') = 1$ *with* 10 *times greater time flexibility than* f_1 *results in a similar time series* $\{f_{d1}'\}_{t=0}^{1} = \langle 0,0,0,0,0,0,0,0,0,0,1\rangle$ *with identical norms:* L^{1-norm}*,* $|\{f_{d1}'\}_{t=1}^{1}|_1 = 1$*, and* L^{2-norm}*,* $|\{f_{d1}'\}_{t=1}^{1}|_2 = 1$*.*

Assignment flexibility. Assignment flexibility, as defined in Definition 8, considers only the number of flex-offer assignments, and this number is independent of the actual values of the time and energy bounds. The limitation of this measure is that energy flexibility has an exponential impact on the number of the assignments, i.e., the number of assignments increases exponentially when energy flexibility is increased. In comparison, the number of flex-offer assignments increases linearly when time flexibility is increased. Thus, this measure favors energy flexibility over time flexibility. Moreover, assignment flexibility, as defined in Definition 8, does not take into account the total energy requirements (c_{min} and c_{max}), and gives the same values for flex-offers with the same time and amount flexibilities, but differing in energy amounts. Furthermore, it can express flexibility of flex-offers that represent either production, consumption, or both. It can be used to compare individual flex-offers and to compare sets of flex-offers by counting the number of possible assignments for the whole set.

Example 14. *The flex-offer* f_2 *with* $tf(f_2)$=$ef(f_2)$=2*, shown in Figure 3, has* 9 *possible assignments. If* $tf(f_2)$ *were* 0*, flex-offer* f_2 *would have* 3 *possible assignments, but if* $ef(f_2)$ *were* 0*,* f_2 *would have* 2 *possible assignments. The flex-offer* f_6 *with* $tf(f_6)$=2 *and* $ef(f_6)$=10*, shown in Figure 7, has* 240 *assignments. If* $tf(f_6)$ *were* 0*,* f_6 *would have* 80 *assignments, but if* $ef(f_6)$ *were* 0*,* f_6 *would have* 3 *assignments.*

Absolute and relative area-based flexibility. Both the absolute and relative area-based flexibility measures (Definitions 10–11) can be used to capture the joint effect of time and energy flexibilities. However, the absolute area-based flexibility measure should only be used for (pure) consumption flex-offers only, as its value is adjusted using the total minimum energy constraint (c_{min}), which is meaningful only for the consumption case where amounts are positive. For the production flex-offer case, where amounts are negative,

Characteristics	Flexibility Measures							
	Time	Energy	Product	Vector	Time-series	Assignments	Abs. Area	Rel. Area
Captures time	Yes	No	No	Yes	No	Yes	Yes	Yes
Captures energy	No	Yes	No	Yes	Yes	Yes	Yes	Yes
Captures time & energy	No	No	Yes	Yes	No	Yes	Yes	Yes
Captures size	No	No	No	No	No	No	Yes	Yes
Captures positive flex-offers	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Captures negative flex-offers	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Captures Mixed flex-offers	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Single Value	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 1: Flexibility definitions characteristics.

the total maximum energy constraint (c_{max}) should be used instead. However, for cases when the flex-offer represents both production and consumption, this flexibility measure is not feasible.

Example 15. *For instance, flex-offer* f_4=([0,2],⟨[−1,2],[−1,−4],[−3,1]⟩) *in Figure 7 has* $c_{min}(f_6)$=−8 *and* $c_{max}(f_6)$=2, *but neither of them expresses the lower or upper bounds of the area, jointly covered by the assignments of* f_6. *In this case, absolute_area_flexibility*(f_6)=24−(−8)=32 *and relative_area_flexibility*(f_6)= $\frac{2*32}{|8|+|2|}$=6.4.

On the other hand, both absolute and relative area-based flexibility measures can be used to compare individual flex-offers. Only absolute area-based flexibility can be used to compare the total absolute flexibility of two or more sets of flex-offers, e.g., by summing up the individual absolute area-based flexibility values of the flex-offers in the sets. To assess the relative flexibility for a set of flex-offers, the sum of relative flexibilities is not meaningful, instead the average relative flexibility could be used.

All the flexibility measures can be applied for both individual flex-offers and sets of flex-offers to compare their underlying flexibility. However, as we see in Table 1, which summarizes the characteristics of all the proposed flexibility definitions, each flexibility measure has specific characteristics and should be used under specific circumstances only. For example, the product flexibility measure cannot properly capture flexibility unless both time and amount flexibility is exhibited. The time-series flexibility measure captures only flexibility induced by energy flexibility. Only the absolute and relative area-based flexibility measures take into account the amount values (size) of the flex-offers. However, the absolute and relative area-based flexibility measures have problems expressing the flexibility of mixed flex-offers.

Application Scenarios. There are 2 major scenarios (see Section 1) where the different measures can be applied. In Scenario 1, the goal of aggregation is to reduce the input complexity of scheduling and retain as much flexibility of flex-offers as possible. In this scenario, measures that capture flexibility induced by both time and energy, e.g., product flexibility and assignments flexibility, are qualified. Measures that capture only time or energy flexibility, such as time-series flexibility, are not appropriate for Scenario 1. However, in cases where aggregation handles the balancing task as well, measures that capture flexibility of

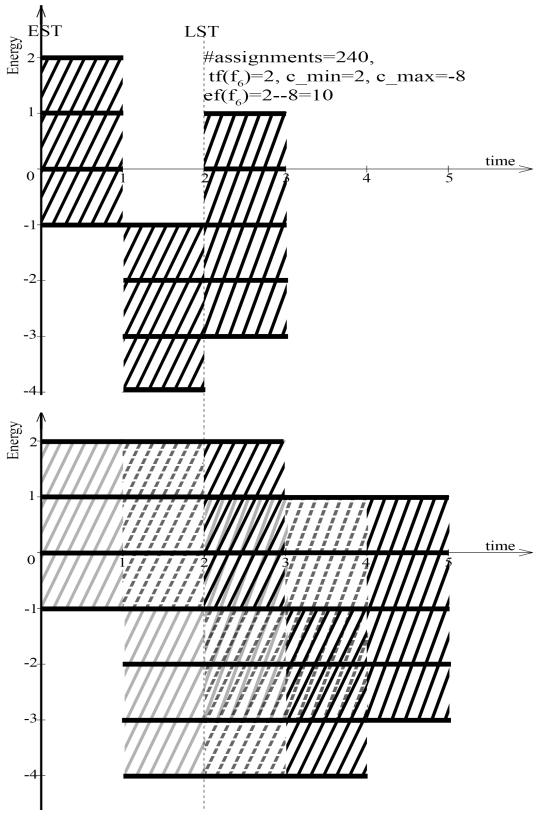


Figure 7: Number of assignments example, flex-offer f_6

mixed flex-offers are needed since the aggregated flex-offers might be mixed ones. Thus, measures that are not suitable for mixed flex-offers, i.e., absolute and relative area-based flexibility, are inappropriate to express flexibility. Instead, measures that capture flexibility of mixed flex-offers such as vector and assignments flexibility, are qualified. In Scenario 2, where an energy market actor (e.g., an Aggregator) trades flex-offers as commodities, measures capturing only time or energy can be used. The reason is because an Aggregator might handle flex-offers from specific appliances that are characterized only by time or energy flexibility. Thus, the time-series measure, the time and energy flexibility measures, and the product flexibility measure are appropriate. In cases where an Aggregator wants to explore and evaluate the potentials of achieving a local balance and handle a power capacity limitation, measures for mixed flex-offers are

more appropriate. However, only absolute and relative area-based flexibilities take into account the size of a flex-offer, but they cannot be applied on mixed flex-offers. Therefore, a combination of measures that includes the absolute or the relative area-based flexibility can be used to handle these more complex cases. *Weighting* is one way of combining different flexibility measures and balancing their influences to fulfill specific characteristics mentioned in Table 1.

5. RELATED WORK

Flexibility in energy supply and demand has a prominent role in the Smart Grid domain, and, among others within this domain, can be associated with distributed generation, load management and demand side management [6]. Many definitions of flexibility have been proposed, but a formal universal definition is still pending [10]. Some proposed measures of flexibility focus on operational aspects and take into account transmission constraints [3], while others are based on time shifting of loads [11]. Furthermore, there has been proposed categorizations of power units based on their characteristics, taking into consideration their qualities and capabilities to dispatch power and solve balancing issues [10].

In comparison, this paper proposes and discusses specific measures to quantify flexibility in energy supply and demand, namely in the units connected to the Smart Grid such as electric vehicles, solar panels, wind turbines, and refrigerators. We use the existing definition of a flex-offer [15], which is a generic model for representing flexibility and adjust it for the cases of energy consumption, production, and both consumption and production. The proposed measures can be applied on individual electrical units and on sets of units as well, e.g., when solving the unit commitment problem [9] or tackling balancing or congestion problems occurring in the grid [13].

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed and explored 8 measures for quantifying flexibility in demand and supply based on the generic flexibility model of a flex-offer, capturing the energy behavior of units connected to the Smart Grid. We identified the independent flexibilities of time and energy and proposed a number of combined measures – *product*, *vector*, *time-series*, *assignments*, *absolute area-based*, and *relative area-based* – which take both time and energy into account. These measures can be used to compare the flexibility of individual flex-offers as well as sets of flex-offers. We demonstrated and discussed the impact of the proposed measures using elaborate graphical examples. We concluded through a discussion that such single-value measures can be used to express the flexibility of the units connected to the Smart Grid. However, none of the measures have all the desirable characteristics. Instead, each measure has specific characteristics and can be used in specific circumstances, all discussed in the paper.

In future work, we will examine the use of the suggested measures for flex-offer aggregation algorithms, including those that partially address the energy balancing problem and consider electric grid constraints. The proposed flexibility measures will be added to the constraints and/or objective functions of these aggregation algorithms, performing aggregation jointly with flexibility optimization. We will also experimentally evaluate the flexibility measures and their effect on the scheduling process in different scenarios. Moreover, we will extend the current proposals to new types of measures capturing more aspects of flexible electrical loads.

7. ACKNOWLEDGMENTS

This work was supported in part by the TotalFlex project sponsored by the ForskEL program of Energinet.dk.

8. REFERENCES

[1] Totalflex project, link: `www.totalflex.dk`.
[2] M. Boehm, L. Dannecker, A. Doms, E. Dovgan, B. Filipic, U. Fischer, W. Lehner, T. B. Pedersen, Y. Pitarch, L. Šikšnys, and T. Tušar. Data management in the mirabel smart grid system. In *Proc. of EnDM*, 2012.
[3] M. Bucher, S. Chatzivasileiadis, and G. Andersson. Managing flexibility in multi-area power systems. *CoRR*, abs/1409.2234, 2014.
[4] H. Farhangi. The path of the smart grid. *Power and Energy Magazine, IEEE*, 2010.
[5] H. Hermanns and H. Wiechmann. Future design challenges for electric energy supply. In *Proc. of ETFA*, 2009.
[6] F. Kupzog and C. Roesener. A closer look on load management. In *Industrial Informatics, IEEE International Conference on*, 2007.
[7] J. A. Lee and M. Verleysen. M.: Generalization of the lp norm for time series and its application to self-organizing maps. In *COTTRELL, M. (Hrsg.): Proc. of WSOM*, 2005.
[8] A.-H. Mohsenian-Rad, V. Wong, J. Jatskevich, R. Schober, and A. Leon-Garcia. Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid. *Smart Grid, IEEE Transactions on*, 2010.
[9] N. Padhy. Unit commitment-a bibliographical survey. *IEEE Transactions on Power Systems*, 2004.
[10] M. Petersen, K. Edlund, L. Hansen, J. Bendtsen, and J. Stoustrup. A taxonomy for modeling flexibility and a computationally efficient algorithm for dispatch in smart grids. In *ACC*, 2013.
[11] K. Pollhammer, F. Kupzog, T. Gamauf, and M. Kremen. Modeling of demand side shifting potentials for smart power grids. In *AFRICON*, 2011.
[12] T. Tušar, E. Dovgan, and B. Filipic. Evolutionary scheduling of flexible offers for balancing electricity supply and demand. In *IEEE Congress on Evolutionary Computation (CEC)*, 2012.
[13] T. Tušar, L. Šikšnys, T. B. Pedersen, E. Dovgan, and B. Filipič. Using aggregation to improve the scheduling of flexible energy offers. *International Conference on Bioinspired Optimization Methods and their Applications*, 2012.
[14] E. Valsomatzis, K. Hose, and T. B. Pedersen. Balancing energy flexibilities through aggregation. In *Proc. of DARE*, 2014.
[15] L. Šikšnys, M. E. Khalefa, and T. B. Pedersen. Aggregating and disaggregating flexibility objects. In *Proc. of SSDBM*, 2012.

What's Wrong with my Solar Panels: a Data-Driven Approach

Xiang Gao, Lukasz Golab and S. Keshav
University of Waterloo
200 University Avenue West
Waterloo, Ontario, Canada N2L 3G1
{x39gao,lgolab,keshav}@uwaterloo.ca

ABSTRACT

Solar panels have been improving in efficiency and dropping in price, and are therefore becoming more common and economically viable. However, the performance of solar panels depends not only on the weather, but also on other external factors such as shadow, dirt, dust, etc. In this paper, we describe a simple and practical data-driven method for classifying anomalies in the power output of solar panels. In particular, we propose and experimentally verify (using two solar panel arrays in Ontario, Canada) a simple classification rule based on physical properties of solar radiation that can distinguish between shadows and direct covering of the panel, e.g,. by dirt or snow.

1. INTRODUCTION

Photovoltaic (PV) technology, i.e., solar panels, has been rapidly dropping in price and increasing in popularity worldwide [7]. The monitoring and measuring capability of PV installations has also improved. While it used to be possible only to measure the total power output of an array of solar panels, micro-inverters (which are devices that covert Direct Current generated by an *individual* panel into Alternating Current) now make it possible to measure the power output of each individual panel at fine granularities (e.g., every minute or every five minutes). Thus, solar panel data analytics is becoming an important area of research and practice.

The power output of a PV system depends on solar intensity and the panels' efficiency of converting light into power (typically 15-20 percent). Additionally, even a perfectly-functioning panel on a sunny day will produce little power if it is shaded or covered by dust or dirt. For instance, many large-scale PV installations are located on farmlands and/or near country roads, which makes them vulnerable to dust, mud, pollen and other types of soiling. Furthermore, even if a farm site is chosen to be shadow-free, grass may eventually grow tall enough to cast shadows on the panels. Numerous studies have observed power drops of 40 or more percent due to shaded, dirty and snow-covered panels [1, 2, 3, 4, 6, 8, 11, 15, 18, 23, 25].

A simple solution is to frequently clean the panels. However, this is not feasible in desert locations that suffer from water shortages, or in remote large-scale installations where an automated sprinkler system is prohibitively expensive. Some PV installations include cameras that monitor the panels, but it may be difficult to tell from videos or still images whether the panels are dirty (see, e.g., Figure 4 in Section 5). Thus, in practice, PV systems often operate in less than ideal conditions.

The problem we address in this paper is how to determine, in a data-driven fashion, what is wrong with a solar panel, on a per-panel rather than per-array basis. Since most large-scale PV systems are equipped with sensors that measure solar intensity and power output at regular intervals, we propose a simple classification approach to explain *anomalies* (i.e., drops) in the produced power based on these time series. This is a challenging problem because it is not obvious how to distinguish between different types of anomalies, and therefore it is not obvious which *features* of the data to use for classification.

We take a first step towards data-driven classification of anomalies in PV power output based on fine-grained per-panel data. Our solution exploits the physical properties of solar radiation. We observe that obstructions which do not touch the panels, such as shading, affect the power output in a subtly different way than dirt or snow lying on the panels. Based on this observation, we derive simple features from the power output time series that distinguish between shadows and soiling. We tested the proposed idea using data obtained from two real PV installations in the province of Ontario, Canada, and obtained 85 percent accuracy.

An obvious limitation of the proposed solution is that it can only tell shadows apart from direct cover, but it cannot distinguish between different types of direct cover (such as dust, dirt, or leaves) or between direct cover and physical panel malfunctions. Nevertheless, this simple classification can already be helpful to PV owners as it can suggest when the panels are due for a cleaning and when unexpected shadows arise. Our preliminary results are promising, and we hope that this paper encourages further research in solar panel data mining.

The remainder of this paper is organized as follows. Section 2 presents the necessary background in solar panel monitoring and defines our problem; Section 3 discusses related work; Section 4 presents our solution; Section 5 describes our experimental results; and Section 6 concludes the paper with directions for future work.

.

2. PRELIMINARIES AND PROBLEM STATEMENT

We begin with a simple example of the factors affecting the power output of a solar panel with the help of Figure 1. The curve labeled "1" corresponds to the maximum solar intensity times the surface area of the panel throughout a hypothetical day, on which the sun rises at 6:00 and sets at 20:00. If the sun were shining all day, there were no clouds, and the panel was able to convert 100 percent of the solar radiation into power, curve 1 would be the maximum power output throughout the day. Chapter 20 of [14] describes how to estimate the maximum clear-sky solar intensity given the time of day, day of year, latitude and tilt angle of the panel, all of which determine the relative position of the panel with respect to the sun.

PV systems usually include a *pyranometer* – a device that measures the solar intensity reaching the panels. The pyranometer is tilted at the same angle as the panels and is designed to stay clean and snow-free. The curve labeled "2" corresponds to the actual solar intensity times the surface area of the panel through the day. Drops in curve 2 compared to curve 1 indicate clouds, and in practice, curve 2 may be much more "noisy" than shown; see, e.g., Figure 2 and Figure 3.

Of course, a solar panel cannot convert all the radiation into power, i.e., its *efficiency* is not 100 percent. PV manufacturers typically specify efficiency as a function of temperature (solar panels tend to be more efficient at lower temperatures) [20]. Curves 3 and 4 in Figure 1 are derived by applying an efficiency formula to curves 1 and 2, respectively. That is, curve 3 is the expected power output given a perfectly sunny day, and curve 4 is the expected power output after taking clouds into account. Note that the area between curves 3 and 4 corresponds to power loss due to clouds, which is unavoidable.

There are two common ways to compute curve 4. One is to start with the solar intensity measured by a pyranometer, as described above, and adjust it according to the efficiency function. If there is no pyranometer onsite, another way is to select one panel as a reference panel and use its actual power output as the expected power output. Of course, this panel, to which we refer as a *reference panel*, must be clean and problem-free.

Finally, curve 5 shows the actual power output of the panel, as measured by a sensor connected to the micro-inverter. Ideally, curve 5 should be identical to curve 4. In Figure 1, the actual power output drops below the expected power output around 11:00, which could be due to external factors such as shadow or dirt. Note that the area between curves 4 and 5 corresponds to power loss due to such external factors, many of which are avoidable, e.g., by cleaning the panels.

We are now ready to state the problem we want to solve. We are monitoring a PV array consisting of multiple panels. We are given 1) enough enough information to compute the expected power output time series (curve 4), e.g., the corresponding solar intensity and temperature time series plus the performance ratio function, and 2) for *each* panel, we are given an actual power output time series (curve 5). Our goal is to identify and classify time intervals during which curve 5 significantly drops below curve 4, as we will formalize in Section 4. We assume that the input time series have a fine granularity (e.g., one measurement every 5 or 15 minutes). The frequency of identifying and classifying anomalies in the power output depends on the application; for concreteness, we assume that at the end of each day, we need to analyze the current day's data.

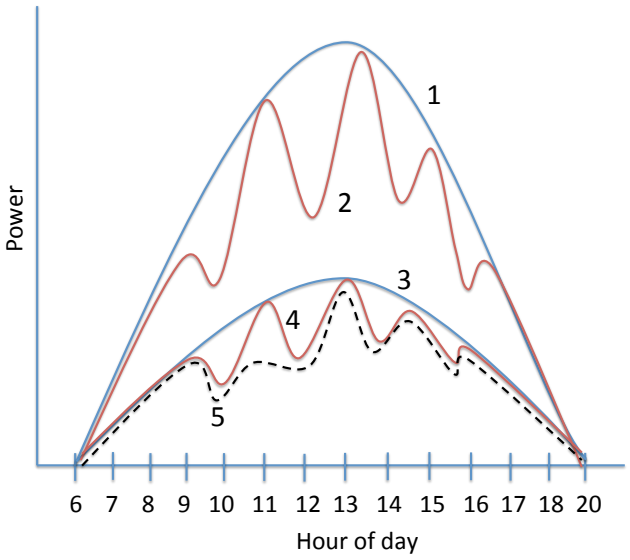


Figure 1: Example of solar panel output assuming perfect efficiency and a sunny day (1), perfect efficiency and clouds (2), actual efficiency without (3) and with (4) clouds, and with other factors (5).

3. RELATED WORK

There has been a great deal of research on understanding and attributing the power loss of a whole PV array due to weather and the external environment. Field trials and simulations were done to model and characterize the effects of cloud cover (see, e.g., [13]), air pollution (see, e.g., [11]), shadows (see, e.g., [4, 17, 23]), dust and dirt (see, e.g., [3, 6, 8, 15, 25]), and snow (see, e.g., [1, 2, 18]). The goal of this body of work was mainly to estimate the percentage power loss over an extended period of time, perhaps as a function of the type or thickness of snow or soiling. Rather than studying a particular factor in a controlled environment (e.g., using clean and dirty panels side-by-side), our work aims to infer the underlying factors based on (per-panel) power output and solar intensity data.

In terms of anomaly detection, there are at least three related approaches, which we summarize below.

The first approach, mentioned in [16, 19, 24], is to periodically compute linear regressions of power output vs. solar radiation and power output vs. panel temperature to detect changes in the behaviour of panels. However, this approach is not meant to distinguish between different types of changes, and therefore anomaly classification was not discussed.

In [5, 21], the solution is to collect statistics about anomalies such as the magnitude of the power drop and the duration of the anomaly. The idea behind our solution is similar, but we show that a single feature is already sufficient to distinguish between shadow and direct covering of a panel. Furthermore, our solution does not rely on the magnitude

of the power drop since the same type of anomaly (e.g., dirt/snow) may cause a different amount of power drop in different circumstances (e.g., different thickness and density of snow or different types of dirt).

The third approach is based on machine learning. In [12], a decision tree classifier was constructed to predict the severity of a physical problem with a solar panel based on features such as discolouration or panel warping. While we also aim to classify anomalies in power output, we focus on external problems rather than hardware faults, and therefore our framework and features are different. In [10], several classifiers were tested on their ability to classify anomalies in PV power output based on statistical properties of the output time series. While our solution also classifies anomalies in power output, it is different from [10] in several ways. First, we assume that we are also given solar intensity data as input, which allows us to separate power drop due to cloud cover from other factors. Second, as we will show, we use simple and interpretable features of the output time series rather than complex statistical properties.

There is also a variety of commercial software tools for estimating and tracking the power produced by solar panels, and estimating power loss due to weather and other factors; examples include Enphase Energy's Enlighten[1], Locus Energy's PVIQ[2], PVSyst[3] and Tigo[4]. Some systems use rough estimates for shading and soiling losses based on historical data, while others include more sophisticated analytics. For example, PVIQ estimates loss due to shading by identifying seasonal patterns, e.g., a drop in power every morning throughout the summer may correspond to a morning shadow. Our solution does not require a year of training data. In general, our solution is complementary to, and may be incorporated in, the above systems to improve the accuracy of power loss estimation and attribution.

4. OUR SOLUTION

Recall that we are given an expected power output time series, computed using pyranometer measurements or using the power output of a clean reference panel, and an actual power output time series. Our goal is to explain anomalies in the actual power output. Also, recall that the expected power output already accounts for clouds, so any further drop in produced power is likely due to other factors such as dirt or shadow. The crux of our solution is the observation that dirt or snow, which physically cover a panel, affect the power output in a different way than shadows. We illustrate this observation with an example and then we explain it in terms of the physical properties of solar radiation.

4.1 Intuition and Physical Explanation

Figure 2 plots the expected ("theoretical") and actual ("real") power outputs (in Watts) of the solar panel circled in red in Figure 5; we will describe the PV array this panel comes from in Section 5.1. The measurements were taken on February 11, 2012, and, as can be seen, this panel is covered by snow. In general, this panel is producing roughly one third of the expected power. Notice that the real power output follows the fluctuations of the expected power output; that is, if clouds come out, the power output drops correspondingly.

Next, in Figure 3 we show another pair of theoretical and real power time series for another panel covered by a morning shadow (from about 9:00 till 11:00) on July 10, 2013. Notice that at that time the power output drops to roughly 20 Watts and generally *does not* follow the fluctuations of the expected power output. That is, whether it is sunny or cloudy, this shaded panel is producing (roughly) uniformly low power.

In order to explain these observations, we need to understand the physical properties of solar radiation [14]. It has two main components: *direct* and *diffuse*. Direct radiation reaches the surface of the Earth in a straight line from the sun without any reflection or scatter by the atmosphere. Diffuse radiation is scattered by the atmosphere and arrives at the surface of the Earth from all directions. There is also a third component, *albedo* radiation, which is the radiation reflected from the ground, but its effect on solar panels is negligible compared to the other two. On a clear sunny day, most of the radiation is direct. On a cloudy winter day even half the radiation may be diffuse depending on location.

Now, it is important to understand that shadow only blocks direct radiation, which would normally reach a solar panel in a straight line from the sun; diffuse radiation is not affected since it arrives from all directions. This is why the power output in Figure 3 drops and remains roughly constant. The only radiation getting through is diffuse, and this does not fluctuate when clouds come out. The peaks in theoretical power output are due to more direct radiation hitting the panel when the sky is clear. On the other hand, covering the panel with dirt or snow blocks both direct and diffuse radiation. This is why the power output in Figure 5 is roughly a constant fraction of the expected power output at all times: depending on the thickness and density of the snow, some fraction of all the radiation is blocked.

This simple property of solar radiation has been mentioned by prior work on PV performance analysis [5, 13, 17, 25]. Our contribution in this paper is to turn this observation into a classification feature, as we explain below, and experimentally verify its accuracy on real data.

4.2 Anomaly Classification

We now translate the above observations into features that may be used in classification. At any point in time, we define the *Performance Ratio* (PR) of a solar panel as the ratio of actual power produced to the expected (theoretical) power. That is, PR is the ratio of curves 5 and 4 from Figure 1, or the ratio of the two curves shown in each of Figures 2 and 3. For example, if the expected power is 100 Watts but the produced power is 40 Watts, the PR at that point in time is 0.4.

Let S be a set of data points. The *Coefficient of Variation* (CV) of S is a standard statistical metric, defined as the ratio of the standard deviation of the data points to their mean. Now, note that the Coefficient of Variation of the Performance Ratio (CVPR) is low in Figure 2 but higher in Figure 3. This is the main idea of the proposed solution.

The input to our problem consists of the expected and actual power output time series for a given solar panel, as discussed earlier. In the first step, we identify time intervals in which the PR is below some threshold τ_{PR}. In the second step, we compute the CVPR for each such time interval. If

[1] http://enphase.com/enlighten/
[2] http://locusenergy.com/solutions/pviq-analytics/
[3] http://www.pvsyst.com
[4] http://www.tigoenergy.com/

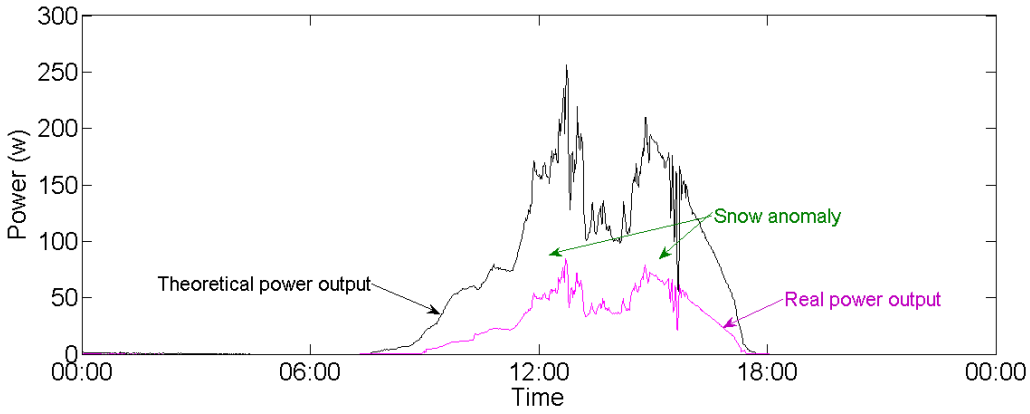


Figure 2: Expected and actual power output of a panel covered by snow.

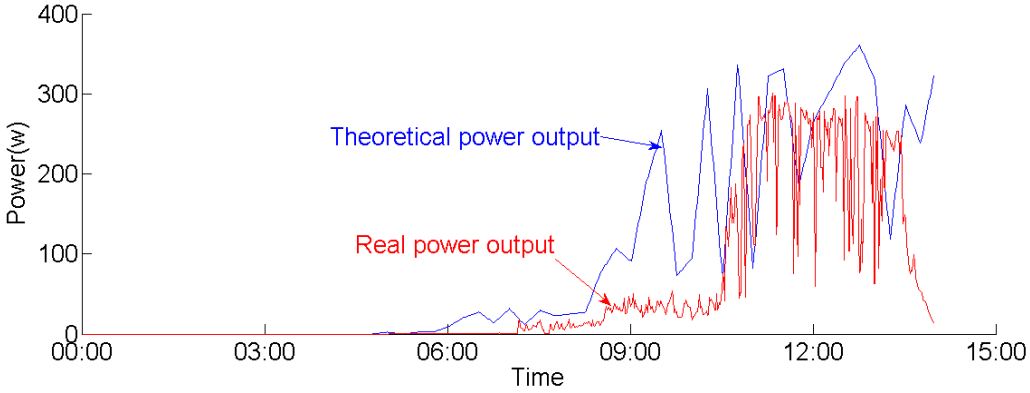


Figure 3: Expected and actual power output of a shaded panel.

the CVPR is below some threshold τ_{CVPR}, we classify the anomaly as direct cover. Otherwise, we classify the anomaly as a shadow. We reiterate that per-panel data are required for this method. Otherwise, if, say, only one panel is shaded, then the whole array's PR may still be very close to one and no anomaly will be detected.

The threshold τ_{PR} controls the aggressiveness of the above classification rule. A high value may lead to false positives, but a low value can miss some anomalies such as small shadows or delay the identification of anomalies such as dirt. The other threshold, τ_{CVPR}, can be learned from labeled data. We will discuss threshold selection further in Section 5.

We point out two simple optimizations of the above classification rule. First, after we find a time interval with low PR, rather than computing CVPR from all the points within this interval, we can remove outliers (highest and lowest PR values in the interval) and compute the CVPR from the remaining points. This will help guard against data errors. The second optimization is to only consider anomalies occurring when the solar intensity is sufficiently high. During periods of low intensity (e.g., dusk or dawn), there is little power being generated and the PR can be noisy.

Note that our solution can easily be extended. For example, in the context of a decision tree, we may test the value of CVPR in the root node of the tree, and then add further tests on other attributes of the data to further specify the cause of a power drop (e.g., dust vs. leaves on the panel vs. bird droppings). That said, we believe that classifying anomalies into shadow vs. direct-cover is already very useful as it can determine when the panels are dirty, for whatever reason, and need cleaning.

5. EXPERIMENTS

This section describes our experimental results regarding the accuracy of the proposed classification rule and the accuracy of other classification algorithms that may be applied to our problem, starting with a description of our two data sets, followed by our findings.

5.1 Data

In order to test an anomaly classifier, we need examples of shading and soiling along with the corresponding (expected and actual) power output time series. We obtained these from the following two PV installations.

TRCA: an array of 15 panels, three each from five different manufacturers, located in Toronto, Ontario. The panels are facing due south with a 30 degree tilt and are managed by the Toronto and Region Conservation Authority (TRCA). This data set contains power output, solar intensity (from an on-site pyranometer), temperature and wind-

Figure 4: Example of an image in the TRCA data set.

Figure 5: Example of an image showing snow-covered TRCA panels.

speed measurements every minute for one year, from December 2011 till December 2012. We calculated expected power output (curve 4 in Figure 1) from the solar intensity time series and the efficiency formulas provided by the PV manufacturers. Additionally, we obtained an image data feed containing 600x800 photos of the panels taken every 5 minutes. Due to low resolution, we could not identify dust or dirt; see, e.g., Figure 4 taken at noon on August 3, 2012. However, we found 24 days with snow; see, e.g., Figure 5 taken at noon on February 11, 2012.

UW: an array of 15 panels installed on the roof of one of the University of Waterloo buildings, facing 26.11 degrees southeast with a 15 degree tilt. We obtained access to the array for one month, from June 20 till July 20, 2013. There is no pyranometer onsite, so we selected one panel as a reference panel and ensured it is always clean and anomaly-free. The power output of this panel was used as the expected power output (i.e., curve 4 in Figure 1). Furthermore, there is no camera on-site, so we manually inspected the panels several times a day and recorded the times and locations of shadows. We also manually covered the panels with varying amounts of dirt (consisting of fine sand mixed with dried soil) and measured the corresponding power drop.

Table 1: PR and CVPR values of all 18 shadow anomalies

PR	CVPR
0.44	1.82
0.5	0.75
0.35	0.91
0.51	1.87
0.34	1.99
0.45	0.93
0.41	1.42
0.46	1.82
0.45	1.17
0.35	1.17
0.19	3.45
0.33	1.53
0.31	2.18
0.3	1.87
0.31	2.42
0.15	6.3
0.3	1.91
0.47	2.13

5.2 Results

Altogether we collected 60 examples of anomalies, 24 of which are due to snow (TRCA), 18 due to shadow (UW) and 18 due to dirt (UW). Tables 1, 2 and 3 list the PR and CVPR values for all the shadow, snow and dirt anomalies, respectively. Shadow appears to drop the power output to one-half or less of the expected output. The PR values for snow anomalies range from 0.1 to 0.88 depending on the thickness and density of the snow cover. Dirt appears to have less of an effect on the power output than other anomalies: the PR values for our dirt anomalies range from 0.85 to 0.97. However, this may be an artifact of our experimental procedure: the dirt we manually placed on the panels did not stick to the panels for very long and slid off them within several minutes (recall that the UW panels are tilted 15 degrees). In prior work, the effect of dirt and dust has been reported to be higher. Finally, we note that, as expected, the CVPR of shadow anomalies appears significantly higher than that of direct cover anomalies.

5.2.1 *Our Classifier*

We now test our simple classification rule: for each time interval in which PR drops below τ_{PR}, if CVPR is below τ_{CVPR}, the power drop is due to direct cover; otherwise, the power drop is due to shadow (then, separating direct cover into snow vs. other cover can be done easily with the help of weather data).

The first task is to determine a value for τ_{PR}. In general, we need to trade off between missed anomalies and false alarms. Our shadow and snow anomalies all had a PR under 0.88, but there were seven dirt anomalies with a PR above 0.9. However, as we mentioned earlier, in practice we expect dirt anomalies to have a lower PR than the PR we obtained in our experiments. Thus, $\tau_{PR} = 0.9$ is a reasonable choice. That is, we identify an anomaly if the actual power output of a panel is 90 percent or less of the expected output.

Table 2: PR and CVPR values of all 24 snow anomalies

PR	CVPR
0.48	0.17
0.5	0.18
0.45	0.74
0.56	0.37
0.58	0.6
0.1	1.44
0.55	0.31
0.77	0.11
0.42	0.48
0.11	1.8
0.88	0.02
0.47	0.44
0.62	0.18
0.62	0.34
0.35	0.46
0.76	0.15
0.74	0.08
0.84	0.05
0.85	0.05
0.78	0.19
0.81	0.09
0.81	0.17
0.23	0.67
0.37	0.63

Next, we need to choose a value for τ_{CVPR}. Based on our training data, the best thresholds are 0.75 and 1.17. With $\tau_{CVPR} = 0.75$, 50 out of 60 anomalies are classified correctly, with two snow and 8 dirt anomalies misclassified as shadow. With $\tau_{CVPR} = 1.17$, 51 out of 60 anomalies are classified correctly for an accuracy of 0.85, with 3 shadow anomalies misclassified as direct cover, and two snow and 4 dirt anomalies misclassified as shadow. As we mentioned in Section 4.2, there are simple optimizations that may improve accuracy, such as removing PR outliers within the time interval of an anomaly. Furthermore, having access to more labeled data should help choose a better threshold. That said, based on our results so far, we conclude that a τ_{CVPR} value of around one should work well.

We also point out that only three shadow anomalies had a CVPR value below one, and they happened on cloudy days, on which the solar radiation was not as noisy as that in Figures 2 and 3. As a result the CVPR was lower than it would be had there been periods of sunshine and clouds throughout the day. On the other hand, there are several snow and dirt anomalies with a relatively high CVPR between 1.4 and 1.8. These correspond to thin layers of dirt or snow, which may have allowed more diffuse radiation to reach the panel than a thick and dense cover would.

5.2.2 Other Classifiers

For comparison, we also tested several classifiers using the WEKA machine learning toolkit [9]. Each classifier was given two feature variables: PR and CVPR, and the class label, which could be shadow or direct cover. Table 4 shows the accuracy of the tested classifiers using ten-fold cross validation. The algorithms are: the C4.5 decision tree, the Best First (BF) decision tree, the Naive Bayes (NB) decision tree, the Functional Tree (FT), the Simple Cart decision tree algorithm, Support Vector Machines (SVM) with linear and degree-4 polynomial basis, and the k-Nearest-Neighbour algorithm with three different values of k.

Table 3: PR and CVPR values of all 18 dirt anomalies

PR	CVPR
0.9	0.7
0.91	1.2
0.88	1.13
0.92	0.68
0.82	0.57
0.95	0.55
0.97	0.41
0.94	0.43
0.9	0.13
0.93	1.41
0.9	0.98
0.86	1.45
0.88	0.97
0.85	1.56
0.93	1.04
0.9	0.66
0.89	0.53
0.9	0.89

Table 4: Accuracy of other classification algorithms

Classifier	Accuracy
C4.5	0.93
BF Tree	0.92
NB Tree	0.93
FT	0.86
Simple Cart	0.93
SVM (Linear)	0.88
SVM (degree 4 polynomial)	0.88
kNN (k = 1)	0.95
kNN (k = 3)	0.93
kNN (k = 5)	0.88

The accuracy of the other classifiers is higher than that of our simple rule, at the cost of over-fitting. For instance, the C4.5 algorithm gave the following tree, which overfits the data by making multiple tests on PR; the numbers in brackets correspond to the number of anomalies covered by each leaf node in the decision tree. Interestingly, PR, not CVPR, is tested at the root of the tree. However, as the tree shows, some direct cover anomalies have low PR whereas others have higher PR (depending on the thickness and density of the dirt or snow).

```
PR <= 0.51
|   CVPR <= 0.74: Direct Cover (8.0)
|   CVPR > 0.74
|   |   PR <= 0.11: Direct Cover (2.0)
|   |   PR > 0.11: Shadow (18.0)
PR > 0.51: Direct Cover (32.0)
```

Similarly, the BF tree also overfit the data by making multiple tests on PR and CVPR. The root node actually tests on CVPR but the threshold is too high and a second test on CVPR is required in the second layer of the tree.

```
CVPR < 1.81
|  PR < 0.525
|  |  CVPR < 0.745: Direct Cover (8.0)
|  |  CVPR >= 0.745
|  |  |  PR < 0.22: Direct Cover (2.0)
|  |  |  PR >= 0.22: Shadow (7.0)
|  PR >= 0.525: Direct Cover (32.0)
CVPR >= 1.81: Shadow (11.0)
```

Simple Cart also overfit the data with similar problems to that of the BF tree:

```
CVPR < 1.81
|  PR < 0.525
|  |  CVPR < 0.745: Direct Cover (8.0)
|  |  CVPR >= 0.745: Shadow (7.0)
|  PR >= 0.525: Direct Cover( 32.0)
CVPR >= 1.81: Shadow (11.0)
```

6. CONCLUSIONS AND FUTURE WORK

In this paper, we addressed the problem of identifying and explaining anomalies in the power output of solar panels. We developed and tested a simple classification rule based on the physical properties of solar radiation. The proposed rule can distinguish between power drop due to shadow and power drop due to direct cover such as dust or snow on the panel.

Based on our experimental results, there is room for improvement of our anomaly classifier, both in terms of accuracy and ability to further pinpoint the nature of a direct cover (dust, dirt, leaves, etc.). In general, given the rising popularity of solar panels and the availability of per-panel data, there is much more solar panel data mining that can be done. Examples include clustering the power output time series (and other measurements) to determine similar panels (in terms of performance and/or anomalies), outlier detection, and association rule mining among different panels (e.g., if there is a shadow on panel x then there will be a shadow on panel y within 15 minutes).

7. ACKNOWLEDGEMENTS

We would like to thank the Toronto and Region Conservation Authority (TRCA) for giving us a copy of their PV power output and image data, and we thank Bo Hu for setting up the power output monitoring infrastructure on the University of Waterloo PV array.

8. REFERENCES

[1] R. W. Andrews, A. Pollard and J. M. Pearce, The effects of snowfall on solar photovoltaic performance, Solar Energy 92 (2013): 84-97.

[2] G. Becker, B. Schiebelsberger, W. Weber, An approach to the impact of snow on the yield of grid-connected PV systems, in Proc. 21st European Photovoltaic Solar Energy Conference (EU PVSEC), 2006.

[3] J. R. Caron and B. Littmann, Direct Monitoring of Energy Lost Due to Soiling on First Solar Modules in California, IEEE Journal of Photovoltaics 3.1 (2013): 336-340.

[4] C. Deline, Partially shaded operation of a grid-tied PV system, in Proc. 34th IEEE Photovoltaic Specialists Conference (PVSC), 2009.

[5] A. Drews, A. C. De Keizer, H. G. Beyer, E. Lorenz, J. Betcke, W. Van Sark, W. Heydenreich, E. Wiemken, S. Stettler and P. Toggweiler, Monitoring and remote failure detection of grid-connected PV systems based on satellite observations, Solar Energy, 81.4 (2007): 548-564.

[6] M. S. El-Shobokshy and F. M. Hussein, Effect of dust with different physical properties on the performance of photovoltaic cells, Solar Energy 51 (1993): 505-511.

[7] D. Frankel, K. Ostrowski and D. Pinner, The disruptive potential of solar power, MicKinsey Quarterly, April 2014.

[8] D. Goossens and E. Van Keschaever, Aeolian dust deposition on photovoltaic solar cells: the effects of wind velocity and airborne dust concentration on cell performance, Solar Energy 66.4 (1999): 277-289.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The WEKA Data Mining Software: An Update, SIGKDD Explorations (2009) 11(1):10-18.

[10] B. Hu, Solar Panel Anomaly Detection and Classification, University of Waterloo M.Math Thesis, 2012.

[11] J. K. Kaldellis, P. Fragos and M. Kapsali. Systematic experimental study of the pollution deposition impact on the energy yield of photovoltaic installations, Renewable Energy 36.10 (2011): 2717-2724.

[12] J. M. Kuitche, R. Pan and G. TamizhMani, Investigation of Dominant Failure Mode(s) for Field-Aged Crystalline Silicon PV Modules Under Desert Climatic Conditions, IEEE Journal of Photovoltaics 4.3 (2014): 814-826.

[13] D. H. W. Li, G. H. W. Cheung and J. C. Lam, Analysis of the operational performance and efficiency characteristic for photovoltaic system in Hong Kong, Energy Conversion and Management 46 (2005): 1107-1118.

[14] A. Luque and S. Hegedus, Eds., Handbook of photovoltaic science and engineering. John Wiley & Sons, 2011.

[15] M. Mani and R. Pillai, Impact of dust on solar photovoltaic (PV) performance: research status, challenges and recommendations, Renewable and Sustainable Energy Reviews 14 (2010): 3124-3131.

[16] S. Mau and U. Jahn, Performance analysis of grid-connected PV systems, in Proc. 21st European Photovoltaic Solar Energy Conference (EU PVSEC), 2006.

[17] T. Oozeki, T. Izawa, K. Otani and K. Kurokawa, An evaluation method of PV systems, Solar Energy Materials and Solar Cells, 75.3 (2003):687-695.

[18] L. Powers, J. Newmiller and T. Townsend, Measuring and modelling the effect of snow on photovoltaic system performance, in Proc. 35th IEEE Photovoltaic Specialists Conference (PVSC), 2010.

[19] S. J. Ransome, J. H. Wohlgemuth, S. Poropat and E. Aguilar, Advanced analysis of PV system performance using normalised measurement data, in Proc. of 31st IEEE Photovoltaic Specialists Conference (PVSC),

2005.

[20] E. Skoplaki and J. A. Palyvos, On the temperature dependence of photovoltaic module electrical performance: A review of efficiency/power correlations, Solar energy 83.5 (2009): 614-624.

[21] S. Stettler, P. Toggweiler, E. Wiemken, W. Heydenreich, A. C. de Keizer, W. van Sark, S. Feige, M. Schneider, G. Heilscher and E. Lorenz, Failure detection routine for grid-connected PV systems as part of the PVSAT-2 project, in Proc. 20th European Photovoltaic Solar Energy Conference (EU PVSEC), 2005.

[22] I. H. Witten, E. Frank and M. A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, 3rd ed., Morgan Kaufmann Publishers, 2011.

[23] A. Woyte, J. Nils and R. Belmans, Partial shadowing of photovoltaic arrays with different system configurations: literature review and field test results. Solar Energy 74 (2003): 217-233.

[24] A. Woyte, M. Richter, D. Moser, S. Mau, N. Reich, U. Jahn, Monitoring of photovoltaic systems: good practices and systematic analysis, in Proc. 28th European Photovoltaic Solar Energy Conference (EU PVSEC), 2013.

[25] J. Zorrilla-Casanova, M. Piliougine, J. Carretero, P. Bernaola, P. Carpena, L. Mora-Lopez and M. Sidrach-de-Cardona, Analysis of dust losses in photovoltaic modules, World Renewable Energy Congress, 2011

What are the Most Important Research Challenges in Energy Data Management? (panel)

Torben Bach Pedersen
Aalborg University
tbp@cs.aau.dk

ABSTRACT

This panel paper aims at initiating discussion at the Fourth International Workshop on Energy Data Management (EnDM 2015) about what the most important research challenges within Energy Data Management are. The author is the panel organizer, extra panelists will be recruited from the workshop audience.

Keywords

Energy Data Management

1. RESEARCH CHALLENGES

The panel should try to answer (at least) the following questions:

- What are the research challenges within energy data management?
- What are their nature (scientific, technical, interdisciplinary,..) ?
- Which ones are the most interesting from a scientific point of view?
- Which ones are the most important from a societal point of view?

Below, some of the panel organizer's personal opinions on these questions are listed.

Research challenges within energy data management are abundant. Among the important scientific ones are a) the modeling and management of energy flexibilities, including more powerful flexibility models as well as scalable techniques for aggregating, scheduling, and disaggregating flexibilities; b) creating open and realistic benchmarks with associated open datasets; and c) development of robust and effective methods and techniques for predicting and forecasting energy consumption and production, as well as their associated flexibilities, at a very fine-grained level.

Technical research challenges include d) creating community-wide agreed-upon common definitions of data and information concepts, e.g., standardized ontologies specifying common concepts and e) the standardization of communication protocols, e.g., for communicating available flexibilities.

Interdisciplinary challenges, which are perhaps the most important from a societal point of view, include f) the interplay between hardcore data management techniques/tools and user-oriented human-computer interaction concepts to determine how and at which level of detail to interact with a smart grid system; and g) realizing the economic potential in energy data management systems by inventing, implementing, and taking to market new economics-based business models and energy taxation schemes that can ensure the (financial) interest, and thus the participaton, of all the many involved parties in solving the challenge of using very high rates of renewable energy in the grid. An example of such interdisciplinary collaborations is found in the Danish Totalflex project www.totalflex.dk.

2. PANEL ORGANIZER

Prof. Torben Bach Pedersen is full professor of computer science at Aalborg University, Denmark. He received his Ph.D. in 2000. His research interests span Big Data and business intelligence topics such as data warehousing, multidimensional databases, OLAP, and data mining, with a focus on non-traditional and complex types of data. He has published more than 140 peer-reviewed papers on these topics. He has served as PC Chair for DaWaK 2009+10, DOLAP 2010, and SSDBM 2014, General Chair for SSTD 2009, and on numerous program committees, including SIGMOD, (P)VLDB, ICDE, and EDBT. He has worked on energy data management since 2007, was involved in the MIRABEL EU FP7 project on energy data management, as is now leading the research in the large interdisciplinary Danish project, TotalFlex.

Event Processing, Forecasting and Decision-Making in the Big Data Era (EPForDM)

Alexander Artikis (NCSR Demokritos),
Antonios Deligiannakis (Technical University of Crete)

Challenges from Industrial Data Analytics

Michael May
Siemens AG, Germany

ABSTRACT

Big data applications in industry pose a number of unique challenges, setting them apart from domains such as consumer analytics in the web. Central for many industrial applications is time series data generated by often hundreds or thousands of sensors at a high rate, e.g. by a turbine. Another important data source are log files generated by control units in complex technical equipment, e.g. PLCs (programmable logic controller). This data can be used for failure statistics, root cause analysis, predictive maintenance, or for optimizing the performance during product design. Especially interesting are use cases that combine in-situ streaming analytics inside the local devices with centralized information, e.g. time series data collected from a whole fleet of wind turbines. In this talk I will describe a number of SiemensâĂŹ machine learning applications, especially failure diagnostics at the CERN Large Hadron Collider, self-optimizing wind turbines, and levee monitoring for Waternet Amsterdam. I will also discuss architectural challenges for such systems from a Big Data point of view.

Short Bio

Michael May is Head of the Technology Field Business Analytics & Monitoring at Siemens Research and Technology Center, and responsible for ten research groups in Munich, Vienna, Brasov, St. Petersburg, Princeton, and Berkeley. He is driving research at Siemens in data analytics and big data architectures and implements with his teams data analytics solutions across Siemens. Before joining Siemens in 2013, he was Head of the Knowledge Discovery Department at the Fraunhofer Institute for Intelligent Analysis and Information Systems in Bonn, Germany. In cooperation with industry he developed Big Data Analytics applications in sectors ranging from telecommunication, automotive, retail, logistics to finance and advertising. Michael was responsible for a number of National and European funded research projects in the area of Data Mining, Machine Learning, and Big Data. Between 2002 and 2009 he coordinated two Research Networks in Data Mining and Machine Learning at the European level, and he was local chair of ICML 2005. He did his PhD on machine discovery of causal relationships at the Graduate Programme for Cognitive Science at the University of Hamburg

Complex Event Recognition under Uncertainty: A Short Survey

Elias Alevizos[1], Anastasios Skarlatidis[1], Alexander Artikis[2 1], Georgios Paliouras[1]
[1]National Centre for Scientific Research (NCSR) "Demokritos", Greece
[2]University of Piraeus, Greece
{alevizos.elias, anskarl, a.artikis, paliourg}@iit.demokritos.gr

ABSTRACT

Complex Event Recognition (CER) applications exhibit various types of uncertainty, ranging from incomplete and erroneous data streams to imperfect complex event patterns. We review CER techniques that handle, to some extent, uncertainty. We examine both automata-based techniques, which are the most often, and logic-based ones, which are less frequently used. A number of limitations are identified with respect to the employed languages, their probabilistic models and their performance, as compared to the purely deterministic cases.

1. INTRODUCTION

Systems for Complex Event Recognition (CER) accept as input a stream of time-stamped simple, derived events (SDE)s. A SDE ('low-level event') is the result of applying a computational derivation process to some other event, such as an event coming from a sensor. Using SDEs as input, CER systems identify complex events (CE)s of interest—collections of events that satisfy some pattern. The 'definition' of a CE ('high-level event') imposes temporal and, possibly, atemporal constraints on its subevents, i.e. SDEs or other CEs. For example, consider the recognition of attacks on computer network nodes, given the TCP/IP messages. A CER system attempting to detect a DOS attack has to identify (as one possible scenario) both a forged IP address that fails to respond and that the rate of requests is unusually high.

Due to the complex nature of information sources, the input events arriving at a CER system almost always carry a certain degree of uncertainty and/or ambiguity. Sensor networks introduce uncertainty into the system due to reasons that range from inaccurate measurements through network local failures to unexpected interference of mediators. The latter is a new phenomenon that stems from the distribution of sensor sources. Sensor data may go through multiple mediators en route to the CER systems. Such mediators apply filtering and aggregation mechanisms, most of which are unknown to the system that receives the data. For example, a road sensor collecting traffic data may calculate the average speed of cars passing over it within a time period, but this calculation might not be accurate, it might be corrupted or it might even fail to reach the CER system, due to some network failure, unrelated to the sensor. Again, in the traffic management domain, it might not be possible to define all the possible situations which indicate the occurrence of an accident. Hence, the uncertainty that is inherent to sensor data is multiplied by the factor of unknown aggregation and filtering treatments [5]. Even if we assume perfectly accurate sensors, the domain under study might be difficult or impossible to model precisely, thereby leading to another type of uncertainty.

Until recently, most CER systems did not make any effort to handle uncertainty [9]. This need is gradually being acknowledged and it seems that this might constitute a significant line of research and development for CER. Almost all of the papers presented have appeared after 2008. The purpose of this paper is to present a short overview of existing approaches for performing CER under uncertainty. It should be noted that handling uncertainty in activity recognition (where SDEs come mainly from video streams or RFID tracks) is an active research field that has strong similarities with CER. However, in this short survey we have chosen to present only those methods that come directly from the field of CER.

The structure of the paper is as follows: In Section 2 we discuss the dimensions along which a proposed solution for handling uncertainty may be evaluated. Section 3 presents the reviewed approaches, summarizes them in a tabular form and comments on their limitations. Some open issues and lines of potential future work are identified in Section 4.

2. EVALUATION DIMENSIONS

We restrict attention to the following types of uncertainty. First, the rules defining a CE may be imperfect. Second, the SDE stream may be incomplete and/or include erroneous events. Detailed discussions about types and sources of uncertainty in CER may be found in [4, 22].

We follow the customary division between representation, inference and learning. In other words, we are interested in what kind of knowledge a system can encode (representation), what kind of queries it can answer (inference) and if/what parameters and models it can learn. However, although learning in general is a very active research area, we have decided not to include a detailed discussion about the learning capabilities of the examined approaches in our sur-

vey. The reason is quite simple. Almost none of the systems touches upon this subject. Instead, we draw some conclusions as far as the performance of each system is concerned.

2.1 Representation

Following the terminology of [15], we define an event as an object in the form of a tuple of data components, signifying an activity and holding certain relationships to other events by time, causality and aggregation. An event with N attributes can be represented as

$$E(Type, ID, Attribute1, \ldots, AttributeN, Time)$$

where $Time$ might be a point, in case of an instantaneous event, or an interval during which the event happens, if it is durative. In CER, we are interested in detecting patterns of events among the streams of SDEs. Therefore, we need a language for expressing such pattern detection rules.

Formalisms for reasoning about events and time have appeared in the past, such as the Event Calculus [6, 14] and Allen's Interval Algebra [2,3], and have already been used for defining event algebras (e.g. in [18]). With the help of the theory of descriptive complexity, recent work has also identified those constructs of an event algebra which strike a balance between expressive power and complexity [27]. Based on the capabilities of existing CER systems and on related theoretical work, the following list enumerates those operations that should be supported by a CER engine:

- *Sequence*: Two events following each other in time.
- *Disjunction*: Either of two events occurring, regardless of temporal relations. Conjunction (both events occurring) may be expressed by combining *Sequence* and *Disjunction*.
- *Iteration*: An event occurring N times in sequence, where $N \geq 0$.
- *Negation*: Event not occurring at all.
- *Selection*: Select those events whose attributes satisfy a set of predicates/relations, temporal or otherwise.
- *Projection*: Return an event whose attribute values are a possibly transformed subset of the attribute values of its sub-events.
- *Windowing*: Apply pattern for events within a specified time window.

In a probabilistic setting, uncertain events are assigned an occurrence probability. More complex models also allow for probabilities on the attributes of the events as well. Furthermore, the rules for expressing CE definitions may also be probabilistic. The semantics for the probability space are usually those of possible worlds. A possible world is one of the possible SDE streams, as defined by the SDE probabilities. Thus, the probability space is understood as the set of all the alternative event streams that may have occurred and the distribution is defined over this set. Event attributes are usually discrete and the continuous case is outside the scope of most CER systems.

2.2 Inference

In probabilistic CER, the most basic inference task is to compute the probability of occurrence of a CE. In other words, the task is to compute the marginal probabilities of the CEs, given the SDEs. In some settings, we might also be interested in performing maximum a posteriori (MAP) inference, in which the task is to compute the most probable states of some CEs, given the evidence SDEs stream. A simple example from the domain of video recognition is the query in which the user asks about the most probable time interval during which a certain activity occurs.

Another dimension concerns the ability of a system to perform approximate inference. In the literature of statistical relational learning, it is widely believed that for all but the simplest cases, exact inference stumbles upon serious performance issues, unless several simplifying assumptions are made. For this reason, approximate inference is considered essential. When this capability is present, certain systems provide answers with confidence intervals and/or the option of setting a confidence threshold above which an answer may be accepted.

2.3 Performance

CER systems are usually evaluated for their performance in terms of throughput, measured as number of events processed per second. For some queries, the latency, as measured by the time required to process an event, is also important. Less often, the memory footprint is reported. Note that no standard benchmarks exist, although some work towards this direction has begun [12, 16, 17]. Reporting throughput figures is not enough by itself, since there are multiple factors which can affect performance, such as query selectivity (see [16] for a list of such factors). When uncertainty is introduced, the complexity of the problem grows and other performance-affecting factors enter the picture, such as the option of approximate inference. Moreover, systems need to be evaluated along another dimension, that of accuracy.

The issue of accuracy is of critical importance and is not orthogonal to that of performance. Precision and recall are the usual measures of accuracy, but neither one of them may be sufficient by itself. Therefore, a more appropriate measure would be that of the F-measure, i.e. the harmonic mean of precision and recall.

3. APPROACHES

Since many of the CER engines employ finite automata, either deterministic (DFA) or non-deterministic (NFA), it is not surprising that automata are one of the dominant approaches for handling uncertainty. Less frequently, logic-based approaches are preferred. In this section, we present both of these areas.

We summarize our results in Tables 1 - 3. The columns of Table 1 correspond to the list of operators presented in Section 2.1 and refer to the expressive power of the language employed. An extra column has been added to indicate whether a system supports event hierarchies, i.e. the ability to define CEs at various levels and reuse those intermediate inferred events in order to infer other higher-level events. In Table 2 we present the probabilistic properties of each method, with respect to the independence assumptions they make and to their capacity for assigning probabilities to the input data (SDEs) and/or the rules for CE definitions. Some systems

Language Expressivity										
Paper	σ	π	$\wedge$	$\vee$	$\neg$	;	*	W	H	Remarks
Kawashima et al [13]	✓	✓				✓	✓	✓		
Re et al [19]	✓					✓	✓			
Chuanfei et al [7]					✓	✓	✓	✓		Not enough details in paper about σ, π, $\wedge$, $\vee$, $\neg$.
Shen et al [20]	✓	✓				✓	✓	✓		
Wang et al [21]	✓		✓	✓	✓	✓		✓	✓	
Zhang et al [26, 27]	✓	✓	✓	✓	✓	✓	✓	✓		
Cugola et al [10]	✓	✓	✓		✓	✓		✓	✓	* implicit; Support for continuous event attributes.
Wasserkrug et al [23, 24, 25]	✓	✓	✓			✓			✓	Explicit time representation

Table 1: Expressive power of CER systems. Columns: σ: selection, π: projection, $\wedge$: conjunction, $\vee$: disjunction, $\neg$: negation, ;: sequence, *: iteration, W:windowing, H: hierarchies.

Probabilistic Expressivity				
Paper	Data (occurrence and/or attributes)	Rules	Independence Assumptions	Remarks
Kawashima et al [13]	Occurrence		All events independent	
Re et al [19]	Both		1st-order Markov for SDEs (different streams independent)	
Chuanfei et al [7]	Occurrence		1st-order Markov with extensions	
Shen et al [20]	Both		SDEs independent	
Wang et al [21]	Occurrence		SDEs independent or Markovian (different streams independent).	
Zhang et al [26, 27]	Occurrence		SDEs independent	Probability distribution on time attribute
Cugola et al [10]	Both	✓	Event attributes independent. SDEs independent. CEs dependent only on events immediately below in hierarchy.	Bayesian Networks
Wasserkrug et al [23, 24, 25]	Both	✓	SDEs independent	Bayesian Networks

Table 2: Expressive power of CER systems with respect to their probabilistic properties.

Inference					
Paper	Marginal / MAP	Confidence Thresholds	Approximate	Performance	Remarks
Kawashima et al [13]	Marginal	✓		0.8-1.1 K events/s with *Kleene+*	
Re et al [19]	Marginal	✓		> 10 points increase in accuracy. 100K tuples/s for Extended Regular Queries.	
Chuanfei et al [7]	Marginal	✓		4-8K events/s for pattern lengths 6-2	
Shen et al [20]	Marginal	✓		1000K events/s, almost constant for varying window size. 1000K-100K events/s for 10-1 alternatives.	
Wang et al [21]	Marginal	✓		8K-13K events/s for 2-6 nodes	Distributed
Zhang et al [26, 27]	Marginal	✓		Reduction from exponential to close-linear cost w.r.t to selectivity / window size	
Cugola et al [10]	Marginal	✓		50% overhead	
Wasserkrug et al [23, 24, 25]	Marginal	✓	✓	CEs within desired confidence interval. Sub-linear decay of event rate w.r.t possible worlds.	

Table 3: Inference capabilities of probabilistic CER systems

may allow only uncertainty with respect to the occurrence of an event, whereas others may allow uncertainty for the event attributes as well. Finally, Table 3 presents some of the systems' properties when performing inference, such as whether they perform marginal or MAP inference, whether they give the user the option to set minimum confidence thresholds and whether they can perform approximate inference. Some comments about their performance are also included.

3.1 Cayuga

The Lahar system of Re et al [19] constitutes one of the earliest proposals. It is based on the Cayuga [11] CER engine. The design goal behind the Lahar system is to develop an efficient inference mechanism for answering queries over probabilistic SDE streams, i.e. streams whose events are tagged with a probability value. It is assumed that events follow a first-order Markov process. The possible queries are categorized in three different classes. Regular queries are composed of subgoals which do not share any variables, can readily be transformed into regular expressions with a corresponding automaton and can be evaluated in time linear to the size of the event stream. Extended regular queries allow for shared variables which must be present in all of the subgoals. Therefore, the query can be broken into independent, regular "ground" queries (by substitution) and its success probability can be computed by combining the probabilities of its constituent "ground" queries. Finally, in safe queries, variables might not be shared among all subgoals. These queries are evaluated by using a version of the Probabilistic Relational Algebra with a complexity that is quadratic to the number of timestamps in the SDE stream. Lahar was tested on object tracking in which persons and objects were equipped with RFID tags and the persons' paths and/or locations had to be assessed. Significant improvements in precision and recall were observed against deterministic approaches, with only a relatively slight overhead on throughput, which reached hundreds of thousands of events per second. A method which attempts to overcome the strict markovian hypothesis and to apply certain optimizations, such as early pruning, may be found in [7].

3.2 SASE

A simple solution for handling uncertainty with automata was proposed by Kawashima et al [13], as an extension of the SASE+ event processing engine [1]. The system builds a deterministic automaton for every user query (CE definition) and detects patterns above a certain confidence threshold by developing a matching tree as new SDEs arrive until the time window of the query expires. Branches of the tree below the given threshold are pruned early for optimization purposes. The SDEs are assumed to be independent (therefore, probability values are calculated by multiplication) and are tagged with an occurrence probability. Neither probability values for the event attributes are allowed nor for the queries themselves. Throughput values can reach several hundreds of events per second, but these numbers correspond to experiments with a single query of low complexity – a sequence operator with equality selection on the attributes and no shared variables.

Another early, NFA-based approach to incorporate uncertainty within an existing CER system is presented in [20] by Shen et al. This work uses SASE+ as its starting point and amends it in order to handle probabilistic SDEs. Each SDE is defined as a set of alternatives, each with its occurrence probability, with all alternatives summing to a probability value of 1 or less than 1 if non-occurrence is considered. The probability space is therefore defined over the possible worlds, as determined by the different (mutually exclusive) alternatives of the SDEs. The CE definitions are encoded as NFAs, but, in order to avoid enumerating all possible worlds, a special data structure, called Active Instance Graph, is used. The Active Instance Graph is a Directed Acyclic Graph connecting events with previous candidate events, i.e. whose possible occurrence may lead to the recognition of the CE. By backward-traversing the AIG, the sequence(s) that satisfy the CE definition may be retrieved and this structure also allows for dynamic filtering of events when other constraints (besides temporal sequence) are present. Finally, each event is associated with its lineage, i.e. a function which captures "where the event came from", used for computing its probability.

Inspired yet again by SASE, the work recently proposed by Wang et al [21] attempts to address two important issues. The first, related to previous NFA-based methods, concerns their inability to express CE hierarchies. The second is a performance issue and, this work is the first one which develops a CER system which is both probabilistic and distributed. The CE recognition process depends on a data structure, called Active Instance Stack, which is an optimized version of the already mentioned Active Instance Graph. Probabilities may be assigned only to events and refer to occurrences (neither probabilities for CE definitions nor for event attributes are allowed). Events are also assumed to be either independent or to follow a first-order Markov process. A data partitioning scheme is used in order to distribute different parts of the streams to different nodes and the local results are later combined to produce a global result. Finally, CE hierarchies may be constructed by having different event processing agents producing different CE types and connecting them through channels (agents are pattern matching components which can be connected to form an event processing network).

In most of the automata-based methods (with the exception of [10], presented in Section 3.3), uncertainty concerns the occurrence of the event itself as a whole, but the event attributes, including timestamps, are certain. In Zhang et al [26], the issue of imprecise timestamps is addressed, while all the other attributes have crisp values. Due to sensors' sensitivity or time granularity differences between event sources, timestamps are assumed to follow a probability distribution (usually uniform). Each event may thus have several alternative occurrence timestamps and many possible worlds, i.e. event histories, are available to the system. The temporal relations between events may differ among the possible worlds and a CE recognized in one of them may not be recognized in another. One solution is to enforce an ordering of the events from all possible worlds and then leverage an existing CER engine, such as SASE, for the CE recognition task. However, the authors present another, more efficient method, which avoids a complete enumeration of all possible worlds by employing an incremental, three-pass algorithm through the events in order to construct event matches and their intervals. This method achieves high throughput but supports only *sequence* patterns with simple equality/inequality predicates. Moreover, it was extended in [27] by Zhang et al, which added *negation* and *Kleene plus* and allowed for user-defined predicates.

3.3 CEP2U

A more recent effort extends the TESLA [8] event specification language with probabilistic modelling, in order to handle the uncertainty both in input SDEs and in the definitions of CEs [10]. The semantics of the TESLA language are formally specified by using a first order logical representation with temporal constraints that express the length of time intervals. The CE recognition algorithm however employs automata. At the input level, the method supports uncertainty regarding the occurrence of the SDEs, as well as uncertainty regarding their content. In the former case, SDEs are associated with probabilities that indicate a degree of confidence. In the latter case, the attributes of an event are modelled as random variables with some measurement error. The probability distribution function of the measurement error is assumed to be known (e.g. Gaussian distribution). Since uncertainty also derives from incomplete or erroneous assumptions about the environment in which the system operates, the method also models the uncertainty of the CE definitions. In particular, the method automatically builds a Bayesian network for each rule. The probabilistic parameters of the network are manually estimated by domain experts.

3.4 Logic-based methods

Wasserkrug et al [23,24,25] employ the technique of knowledge based model construction (KBMC), whereby knowledge representation is separated from the inference process. Inference is preformed on a Bayesian network as needed (when new SDEs arrive), without constructing the whole network beforehand. Each event is assigned a probability, denoting how probable it is that the event occurred with specific values for its attributes. Uncertainty about the value of a single event attribute may be represented by multiple event instances with different probabilities and with the same values for all other attributes.

In turn, CE definitions are encoded in a two-fold way, with a selection operation (mostly based on event type) performing an initial filtering, followed by a pattern-detection schema for more complex operations, based on temporal relations and attribute equalities. The selection mechanism imposes certain independence properties on the Bayesian network. Inferred CEs are conditioned only on selectable lower-level events, preventing the network from being cluttered with many dependency edges. This framework is not limited to representing only propositional or first order knowledge. It could potentially handle higher-order knowledge, since this pattern-matching step could, in principle, be defined in any kind of language. However, the system presented in the evaluation experiments allows only predicates expressing temporal constraints on event timestamps or equality relations on event attributes.

Calculation of the probabilities for the inferred CEs is done by dynamically constructing a Bayesian network upon every new event arrival. The nodes of the network correspond to SDEs and CEs. First, SDEs are added. Nodes for CEs are inserted only when a rule defining the CE is sat-

isfied, having as parents the events that triggered the rule, which might be SDEs or even other CEs, in case of hierarchical CE definitions. The probability and attribute values of the inferred CEs are determined by mapping expressions associated with the corresponding rule. In order to avoid the cost of exact inference, a form of sampling is followed, which allows for bypassing the construction of the network by sampling directly according to the rules for CE definitions.

3.5 Comments

In Table 1 we list the operators supported by each method. Table 2 presents their probabilistic properties: their independence assumptions and the support for data and/or rules uncertainty. Their properties with respect to inference are shown in Table 3 (marginal/MAP inference, support for confidence thresholds, approximate inference).

As shown in Table 2, all of the presented approaches have the ability to represent probabilistic SDEs, where uncertainty may refer to their occurrence or/and the content of their attributes. However, a feature which is lacking in most of the methods is the capacity to assign probabilities to rules expressing CE definitions. In this case, probabilistic graphical models, with their ability to represent all events as nodes in a homogeneous manner and encode the direction of causation, can prove useful. The two methods which allow rule probabilities, use such a model, namely Bayesian Networks.

The KBMC method of [23, 24, 25] and the CEP2U system of [10] allow for both hierarchies and probabilistic rules (see Table 2). Both of them use Bayesian Networks for inference, with the nodes of the network representing events, SDEs and CEs. CEP2U was designed from the very beginning with the goal of minimizing the performance overhead incurred by the introduction of uncertainty. Indeed, the maximum overhead mentioned in the experiments was almost always less than 50%, compared to the deterministic case. On the other hand, the KBMC technique is still far from achieving event rates comparable (say, within an order of magnitude) to those of purely deterministic models. This performance robustness of CEP2U against uncertainty comes at a price though, since some simplifying assumptions have to be made. CEP2U constructs only a single Bayesian Network for each rule (not for each grounding) and a simple solution is proposed for the problem of propagating probabilities from lower to higher level CEs. Occurrence probabilities of intermediate events are propagated to higher level events with a value of 1, essentially decomposing the total probability space into smaller and more manageable spaces. This means that these Bayesian Networks function more like look-up tables, hence the much lower cost of inference. The effects of this simplification on accuracy, however, are unclear.

A related issue is that of the independence assumptions made by each method. Automata-based methods tend to make a substantial number of simplifying assumptions about the independence of events or streams, resulting in simpler probabilistic models The most complex dependency models employed make the assumption that events may follow a first-order Markov process, as in [19, 21] (a slightly more complex model may be found in [7]). In domains characterized mostly by sequential patterns upon homogeneous streams, this assumption may be sufficient. When multiple streams with different event types are involved and hierarchies of CEs are required, which take into account lower-level CEs across a time window, more complex dependencies need to be encoded.

Bayesian Networks offer such a flexibility but they suffer from problems of high inference complexity. In order to keep the inference cost low, certain simplifications are introduced again. For example, CEP2U assumes that an inferred CE is the only cause for all of its sub-events (note that, in CEP2U, the direction of causation is from the higher level to the lower level events), i.e. one sub-event cannot be used to define other CEs and it is not possible to have multiple definitions for a CE. Although this obviously helps in making the Bayesian Networks (which can be manually edited by the user) the assumption of such a strict separation of rule conditions limits the expressive power of the system (and would presumably require tedious tuning to correct it).

4. CONCLUSIONS

Our short review of probabilistic CER systems identified the following limitations: In terms of language expressivity, the basic drawback of most systems is the absence of support for constructing hierarchies of CEs. Moreover, most systems do not support uncertainty in the rules defining CEs. Those that do support rule uncertainty either make too strong simplifying assumptions, thus possibly limiting accuracy in domains with complex dependencies, or face serious issues of under-performance, even when approximate inference is employed. Distributed processing of probabilistic SDE streams is still at its early stages, with only one method employing it. Notice also that none of the systems supports MAP inference, a feature which is useful in certain domains (e.g. in video recognition, where it is sometimes desirable to retrieve those time intervals during which it is most likely for an activity to have occurred). Those issues should act as indicators for possible directions of future work.

5. ACKNOWLEDGMENTS

This work has been funded by the EU SPEEDD project (FP7-ICT 619435).

6. REFERENCES

[1] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD Conference*, pages 147–160, 2008.

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[3] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, July 1984.

[4] A. Artikis, O. Etzion, Z. Feldman, and F. Fournier. Event processing under uncertainty. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 32–43, New York, NY, USA, 2012. ACM.

[5] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, V. Kalogeraki, J. Marecek, and others. Heterogeneous stream processing and crowdsourcing for urban traffic management. In *EDBT*, pages 712–723, 2014.

[6] I. Cervesato and A. Montanari. A calculus of macro-events: progress report. In *Seventh International Workshop on Temporal Representation and Reasoning, 2000. TIME 2000. Proceedings*, pages 47–58, 2000.

[7] X. Chuanfei, L. Shukuan, W. Lei, and Q. Jianzhong. Complex event detection in probabilistic stream. In *Web Conference (APWEB), 2010 12th International Asia-Pacific*, pages 361–363, Apr. 2010.

[8] G. Cugola and A. Margara. TESLA: a formally defined event specification language. In *Proceedings of Conference on Distributed-Event Based Systems (DEBS)*, pages 50–61, 2010.

[9] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44(3), 2011.

[10] G. Cugola, A. Margara, M. Matteucci, and G. Tamburrelli. Introducing uncertainty in complex event processing: model, implementation, and validation. *Computing*, pages 1–42, 2014.

[11] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White. Towards expressive publish/subscribe systems. In Y. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, editors, *Advances in Database Technology - EDBT 2006*, number 3896 in Lecture Notes in Computer Science, pages 627–644. Springer Berlin Heidelberg, Jan. 2006.

[12] T. Grabs and M. Lu. Measuring performance of complex event processing systems. In *Topics in Performance Evaluation, Measurement and Characterization*, pages 83–96. Springer, 2012.

[13] H. Kawashima, H. Kitagawa, and X. Li. Complex event processing over uncertain data streams. In *2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 521–526, Nov. 2010.

[14] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[15] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[16] M. R. Mendes, P. Bizarro, and P. Marques. A performance study of event processing systems. In *Performance Evaluation and Benchmarking*, pages 221–236. Springer, 2009.

[17] M. R. Mendes, P. Bizarro, and P. Marques. Towards a standard event processing benchmark. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 307–310, New York, NY, USA, 2013. ACM.

[18] A. Paschke and M. Bichler. Knowledge representation concepts for automated SLA management. *Decision Support Systems*, 46(1):187–205, Dec. 2008.

[19] C. Re, J. Letchner, M. Balazinksa, and D. Suciu. Event queries on correlated probabilistic streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 715–728, New York, NY, USA, 2008. ACM.

[20] Z. Shen, H. Kawashima, and H. Kitagawa. Probabilistic event stream processing with lineage. In *Proc. of Data Engineering Workshop*, 2008.

[21] Y. H. Wang, K. Cao, and X. M. Zhang. Complex event processing over distributed probabilistic event streams. *Computers & Mathematics with Applications*, 66(10):1808–1821, Dec. 2013.

[22] S. Wasserkrug, A. Gal, and O. Etzion. A taxonomy and representation of sources of uncertainty in active systems. In O. Etzion, T. Kuflik, and A. Motro, editors, *Next Generation Information Technologies and Systems*, number 4032 in Lecture Notes in Computer Science, pages 174–185. Springer Berlin Heidelberg, Jan. 2006.

[23] S. Wasserkrug, A. Gal, and O. Etzion. A model for reasoning with uncertain rules in event composition systems. *arXiv:1207.1427 [cs]*, July 2012.

[24] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin. Complex event processing over uncertain data. In *Proceedings of the second international conference on Distributed event-based systems*, pages 253–264. ACM, 2008.

[25] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin. Efficient processing of uncertain events in rule-based systems. *IEEE Transactions on Knowledge and Data Engineering*, 24(1):45–58, Jan. 2012.

[26] H. Zhang, Y. Diao, and N. Immerman. Recognizing patterns in streams with imprecise timestamps. *Proc. VLDB Endow.*, 3(1-2):244–255, Sept. 2010.

[27] H. Zhang, Y. Diao, and N. Immerman. On complexity and optimization of expensive queries in complex event processing. pages 217–228. ACM Press, 2014.

Extending Event-Driven Architecture for Proactive Systems

Fabiana Fournier
IBM Research – Haifa
Haifa University Campus
Haifa 3498825, Israel
+972 4 8296489
fabiana@il.ibm.com

Alexander Kofman
IBM Research – Haifa
Haifa University Campus
Haifa 3498825, Israel
+972 4 8281055
kofman@il.ibm.com

Inna Skarbovsky
IBM Research – Haifa
Haifa University Campus
Haifa 3498825, Israel
+972 4 8281330
inna@il.ibm.com

Anastasios Skarlatidis
Institute of Informatics and Telecommunications, NCSR "Demokritos"
Athens 15310, Greece
+30 210 6503217
anskarl@iit.demokritos.gr

ABSTRACT

Proactive Event-Driven Computing is a new paradigm, in which a decision is not made due to explicit users' requests nor is it made as a response to past events. Rather, the decision is autonomously triggered by forecasting future states. Proactive event-driven computing requires a departure from current event-driven architectures to ones capable of handling uncertainty and future events, and real-time decision making. We present a proactive event-driven architecture for Scalable Proactive Event-Driven Decision-making (SPEEDD), which combines these capabilities. The proposed architecture is composed of three main components: complex event processing, real-time decision making, and visualization. This architecture is instantiated by a real use case from the traffic management domain. In the future, the results of actual implementations of the use case will help us revise and refine the proposed architecture.

Categories and Subject Descriptors

C.0 [Computer Systems Organization]: General – *System architectures*; D.4.8 [**Operating Systems**]: Performance - *Modeling and prediction*; G.3 **[Mathematics of Computing**]: Probability and Statistics - *Distribution functions*, *Time series analysis*; H.1.2 [**Models and Principles**]: User/Machine Systems – Human factors; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving - *Uncertainty, fuzzy, and probabilistic reasoning*.

General Terms

Performance, Design, Human Factors

Keywords

Proactive computing, event-driven, real-time optimization, forecasting, uncertain and future events, visualization.

1. INTRODUCTION

Proactive Event-Driven Computing is a new paradigm ([6],[7], [9]), where a decision is neither made due to explicit users' requests nor as a response to past events, but is autonomously triggered by forecasting future states, either desired or undesired. The decisions and actions are often real-time in the sense that they are done under time constraints and require the exploitation of large amounts of historical and streaming data. The underlying motivation of proactive computing stems from social and economic factors, and is based on the fact that *prevention* is often more effective than *cure*.

Achieving this vision requires novel research in three different directions:

Dealing with large quantities of data. Massive volumes of historical data and massive streaming data have to be analyzed to forecast events. Most systems are not capable of handling big data in real-time because of scalability problems, the need to cleanse noisy data offline, or the difficulty in fusing different types of data coming from different sources online. The result is that most analyses are done on offline data, while online data is not leveraged for immediate operational decisions.

Extending the state-of-the-art in event processing to deal with future events and uncertainty due to incomplete and noisy streaming data [1]. The ability to process past events and forecast future ones makes proactive systems a compelling application area. But, the uncertain nature of future events requires a major leap in event processing systems.

Devising methods for making near-optimal decision within time constraints. The decision about which is the best action to take in proactive computing has two properties that differ from most contemporary decision support systems: (1) the decision should be taken on-line and under real-time constraints, which may dictate the use of approximation techniques and (2) The decision often entails autonomic actions, rather than providing only recommendations for human decision makers.

A proactive-driven architecture should satisfy the requirements above and provide an integrated platform that combines advanced event processing with dynamic forecasting capabilities leveraged towards online optimisation and decision-making. The proposed architecture presented in this paper, an outcome of the SPEEDD (Scalable ProactivE Event-Driven Decision making) project[1], exactly addresses this.

This paper is organized as follows: Section 2 briefly introduces the traffic management use case that will illustrate our proposed architecture. Section 3 presents a general overview of a proactive event-driven architecture, while Section 4 details the SPEEDD proactive event-driven architecture. We survey some related work in Section 5. We conclude the paper in Section 6.

2. ILLUSTRATIVE EXAMPLE

Proactive traffic management concerns the south ring of Grenoble, which is the main West to East artery around the city in France and a primary source for traffic congestion. The goal within this use case is to forecast traffic congestion before it

[1] http://speedd-project.eu

happens and, as a result, automatically act in order to attenuate it. This is done by forecasting traffic congestions a few minutes before they happen, and making decisions within a few seconds of the forecast about adjustment of traffic light settings and speed limits.

There are two sources of data in this use case: real data from sensors and synthetic data generated by a micro-simulator.

The input data (raw events) comes from 130 magnetic wireless Sensys sensors[2] buried in the road along the highway which can provide individual or aggregated data. Sensors are located in 19 collection points. Each collection point has a sensor per lane (slow and fast lane) and, where applicable, also has sensors on the on/off-ramps. Sensors provide data every 15 seconds. Such data can be either individual (concerning every single vehicle), or aggregated (over the 15-seconds time span). However, the individual and aggregated data cannot be collected simultaneously. Currently, aggregated data is being collected.

The simulator used for generating synthetic traffic data is the commercial micro-simulator by Aimsun[3]. The simulator has been calibrated using real traffic data from Grenoble South Ring.

3. PROACTIVE EVENT-DRIVEN ARCHITECTURE

Conceptually, we distinguish between the design time and runtime components.

At the *build or design time*, proactive applications are developed using *authoring tools* either directly by *experts* or with the help of *learning systems*. *Visualization tools* can be used to analyze the stored historical data during design time. By using the *authoring* and *visualization tools*, the experts may also annotate the historical data, in order to provide training examples for the machine learning algorithms. The products of the design time activities are event processing definitions and decision making configurations that will be deployed and executed at the runtime.

The *runtime* consists of four building blocks or components: event processing, forecasting, real-time decision making, and visualization tools. In general, raw events emitted by various event sources (e.g., traffic sensors) are processed by the complex event processing (CEP) engine and forecasted events serve for real-time decision making. The CEP engine processes raw as well as derived (detected and forecasted) events to detect and forecast higher-level events, or situations. These serve as triggers for the decision making component, which uses domain-specific algorithms to suggest the next best action to resolve or prevent an undesired situation.

Let's examine in more details the principles of each building block in the envisaged architecture:

The first building block required to facilitate proactive event driven computing is *a new kind of event processing component*. Event processing is an approach to software systems that is based on reaction to events, often under time constraints. It includes specific logic to filter, transform, or detect complex events and patterns in events as they occur [8]. The CEP component needs to be extended to cope with detecting and forecasting derived events under uncertainty.

The second building block facilitates *event recognition and forecasting*, that is, identifying events that either have occurred or are likely to occur in the near future. This is a key enabler of proactive computing, allowing decision-making to commence even before an event has been (completely) detected. This building block continuously refines event recognition and forecasting given the incoming, possibly noisy, data streams, in order to improve the recognition accuracy and probability estimations. Recognition and forecasting exploit models that can be created by human experts or through goal-driven supervised learning that exploits offline data available to the organization, or a combination thereof. A particularly challenging aspect of event forecasting is the temporal dimension. To facilitate precisely-informed online decision-making, forecasting should indicate not only which event will happen and with what probability, but also *when* it is expected to happen; more generally, forecasting should provide a probability distribution over the expected occurrence time.

The third building block enables the *event-based real-time decision making under uncertainty*. In order to realize proactivity and support autonomous or semi-autonomous decision-making, a body of tools is required that can exploit the forecast models and state predictions as a basis for decision-making. These tools will have to properly consider the nature and degree of uncertainty in the models' forecasts when generating decisions.

The forth building block, the *visualization component* (or *dashboard*) supports the human interpretation of decisions made in runtime. It facilitates decision making process for business users by providing easily comprehensible visualization of detected or forecasted situations along with output of the automatic decision making component – a list of suggested actions to deal with the situation. The proposed architecture can be run in open, closed, or hybrid loop mode. In case of the open loop, the user can approve, reject, or modify the action proposed by the automatic decision maker. The closed loop operation does not require user's approval, the action is performed automatically. A hybrid mode where some types of actions are taken automatically while other types require human attention is also supported.

With the quantity of events, the volume of historical data, and the complexity of applications all growing fast, it is vital that the proposed architecture also exhibit scalable behavior. Scalability has several dimensions, including scalability in streaming events, scalability in volume of historical data, scalability in amount of data sources and sinks, scalability in amount of processing elements, and scalability in terms of physical infrastructure.

4. SPEEDD ARCHITECTURE

In the scope of the SPEEDD project a proactive event-driven architecture has been proposed [10] that follows the conceptual architecture presented in Section 3 and consists of all the building blocks introduced. In the following sections we describe this architecture using the traffic management scenario.

4.1 System Requirements

The requirements for the current prototype are derived from the traffic management use case. The detailed requirements can be found in [2].

The prototype should provide authoring tools that could be applied to the historic data in order to derive event pattern definitions and decision models to be deployed in runtime, as well as a scalable runtime system capable of detecting and predicting

[2] http://www.sensysnetworks.com

[3] http://www.aimsun.com/wp

important situations (traffic conditions) and issuing automatic actions aimed at preventing undesired situations (congestions).

For the traffic management scenario, the projected throughput is 2000 sensor readings per second (computed based on the amount of sensors and the report frequency, assuming aggregated readings sent every 15 seconds by each of the 130 Sensys sensors installed along the Grenoble South Ring).

In terms of integration with external systems the following is required:

- Replay historic events from text files or a database.
- Receive sensor reading messages generated by the micro-simulator.
- Provide a mechanism to log output events and actions to a log for subsequent research.
- Provide a mechanism to connect to the traffic micro-simulator for updating the simulator configuration – action simulation.

4.2 SPEEDD Runtime Architecture

The architecture of the runtime part of SPEEDD follows the Event-Driven Architecture paradigm [12]. This approach facilitates building loosely coupled highly composable systems, as well as provides close alignment with the real world problems, including our representative use case. Every component functions as an event consumer, or an event producer, or a combination of both. The event bus plays a central role in facilitating inter-component communication which is done via events. Figure 1 shows the event-driven architecture for SPEEDD where the runtime part is represented as a group of loosely-coupled components interacting through events. The event bus serves as the communication and integration platform for SPEEDD runtime.

Input from the operational systems (traffic sensor readings) are represented as events and injected into the system by posting a new event message to the event bus. These events are consumed by the CEP runtime. The derived events representing detected or forecasted situations that CEP component outputs are posted to the event bus as well. The decision making module listens to these events so that the decision making procedure is triggered upon a new event representing a situation that requires a decision. The output of the decision making represents the action to be taken to mitigate or resolve the situation. These actions are posted as action events. The visualization component consumes events coming from two sources: the situations (detected as well as forecasted) and the corresponding actions suggested by the automatic decision components. Architecturally, there is no difference between these two – both are events that the dashboard is 'subscribed to', although having different semantics and presented and handled differently. The user can accept the suggested action as is, modify the suggested action's parameters, or reject it (and even decide upon a different action). In the case where an action is to be performed, the resulting action will be sent as a new event to the event bus so that the corresponding actuators are notified.

Specifically, Figure 2 shows the SPEEDD runtime architecture for the traffic management use case, including the technology platforms used to implement the architecture. In the following subsections we describe the details of the runtime architecture including the design of each component and its technology implementation.

4.2.1 Event Bus

The technology chosen for the event bus component is Apache Kafka [16]. It provides a scalable, performant, and robust messaging platform that matches SPEEDD requirements. To implement routing of the events to event consumers we build upon the topic-based routing mechanism provided by Kafka.

To allow scalable processing of massive stream of messages at high throughput, Kafka provides the partitioning mechanism. Every topic can be partitioned into multiple streams that can be processed in parallel, while every partition can be managed in a separate machine. There may be more than one replica for every partition, thus providing resilience in case of failures.

In SPEEDD we exploit Kafka partitioning to build a scalable and fault-tolerant event bus. The topic that receives the biggest incoming traffic is *speedd-in-events* where all the input events are sent. The decision about the partitioning mechanism to use is use-case specific as we want to achieve nearly uniform distribution of load over different partitions. Below, we describe the partitioning approach for our use case, providing the rationale for the design decisions. It is important to mention, though, that we may change the final partitioning mechanism based on the performance experiments on real and simulated data. We will be able to do that at any stage of the project development, thanks to the highly extensible and customizable partitioning framework that Kafka provides.

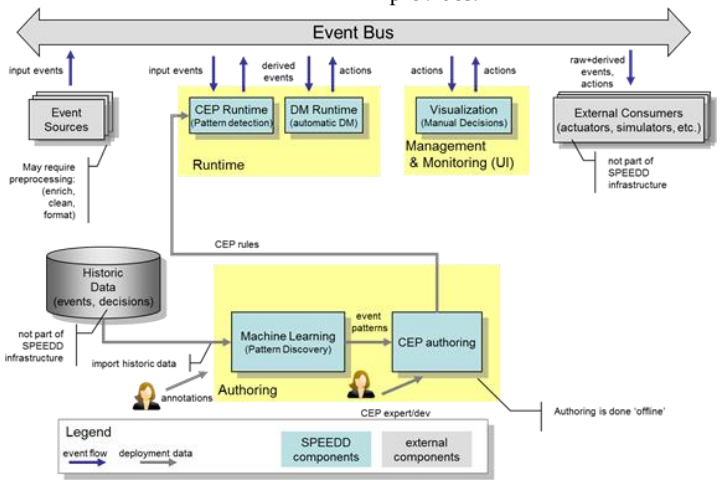


Figure 1. SPEEDD Event-Driven Architecture

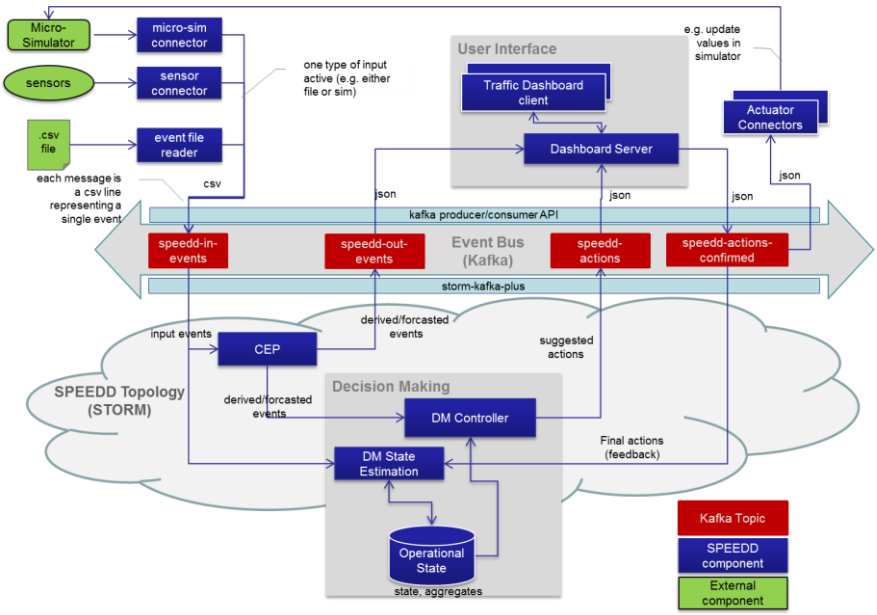


Figure 2. SPEEDD Runtime Event-Driven Architecture (Traffic Use Case)

4.2.1.1 Partitioning for the Traffic Use Case

Assuming that we get relatively equal amount of events produced by every sensor, we could partition sensor reading events based on the sensor id. This should result in uniform distribution of the messages to partitions, which provides horizontal scalability of the topic.

4.2.1.2 Ordering of events

Kafka guarantees that the order of events submitted to a topic's partition is preserved within same partition – the consumers will receive them in the same order. However, the order is not guaranteed across partitions. In our case, this should not be an issue because the CEP component takes care of the out-of-order events as long as the delay between the event and its preceding event that arrives after that event is not too long – this assumption should be valid with Kafka.

4.2.1.3 Storm-Kafka Integration

SPEEDD event processing and decision making components run on top of Apache Storm [25], a distributed scalable stream processing infrastructure.

Integration between Storm streaming platform and our Kafka-based event bus is done based on the Storm-Kafka-Plus project[4]. Storm-Kafka-Plus provides two building blocks. KafkaSpout listens on a Kafka topic and creates a stream of the tuples. KafkaBolt posts incoming tuples to a configured topic. There is an extensible mechanism for serialization and deserialization of tuples to messages and vice versa.

4.2.2 Event/Data Providers

Event providers provide the input interface of SPEEDD runtime with the external world. Every event that occurs in the external world that should be taken into account by SPEEDD to detect or predict an important business situation should be sent to the *speedd-in-events* topic on the event bus as a message representing the event.

As it is illustrated in Figure 2, events for the traffic use case may come from traffic sensors (magnetic wireless Sensys sensors buried in the road), micro-simulator (synthetically generated data), as well as historic data (collected data from sensors).

To enable processing of events generated by any of the above sources, a connector should be developed. The connector uses source-specific integration mechanism to read the data from the event sources and send them to SPEEDD event bus using Kafka producer API.

We define three connector types corresponding to the types of the event sources, that is, file-reader (replay past events from a file) sensor, and micro-simulator connectors.

4.2.3 Action Consumption – Actuators/Connectors

The outcomes of SPEEDD are actions that should be applied in the operational environment to resolve a problem or prevent a potential problem. According to the event-driven architecture principles, actions are represented as outbound events and are available to every interested party to receive and process them. The actuators connectors are interface points in SPEEDD architecture responsible for listening to the speedd-actions-confirmed topic for new actions and connect to operational systems to execute respective operations.

As it is not planned to connect SPEEDD prototype to the traffic operational systems running in production mode, the detect→decide→act loop will be implemented and tested using the AIMSUN micro-simulator [2]. The traffic actuator connector will listen to the outbound action events (speedd-actions-confirmed topic on the event bus) and execute operations supported by the micro-simulator, e.g., update speed limits, set ramp metering rates, etc. The integration with the event bus for actuators is based on the Kafka consumer API.

[4] https://github.com/wurstmeister/storm-kafka-0.8-plus

4.2.4 Complex event processing component

The main role of the CEP component is to detect events and derive situations to feed the decision module, so proactive actions can be taken. To this end, the CEP component needs to deal with uncertainty in the input, as well as the output events.

We use the IBM Proactive Technology Online (Proton) research asset as the CEP engine in SPEEDD. This engine has been released as open source as an outcome of the FI-WARE project[5] and it is extended to cope with predictive capabilities in the scope of the SPEEDD project.

Proton receives raw events, and by applying patterns defined within a context on those events (we follow the terminology in [8]), computes and emits situations (derived events emitted to consumers). Proton is platform-independent, as it is implemented in Java. The architecture is modular and consists of the following components:

Adapters – communication of Proton with external systems

Parallelizing agent-context queues – for parallelization of processing of single event instance, participating in multiple patterns/contexts, and parallelization of processing among multiple event instances.

Context service – for managing of context's lifecycle – initiation of new context partitions, termination of partitions based on events/timers, segmenting incoming events into context groups which should be processed together.

EPA manager – for managing Event Processing Agent (EPA) instances per context partition, managing its state, pattern matching, and event derivation based on that state.

SPEEDD will take advantage of the adaptation of the standalone architecture of Proton to a distributed architecture done in the scope of the FERARI FP7 EU project[6], and will apply the Proton on Storm version of the engine. It is important to note that, while Storm offers an open programming model so developers can add the logic to address complex event driven applications, the resulting implementation is custom to a single application and not a generic re-usable solution. Furthermore, the inclusion of uncertainty requires additional specific coding to deal with. In the architecture proposed, we make use of a generic event processing system that provides the necessary building blocks to build generic event driven applications with the presence of uncertainty.

The Proton architecture on top of Storm preserves the same logical components as are present in the standalone architecture: the queues, the context service and the EPA manager, which constitutes the heart of the event processing system. However the orchestration of the flow between the components is a bit different, and utilizes existing Storm primitives for streaming the events to/from external systems, and for segmenting the event stream.

4.2.5 Decision making component

As aforementioned, the aim of the real-time decision making building block is to provide a body of proactive event-driven decision-making tools, which exploit the detected or forecasted events of the CEP. The Decision Making (DM) module receives as inputs the detected, derived, and forecasted events and emits control actions or appropriate suggestions. Therefore, it functions both as an event consumer and as an event producer at the same time.

In this sense, decision making is the task of finding the optimal response to a specific situation, which is described by the detected or forecasted events. It is naturally represented as a parametric optimization problem. The main task of decision making is to solve this optimization problem, which can be accomplished in two conceptually different ways:

The parametric optimization problem is solved offline such that an explicit solution is obtained. Note that this is a "difficult" task, since an optimal answer to any situation that might arise during operation needs to be computed. If such an explicit solution can be obtained, it takes the form of a feedback rule, e.g. a linear controller K(s) or state feedback - K*x. Therefore, it can be efficiently implemented in a unified architecture using the existing SPEEDD components (e.g., as a Storm Bolt).

The construction of an explicit solution may be computationally intractable for certain problems. In such a case, the solution to multiple distinct instances of the optimization problem needs to be computed at runtime. In contrast to the first case, in which only the evaluation of a feedback rule is required, the algorithmic solution of an optimization problem is not trivial and it is not tractable to solve such a problem within the stream processing environment adopted in SPEEDD (Storm). We, therefore, assume the existence of a use-case specific "optimization black-box" outside the actual SPEEDD framework, which can be queried whenever such a decision is required.

In our illustrative example of freeway ramp-metering (regulating the traffic inflow on a freeway in order to maximize throughput), a low-level ramp metering controller receives measurements of the local traffic density and the local traffic flows, as well as notifications about detected or predicted congestion queues. It then emits a recommendation to change the ramp metering rates accordingly. For a network of interaction freeways, a network-wide planning algorithm can be used for coordination purposes, implemented as an external oracle that can be queried.

Since a road network is naturally a spatially distributed system, the architecture of the decision-making module reflects this structure. Specifically, the module is directly and efficiently implemented as Storm bolts in a distributed manner. Preliminary theoretical results suggest that such local controllers may perform asymptotically optimal with regard to flow maximization for a single freeway; however, coordination is required to achieve optimal operation of more complex road networks. Network-wide planning can be superimposed by querying an external black-box.

4.2.6 Dashboard component

As aforementioned, the proposed event-driven architecture can be run in an open, closed, or hybrid loop mode. In the traffic management use case we only deal with open or hybrid modes, i.e., we don't have fully automatic actuators for the decisions. The closed mode implies connecting the SPEEDD prototype to the actual production systems and, therefore, out of the scope of the project.

In our current scenario, operators interact with the outputs of the SPEEDD modules through a User Interface (UI). The Dashboard Client communicates, via the Dashboard Server, with the composite systems in the SPEEDD architecture. Operators can accept, respond to, or make suggestions and control actions. Actions taken by operators via the UI are fed back into the SPEEDD runtime as events, thus allowing for the seamless

[5] https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Architecture

[6] http://www.ferari-project.eu

integration of expert knowledge and the outputs of complex algorithms.

The Dashboard Server component is based on Node.js [24] asynchronous programming framework. The server functions as a Kafka consumer and producer. The consumer listens for broadcasted messages in the Event Bus under the following topics: *speedd-out-events* and *speedd-actions*. The producer broadcasts messages under the topic *speedd-actions-confirmed* (see section 4.2.1)

The Dashboard Client is designed to provide the user with a clear picture of the current state of the world. It achieves the picture of the current state by aggregating sensor readings in human readable form, current states of the control equipment available (e.g., speed limit signs, message signs, lanes), current events identified by the CEP module, and displays of the automated control events produced by the DM unit (e.g., ramp metering rates). Furthermore, it aims to support the decision-maker by highlighting events which might require attention along with corresponding suggested mitigating strategies.

4.3 SPEEDD Design Time Architecture

In general, there exist two methods to define the rule patterns for a CEP application: machine learning and experts. In the first, the patterns are learnt automatically by a computer program, while in the second, they are given by an external entity; usually a subject expert matter specialized in the domain. It is also possible to combine between these two methods.

Historic data used at design time contains raw events reported during the observed period along with annotations provided by domain experts. These annotations mark important situations that have been observed in the past and should be detected automatically in the future. Domain experts can apply tools and methodologies provided by SPEEDD authoring toolkit to extract derived event definitions from the annotated event history. This is a semi-automatic process involving applying machine learning tools to extract initial set of patterns, then further enhanced and translated with help of the domain experts into deployable CEP artefacts.

Due to the dynamic nature of the proactive traffic management application, the knowledge base of event pattern definitions may require to be refined or enhanced with new ones. Manual creation of event definitions is often a tedious and cumbersome process, thus we employ machine learning techniques to semi-automatically create event pattern definitions by analyzing historical data.

We employ the Probabilistic Event Calculus [23] that combines temporal logic-based formalization with probabilistic modelling. The logic-based representation allows to compactly define relations between events and incorporate existing domain knowledge, while probabilistic modelling allows to naturally handle uncertainty. For the implementation of the machine learning algorithms, we extend the open-source framework LoMRF[7] with state-of-the-art scalable probabilistic inference and incremental learning methods [14]. LoMRF is developed in Scala[8], which compiles to Java bytecode and thus works seamlessly with any other Java-based framework.

Additionally, for scalability LoMRF employs the high-performance parallel processing framework of Akka Actors[9].

The resulting output of the machine learning algorithms is composed of a set of text-formatted files that contain the event pattern definitions. Thereafter, the resulting rules are parsed by the "rtec2proton" translator and converted semi-automatically to JSON formatted Proton EPN definitions. All EPN definitions are then reviewed and manually refined by domain experts using Proton's authoring tool. The output of this process is a JSON file containing the EPN definition.

5. RELATED WORK

Proactive applications have been developed in an ad-hoc manner for several years; some examples include proactive security systems [5], proactive routing in mobile ad-hoc wireless [17], proactive network management with failure handling [11], proactive service level agreement negotiation in service oriented systems [18], proactive caching [15], and proactive management in logistic processes [19] and [9]. However, the lack of a generic paradigm to develop proactive event-driven applications makes it difficult for this capability to spread.

One of the main ingredients for proactive event driven computing is the ability to deal with uncertainty in the events. Despite uncertainty handling has been recognized as one of the most critical and relevant aspects in the area of CEP, it still remains an open issue [1]. Only a few solutions have been proposed, and most of them are tailored to a specific application domain [4]. Examples of previous works can be found in [4], [20], [22], [26], [27] and [28]. Existing CEP approaches examine three major types of uncertainty that may be present in the events that are fed in a CEP system: uncertainty in event content, in the event occurrence, and in the rules. Our CEP component must support these three types. Furthermore, learning event rules in the presence of uncertainty is also an open research area [1].

In terms of real-time optimization techniques, the state-of-the-art is that optimization techniques are being activated mostly off-line and use a variety of optimization methods that fit different assumptions: robust (worst-case) optimization, stochastic optimization, and optimization methods based on black-box models (e.g., [3], [13] and [21]). Our main challenge is to develop real-time proactive planning tools for proactive applications using these optimization methods within an event-based planning framework.

6. SUMMARY AND FUTURE WORK

Event-driven architecture is a software architecture pattern promoting the production, detection, consumption of, and reaction to events. We describe how we extended this architecture from being reactive to proactive, by incorporating capabilities for forecasting and real-time decision making.

The proposed architecture is instantiated by a real use case from the traffic management domain. Although driven by the use case requirements in the SPEEDD project, the proposed architecture is generic and can be applied to any domain that requires proactive event-driven computing.

We are currently working on a first implementation of the use case based on the proposed architecture. Future work includes integration of offline historic data and online streaming data as

[7] https://github.com/anskarl/LoMRF

[8] http://www.scala-lang.org

[9] http://akka.io

well as refinements to the proposed architecture as result of the implementation.

7. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme FP7/2007-2013 under grant agreement 619435 (SPEEDD).

8. REFERENCES

[1] Artikis A., Etzion O., Feldman Z., and Fournier F. 2012. Event Processing under Uncertainty. In *Proceedings of the sixth ACM conference on Distributed Event-Based Systems (DEBS'12).*

[2] Baber C., Bellicot I., Canudas de Wit C., Cooke N., Garin F., Grandinetti P., Hempel A., Kibangou A., Morbidi F., and Schmitt. M. User requirements and scenario definition. Accessible at http://www.speedd-project.eu/sites/default/files/D8.1-UserRequirements_final.pdf

[3] Ben-Tal, A., Boyd, S., and Nemirovski. A. 2006. Extending Scope of Robust Optimization: Comprehensive Robust Counterparts of Uncertain Problems. *Mathematical Programming*, 107:1-2, 63-89.

[4] Cugola G., Margara A., Matteucci M., and Tamburrelli G. 2014. Introducing uncertainty in complex event processing: model, implementation, and validation. *Computing* 1–42.

[5] Dolev S., Kopeetsky M., and Shamir A. 2011. RFID Authentication Efficient Proactive Information Security within Computational Security. *Theory of Computing Systems,* 1-18.

[6] Engel Y., Etzion O., and Feldman Z. 2012. A Basic Model for Proactive Event-Driven Computing. In *Proceedings of the sixth ACM conference on Distributed Event Based Systems (DEBS'12)*.

[7] Engel Y. and Etzion O. 2011. Towards proactive event-driven computing. In *Proceedings of the fifth ACM conference on Distributed Event Based systems (DEBS'11)*.

[8] Etzion O. and Niblett P. 2010. *Event Processing in Action.* Manning Publication.

[9] Feldman Z., Fournier F., Franklin R., and Metzger A. 2013. Proactive event processing in action: A case study on the proactive management of transport processes, in *Proceedings of the Seventh ACM International Conference on Distributed Event-Based Systems (DEBS'13).*

[10] Fournier F., Kofman A., Morar N., Schmitt M., Skarbovsky I., and Skarlatidis A. 2014. The Architecture Design of the SPEEDD Prototype. Accessible at http://www.speedd-project.eu/sites/default/files/D6.1-Architecture_Design_of_SPEEDD_Prototype-v1.0a.pdf

[11] Fu S. and Xu C.Z. 2007. Exploring event correlation for failure prediction in coalitions of clusters. *In Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC'07)*, 1-12.

[12] Hohpe G. Programming without a call stack – Event-driven Architecture. 2006, [Online], at: http://www.eaipatterns.com/docs/EDA.pdf

[13] Hokayem P., Chinquemani E., Chaterjee D., Ramponi F., and Lygeros J. 2012. Stochastic receding horizon control with output feedback and bounded controls. *Journal Automatica*, 48, 1, 77-88.

[14] Huynh, T. N., & Mooney, R. J. 2011. Online structure learning for markov logic networks. *In Machine Learning and Knowledge Discovery in Databases*, 81-96.

[15] Kohler M. and Fies R. 2009. ProActive Caching-A Framework for Performance Optimized Access Control Evaluations. In *Proceedings of IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2009).*

[16] Kreps, J., Narkhede N., and Rao J. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB).*

[17] Kunz T. and Alhalimi R. 2010. Energy-efficient proactive routing in MANET: Energy metrics accuracy. *Ad Hoc Networks*, 8(7), 755-766.

[18] Mahbub K. and Spanoudakis G. 2010. Proactive SLA Negotiation for Service Based Systems. In *Proceedings of the 6th World Congress on Services (SERVICES-1).*

[19] Metzger A., Franklin R., and Engel Y. 2012. Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In *Proceedings of the Annual SRII Global Conference (SRII)*, 313-322.

[20] Ré C., Letchner J., Balazinksa M., and Suciu D. 2008. Event queries on correlated probabilistic streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*, 715–728.

[21] Shapiro, A. 2008. Stochastic programming approach to optimization under uncertainty. *Mathematical Programming*, 112.

[22] Shen Z., Kawashima H., and Kitagama H. 2008. Probabilistic event stream processing with lineage. In *Proceedings of Data Engineering Workshop (DEWS' 08).*

[23] Skarlatidis A., Paliouras G., Artikis A. and Vouros G. 2014. Probabilistic Event Calculus for Event Recognition. *ACM Transactions on Computational Logic (TOCL)*, to appear.

[24] Tilkov S. and Vinoski S. 2010. Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing 14(6)*, 80-83.

[25] Toshniwal, A., Taneja S., Shukla A., Ramasamy K., Patel J. M., Kulkarni S., Jackson J., et al. 2014. Storm@twitter. In *Proceeding of the 2014 ACM SIGMOD international conference on Management of data (SIGMOD '14)*, 147-156

[26] Wasserkrug S, Gal A., Etzion O., and Turchin Y. 2012. Efficient processing of uncertain events in rule-based systems. *IEEE Transactions on Knowledge and Data Engineering*, 24(1), 45–58.

[27] Wasserkrug S., Gal A., Etzion O., and Turchin Y. 2008. Complex event processing over uncertain data. In *Proceedings of the Second ACM conference on Distributed Event-Based Systems* (DEBS '08), 253–264.

[28] Zhang H., Diao Y., and Immerman N. 2010. Recognizing patterns in streams with imprecise timestamps. *Proc. VLDB Endowment*, 3(1-2), 244–255.

Towards Flexible Event Processing in Distributed Data Streams *

Sebastian Bothe
Fraunhofer IAIS
sebastian.bothe@ iais.fraunhofer.de

Vasiliki Manikaki
Technical University of Crete
manikaki@softnet.tuc.gr

Antonios Deligiannakis
Technical University of Crete
adeli@softnet.tuc.gr

Michael Mock
Fraunhofer IAIS
michael.mock@ iais.fraunhofer.de

ABSTRACT

The FERARI project aims to develop a highly scalable distributed streaming architecture supporting complex event processing in a communication-efficient manner. Two key requirements for our system are that its architecture is not tied to the underlying streaming platform used in its implementation and that it allows the easy definition of communication-efficient methods for monitoring a global condition over a distributed set of states. In this paper we present the architecture of our system and explain how these key requirements are met. Concerning the actual implementation of our system in a scalable distributed streaming platform, it is reasonable not to re-invent the wheel but to use one of the actual Big Data Streaming platforms as a starting point. For this reason, we evaluate some popular platforms and discuss whether they meet our requirements.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

1. INTRODUCTION

In recent years, an area with great future potential for Big Data is machine-to-machine interaction (M2M), and the Internet of Things. Examples of relevant applications include smart energy grids, car-to-car communication, mobile network quality monitoring, optimizing operation of large and complex systems, fault detection in clouds, automated negotiation systems – all these have been identified as important hot use cases for Big Data.

Current Big Data technologies, developed for systems that process and analyze human generated data such streams, managing social networks at Facebook, or indexing web pages at Google, seem inadequate for processing of such M2M applications. In order to understand this, note that the data volumes generated from M2M interaction surpass by far the amount of data generated by humans. M2M data is typically required to be processed in real-time as it is produced, it is predominantly transient (does not need to be and is too large to be stored for future reference), and is typically much more structured in nature than human-generated data.

Due to the sheer size of M2M data, approaches that seek to centralize this data are not an option, as they (i) would require enormous infrastructures both for storage, as well as for the required bandwidth for transmitting this data, (ii) would impose unnecessary latency due to the data shuffling possible, and (iii) do not consider the characteristics and limitations of the data sources of M2M data - sensors are often the sources of M2M data and constant communication of sensor readings would quickly drain the energy of sensor nodes. It is thus important to be able to process M2M data and to detect important events without centralizing the collected data, but rather doing as much processing and filtering of the data at the nodes that produce it.

The project *Flexible Event pRocessing for big dAta aRchItectures* (FERARI) aims at developing a highly scalable distributed streaming architecture supporting complex event processing (CEP) in a communication-efficient manner. While most CEP systems are built on the premise that primitive events are obtained and transmitted by the remote data sources based on their own data, a key element of the FERARI architecture will include the development of communication efficient distributed methods for also detecting events, expressed over the data of multiple nodes, in a distributed manner. We consider the important case of complex events that can be expressed as a monitoring task that alerts whenever a complex function, expressed over the data of multiple nodes, has exceeded a threshold. In order to make the complex event processing feasible, a key component is to perform in-situ processing at the nodes generating the data, thus avoiding continuously pushing related data or events to our CEP engine. A key component that we will utilize for such distributed monitoring tasks is the recently developed [10, 11, 12, 13, 8, 7, 9] geometric approach. The details of this geometric approach are presented in Section 2.

We present the general architecture of FERARI and argue that a flexible CEP system for M2M data should not be tied to a specific implementation using existing stream processing systems as its infrastructure. To develop a generic architecture, in Section 3 we specify the essential building blocks that it must contain and then consider which of some existing big data streaming platforms seems more appropriate for our actual implementation. Given our architecture, in Section 4 we explain how an important part of this architecture, namely the distributed detection of events using the geometric approach, can be developed in an existing open source platform, such as Apache Storm, and explain how some distributed monitoring tasks (that may use the geometric approach, or not) fit within our architecture.

2. BASICS - THE GEOMETRIC APPROACH

We now describe in more detail the geometric approach for function monitoring over a distributed system of n sites. Figure 1 demonstrates the basic ideas of the geometric approach that we discuss in this section.

Each site S_i maintains a local d-dimensional vector, termed as the *local statistics vector*, with the j-th ($j = 1 \dots d$) element of the local statistics vector of S_i denoted as $\vec{v}_{j,i}$. All sites contain a vector of the same dimensionality (i.e., number of elements). The *global statistics vector* $\vec{v}$ is computed as the average[1] amongst all local statistics vectors. Thus, the j-th component of the global statistics vector, denoted as $\vec{v}_j$ is computed as: $\vec{v}_j = \frac{1}{n}\sum_{i=1}^{n} \vec{v}_{j,i}$.

For the framework to be applicable, any supported monitoring function $f : \mathcal{R}^d \rightarrow \mathcal{R}$ must be expressed over the global statistics vector $\vec{v}$ (thus, over the average of all local statistics vectors). An important feature is the wide applicability of the geometric approach, as the threshold function can in general be non-linear. Given a threshold T, the framework in [10, 11, 12, 13, 8, 7, 9] can safely determine whether $f(\vec{v}) > T$.

The geometric approach decomposes the monitoring task into a set of constraints (one per site) that each site can monitor *locally*. To achieve this, during the operation of the algorithm, each site S_i maintains (i) the estimate vector $\vec{e}$, which is equal to the global statistics vector $\vec{v}$ computed by the local statistics vectors transmitted by sites at certain times, and (ii) a delta vector $\Delta\vec{v}_i$, denoting the difference of the current local statistics vector from the last local statistic vector that S_i has transmitted. Based on these two quantities, S_i calculates its drift vector $\vec{u}_i = \vec{e} + \Delta\vec{v}_i$. Additional optimization have been developed in the framework, such as the ability to *balance* only a portion of the network in case of violations. In that case, an additional *slack* vector needs to be maintained and added in the calculation of the drift vector.

The domain space $\mathcal{R}^d$ represents the potential locations of the global statistics vector at any time. Let all points in $\mathcal{R}^d$ where $f(\vec{v}) <= T$ be colored by the same color (i.e., white in Figure 1), while the remaining points be colored by a different color (i.e., green in Figure 1). Because the sites do not perform transmissions at each time period, the current global statistics vector $\vec{v}$ is not known to the sites. However, what is guaranteed is that $\vec{v}$ will always lie within the convex hull $Conv(\vec{u}_1, \dots, \vec{u}_n)$ of the drift vectors and, thus, within the convex hull $Conv(\vec{e}, \vec{u}_1, \dots, \vec{u}_n)$ of the drift vectors and the estimate vector. Thus, if $Conv(\vec{e}, \vec{u}_1, \dots, \vec{u}_n)$ is *monochromatic* (i.e., either entirely below/equal to the threshold, or entirely above to the threshold), then all sites are certain about the color of the function $f()$, since this will coincide with the color of $f(\vec{e})$. Of course, each node cannot compute $Conv(\vec{e}, \vec{u}_1, \dots, \vec{u}_n)$, since it is not aware of the current drift vectors of other sites. However, an important observation [11] is that if each site monitors the sphere $B(\vec{e}, \vec{u}_i)$ constructed with diameter the estimate vector and its own drift vector, then the union of these spheres covers the convex hull. Thus, it suffices for each node to simply monitor whether its sphere is monochromatic. If all the spheres are monochromatic, then the convex hull is also monochromatic and, thus, $f(\vec{v})$ has the same color as $f(\vec{e})$. Otherwise, nodes transmit their local statistics vectors, and a new estimate vector is computed and made known to all nodes.

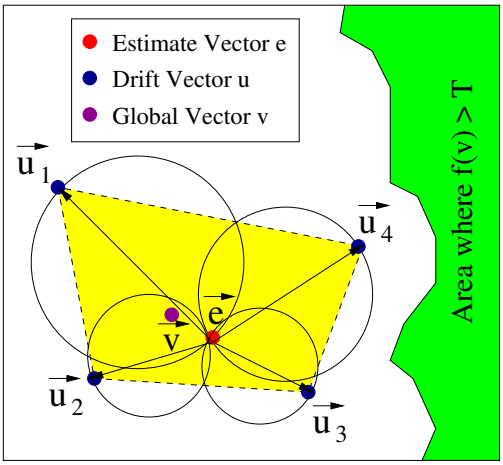


Figure 1: Local constraints using the Geometric Approach. Each node constructs a sphere with diameter the drift vector $\vec{u}$ of the node and the estimate vector $\vec{e}$. The global statistics vector $\vec{v}$ is guaranteed to lie in the convex hull of $\vec{e}$, $\vec{u}_1$, $\vec{u}_2$, $\vec{u}_3$, $\vec{u}_4$. The union of the local spheres covers the convex hull.

Using Safe-Zones. The more recent work of [8, 7, 9] simplifies the local tests performed by nodes by having each node test whether its drift vector [8, 9] or its local statistics vector [7] lies within a convex region. This test is very efficient and only depends on the complexity of the bounding convex region. For example, the work in [9] demonstrates how this convex region can be determined by the intersection of hyperplanes. In that case, the local test of each node simply checks that a tested vector lies on the "correct" side of these hyperplanes.

3. PROPOSED ARCHITECTURE

To build a flexible event processing application, we assemble an appropriate algorithmic approach and a stream processing platform. The approach we use to create this assembly is

[1]The same framework also applies when the global statistics vector is calculated as a weighted average of the local statistics vectors.

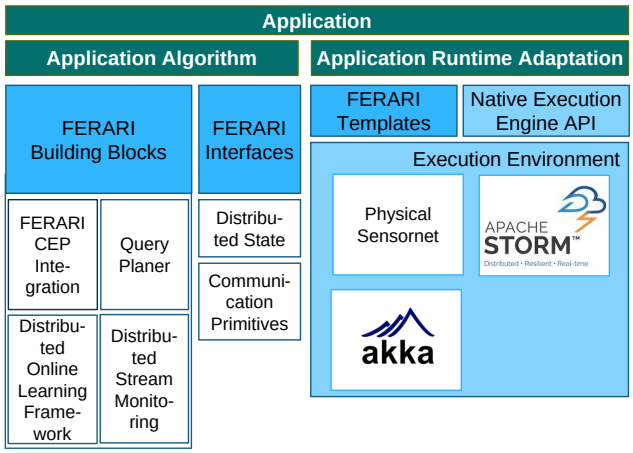


Figure 2: FERARI architecture overview.

described in Section 3.1. We show how the application can be organized such that the application core can be made independent from the execution environment. Additionally, several open source projects are working on scalable distributed streaming platforms. It is therefore reasonable not to start from the beginning. To identify a good starting point, we compare the current systems in Section 3.2.

3.1 Architecture Components

We design the FERARI architecture to allow the fast development of distributed flexible event processing applications. To achieve this, it is required that we can create new functionality on the basis of existing ones, and that we can exchange parts of the system without affecting other parts. A common approach in software development for this requirement is to decompose the complex system into smaller units, called components or modules. Each of the components has clearly defined bounds and a dedicated functionality. Additionally, the interaction between the components is defined in terms of interfaces. We use this approach and derive appropriate components and interfaces. An overview of the proposed FERARI architecture is depicted in Figure 2.

We now describe the components and the decisions, that lead to our choices. It is important to keep the architecture as generic as possible, as we need to execute applications in different runtime environments. As a runtime we consider a distributed system, that provides the actual execution context for the application. This may be a simulator, a distributed stream processing framework or even a distributed physical sensor network. Especially for in-situ processing there needs to be an adaptation optimized for the application's concrete environment. For that reason, we start the decomposition of the application by isolating its algorithm from the application's part that depends on a specific execution environment. The first part we call *application algorithm* and the latter *application runtime adaptation*. The division now makes the application algorithm independent from the selected runtime platform and can, therefore, be reused in different execution environments. This decoupling from the runtime now raises the demand to define abstract versions of mechanisms used by our distributed algorithms.

The distributed algorithms in our current set of example applications need access to communication primitives and a distributed state. The concrete mechanisms for both depend on the selected runtime adaptation, which is not accessible by the application algorithm. To enable this access, we introduce a set of *FERARI interfaces*. We develop the FERARI system using an agile process, therefore we created the interfaces that we need for the geometric approach. We want to monitor a global condition over a function, using the derived local conditions (c.f. Section 2). The local and global condition are based on a local and a global state respectively. These two parts of the state are represented by the interface for distributed state. Additionally, the coordinator needs to send information to the local node and vice versa. Primitives for sending messages in each of theses directions are provided by the other interface. The application algorithm can now be created using this interfaces. On side of the application runtime adaptation, we need to provide the concrete implementation for the mechanism exposed by the interfaces. All of the execution environments use different types of abstractions, interfaces and nomenclature. Additionally, it is a very demanding task to create a efficient runtime adaptation for an arbitrary set of possible applications. Following an agile development principle, we instead solve the more viable task of creating templates for common application types. These FERARI templates allow the re-use of runtime adaptations for different applications. They can provide a mapping of common patterns onto the runtime implementation. Beyond this, some applications may require access to special features of a runtime system. This may for instance be optimized network operations. Especially in the case of in situ processing, access to these features is important to create an efficient runtime adaptation. Therefore we provide access to these features by exposing the native API of the execution engine.

It is possible to create applications using just the FERARI interfaces and a runtime adaptation, and we give an example for this in Section 4.1. Nevertheless, writing the application code using these interfaces requires skills in distributed programming and especially knowledge about communication efficient algorithms. Therefore, we plan to provide a set of ready to use blocks that can be called from the application code directly, the *FERARI Building Blocks*. As a first example of these blocks, we describe the current status of the distributed stream monitoring block in Section 4. An outlook for the other planned blocks is given in our concluding remarks (Section 5).

3.2 Candidate Streaming Platforms

As we identified in Section 3.1, it is important to decouple the application algorithm and the execution runtime. We will now focus on candidate platforms within the existing open source streaming platform implementations. The FERARI architecture itself will be available under a liberate open source license, the Apache license. For that reason, we consider only platform candidates that are compatible with this license. Since there are a lot of streaming platform candidates, we focus on some popular ones, Storm, Trident, Spark Streaming and Akka. We require a system that can be scaled to handle the streams of data for huge volume and high velocity. In current stream processing platforms, this is achieved by horizontally scaling out, which means

additional processing nodes are added to increase the capacity. An effect of horizontal scaling is that the increased amount of nodes also increases the probability of nodes failures. Therefore, the platform needs to provide mechanisms to deal with system components failing. In Section 2 we explained how the geometric approach divides the monitoring task to a local (checking for local violations) and a global part (synchronization and determining if there is a global violation/event). These two parts interact with each other in adaptation cycles. The ability to support such cycles is a key requirement for the platform that we choose. Additionally, our application scenarios include monitoring tasks. In this application area it is important to create immediate reaction, for instance raise an alarm as soon as possible. The requirement for the platform is, therefore, that it allows processing with low latency. Another important aspect of this application scenario is that potential alarms must not be omitted. The processing of an input event may either be guaranteed by the underlying platform (i.e., by ensuring that messages are not lost), or it may be a concern of the application. We now evaluate our candidates with respect to the properties, possibility of adaptation cycles, latency of processing and guarantees for processing provided by the platform.

Apache Storm [3] initially was developed at Twitter, got open-sourced in 2011 and is now an Apache top level project. Processing in Storm is organized by a graph, the Storm topology. Input stream data items enter the topology by spouts and are called tuples in Storm terminology. Each of these tuples is processed by Storm bolt, one at a time. The approach of processing a single tuple at a time allows for low latency processing. A storm bolt can execute arbitrary Java code, and it emits new tuples as processing results. These result tuples are then processed by the next bolts in the topology. A storm topology can contain cycles, and therefore allows for the adaptation cycles that we need. The Storm system recovers from node failures, by restarting the broken processing task at a different place. In case of failures, all state associated with a crashed tasks is lost. Storm provides two types of processing guarantees, best effort and at least once processing semantics. More on Storm's implementation can be found in [14].

Trident is not an independent system - it is an abstraction layer on top of Storm. It extends Storm by introducing exactly-once execution semantics and a model for consistent states. To achieve this, it switches from processing each tuple individually to processing small amounts of tuples, the mini batches, together. In general, mini batching increases the time that elapses between the entering of a new tuple in the system and the result being available. Further, the processing topology in Trident is required to be a cycle-free graph, which conflicts with the adaptation loop requirement that we have.

Another development independent from Storm is the distributed processing framework Akka [1]. The organization of the processing units follows the actor model. The actors interact with each other by message passing and especially cyclic connections can be constructed. There are no guarantees on delivery or processing of messages - they are handled at best effort. Each single message is processed individually, therefore achieving low latency.

	Storm	Trident	Akka	Spark
Cycles	yes	no	yes	no
Processing Guarantees	best effort and at least once	exactly once	best effort	exactly once
Processing Granularity	single tuple	mini batch	single message	mini batch

Figure 3: Properties of candidate distributed stream porcessing platforms

Apache Spark Streaming [2] is an extension of the recently developed Spark processing system. Spark is a batch processing system designed for caching intermediate results. The streaming variant organizes the processing of the stream in mini batches. Each operation on these batches is guaranteed to be performed exactly once. There is no way known to the authors for constructing loops in the processing operations.

A summary of the supported features of the evaluated platforms is provided in Table 3. Our analysis reveals that Storm provides most of the features we need for our system. Akka remains to be an interesting candidate, if the targeted application does not require at least once processing guarantees.

4. DISTRIBUTED STREAM MONITORING

We now focus on the important *Distributed Stream Monitoring* building block and explain how distributed monitoring functions can be incorporated in our framework. As we have mentioned, we are interested in detecting events, which are emitted when a function, computed over the data of different distributed nodes, has crossed a specific global threshold. We give a basic example, counting the number of distinct items in streams, to show the usage of the FERARI interfaces in Section 4.1. This basic example does not yet make use of the geometric approach. Since it is desirable to allow code reusability for different monitoring functions, we then present how declared distributed monitored functions can be implemented using a hierarchy that we define. Our function hierarchy alleviates the development of many important details of the geometric method, requiring minimal new code for each new monitoring function, while at the same time providing support for both the original geometric approach, as presented in [10, 11, 12, 13], as well as the more recent Safe-Zone [8, 7, 9] approach, which improves upon the original approach. In Section 4.2 we present our function hierarchy that allows for easily defining and incorporating new functions for distributed monitoring. Please note that both methods presented in Sections 4.1.1. and 4.2.1 do not depend on the underlying execution environment (i.e., Storm, SPARK etc). We show in Sections 4.1.2 and 4.1.3 respectively, how they can be be mapped to the FERARI architecture.

4.1 Monitoring Global Threshold with Count Distinct

4.1.1 Application Algorithm

Counting the number of distinct elements in streams of data is a common pattern in various types of applications. For

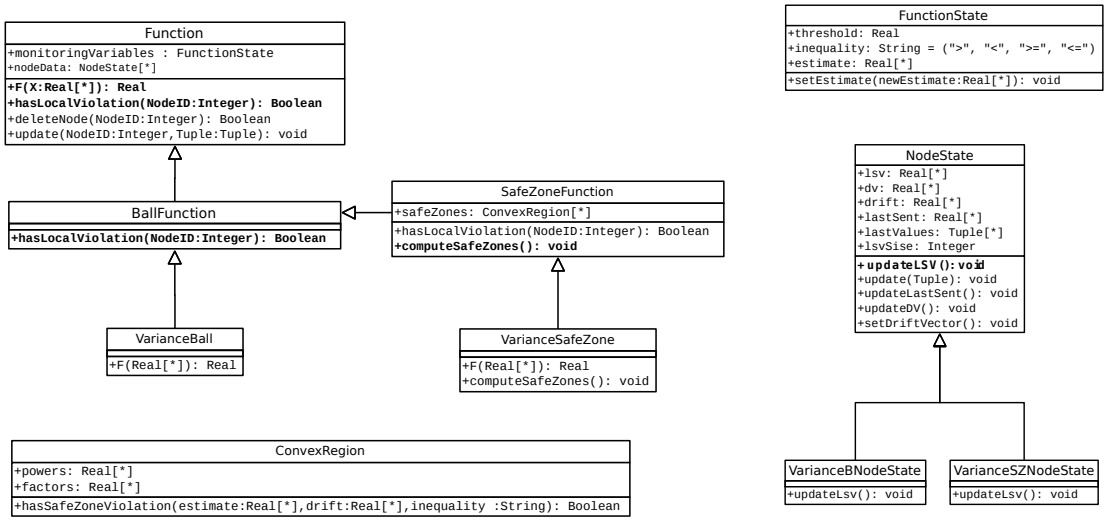


Figure 4: Function Class Hierarchy.

instance, in a mobile fraud detection scenario, it is important to keep track of the number of different locations a mobile device is used within a certain period of time. If a device changes too many times, this could be an indication of fraudulent usage. Another example for the same counting pattern, originating from a different scenario is to determine the popularity of an artist. For instance a service like last.fm[2] may monitor streams of events, in which each of the events in a stream corresponds to a user listening to a song. The popularity of an artist is measured by the number of distinct users listening to songs of the artist. To discover artists getting popular, it is relevant to know when an artists exceeded a certain global threshold, the count of distinct listeners.

An data sample of such events was collected by the author of [4] and is now publicly available at the webpage[3]. We will use these data and describe, how the pattern of counting distinct elements in streams of data can be instantiated using the proposed FERARI architecture. There are two application dependent blocks we need to fill, the application algorithm and the runtime adaption. Concerning the application algorithm, it is common practice in the streaming scenario to approximate the count of distinct items by applying sketching techniques. One type of such sketches are linear sketches, which fit our needs in this example. The details of the sketching algorithm can be found for instance in [5]. The sketch is a compact synopsis of our data that can be communicated more efficiently between the local nodes and the coordinator. For each artist we maintain a sketch to count the distinct at each of the distributed processing units. Such a sketch is communicated with the global coordinator only if it is required, i.e. if the count of distinct users has increased by some percentage or by a fixed amount. The coordinator, than also updates it's global sketch for this

[2] `http://www.last.fm/`
[3] `http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html`

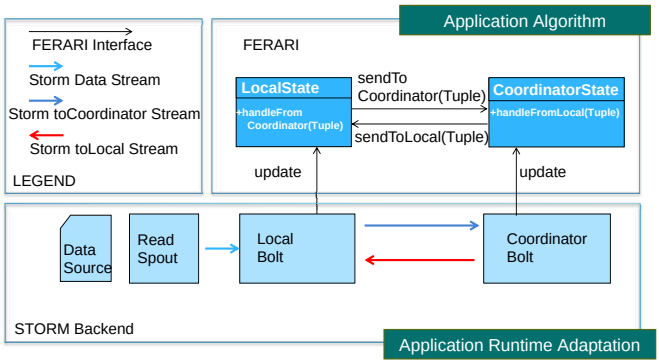


Figure 5: Interface for application algorithm.

artist appropriately. Additionally, the coordinator detects whether the global threshold has been crossed. In the case of a global threshold violation, the coordinator takes appropriate actions, for instance raises the information about the new popular artist and interacts with the local processing units to make them reset the counters.

4.1.2 Mapping to the FERARI Architecture

This algorithm can now be created using the FERARI interfaces. The local part implements the `LocalState` interface, with two methods `update` and `handleFromCoordinator`. The `update` method is called on incoming data and provides new the listen events. `HandleFromCoordinatorl` is used to reset the local counters as indicated by the coordinator. Further, increased counts are reported to the coordinator, via the `sendToCoordinator` method. For the global part, the coordinator, the equally named interface `CoordinatorState` is implemented. Here, we use the update method to receive the notifications of increased counts. As already mentioned, the coordinator resets counters in regular intervals and notifies the local units by the `send-`

ToLocal method. Note that the application algorithm now is formulated using FERARI interfaces and does not dependend on the runtime. The application algorithm can now be executed using an appropriate runtime system as proposed in Section 3. We choose the Storm runtime for our example and setup the topology as follows:

- A spout inserts the listen events as Storm tuples to the topology.
- At random choice the tuples reach one of the `LocalBolts`.
- The `LocalBolts` are connected with the `CoordinatorBolt` by a named channel, a Storm stream. This connection is achieved using a Storm `global grouping`. Vice versa, the Coordinator uses another stream to the `LocalBolts`, which is translated to a Storm `all grouping` in the topology.

The Storm runtime adaption for this example, now maps `sendToLocal` and `sendToGlobal` to emitting tuples on the dedicated channels for each of the operations. The processing in the `LocalBolt` decides on basis of the channel name, if the `update` method or the `handleFromCoordinator` is invoked. In this example the coordinator only receives tuples containing the updated sketches from the `LocalBolts`, therefore the mapping to the processing hook is unique. The complete view on the FERARI interfaces and the link to the runtime adaption is given in Figure 5.

We now describe our more general solution to define monitoring tasks exploiting the geometric approach.

4.2 Monitoring Global Function Thresholds

4.2.1 Application Algorithm

In Figure 4 we present the function hierarchy that we have developed in order to facilitate writing of applications that make use of the geometric approach. The abstract class `Function` represents the core elements that the geometric approach contains. In this class, the abstract method `F` must be provided for all developed functions and simply returns the value of the function, computed over a multi-dimensional point in the input domain. When an instance of a function is created, this is done by also specifying two important parameters: a `threshold` value and a parameter `inequality`, which may obtain one of four possible values ">", "<", "<=", ">=". A distributed event is then detected when the condition $f(v)$ *inequality threshold* becomes true. For example, when `inequality` = " >", an event is detected when $f(v) > threshold$. Once an event is detected, it remains valid for the entire time until another global violation occurs, meaning that the monitored condition has stopped being true.

In order to minimize the implementation overhead when adding new monitoring functions, a significant part of the functionality of the geometric approach has been implemented in our architecture, either at the most general `Function` class, or at the two abstract classes `BallFunction` and `SafeZoneFunction`. Starting from most general `Function` class, we notice that it contains contains two types of variables. The `monitoringVariables` parameter contains information related to the function input parameters (`threshold, inequality`) and the estimate vector `estimate`. The `nodeData` parameter contains information for one or more nodes. This is general enough to accommodate implementations over distributed systems such as Storm, where each processing node (i.e., bolts in Storm) may receive and process data for multiple nodes. For each node, we maintain:

- The most recently received data (`lastValues` variable in the `NodeState` class). The addition of this data is done through the `update` method. All stored tuples are accompanied by the corresponding timestamp that specifies when they were produced.
- The current local statistics vector (`lsv`) of the node. This is calculated, if recent data arrives by the FERARI interface `update` method of the node, through the abstract method updateLSV. For each new declared function, this method must be defined in a subclass of `NodeState`. The parameter `lsvSize` specifies the dimension of the `lsv` vector.
- Parameters relevant to the geometric approach, such as the drift vector `drift`, the delta vector `dv` from the last transmission of this node, a vector `lastSent` containing the last transmitted `lsv` vector, and the corresponding methods that update the values of these parameters. The transmission is achieved by using the FERARI interface `sendToCoordinator`.

Besides the abstract `F` method that has already been mentioned, the class `Function` also contains some additional methods. The abstract `hasLocalViolation` method answers whether a local violation has occurred using the geometric approach. In case a violation has occurred, it communicates using the FERARI interface method `sendToCoordinator`. The `hasLocalViolation` method is defined in a different way for the two subclasses of `Function`.

The `BallFunction` and `SafeZoneFunction` classes contain important functionality regarding the detection of events. The `hasLocalViolation` method is implemented in both the `BallFunction`, as well in the `SafeZoneFunction` subclasses. In `BallFunction`, the way to check for a local violation in a generic way is performed using a grid of points within the sphere that each node constructs in order to check for a local violation. Any function that wants to use the original technique with the spheres (c.f. Section 2) simply needs to: (i) Create a subclass of `BallFunction` that provides the code for the `F` method, (ii) Create a subclass of `NodeState` that provides the code for the `updateLSV` method, and (iii) optionally provide a better method for `hasLocalViolation` if for the specific function it is simple to compute its maximum and minimum values within a sphere. If the third step is omitted, the development of a new function literally requires just a few lines of code and very limited knowledge on the internals of the geometric approach.

The `SafeZoneFunction` class inherits the `BallFunction` class because we may want to define a function that uses a safe zone only when the estimate vector lies on one side of the threshold, while checking for a local violation using the spheres

in the other case. In case we want to use a safe zone, this is determined by the intersection of one or more convex regions (class `ConvexRegion`). Given the value of `lsvSize`, and two vectors `factors` and `powers` (having a dimensionality of `lsvSize+1` and `lsvSize`, respectively) each convex region is defined as the set of points P that satisfy a multivariate polynomial of the form: $\sum_{i=1}^{lsvSize} factors[i] * P[i]^{factors[i]} = factors[lsvSize]$. When developing the code for a function that uses safe zones, one simply needs to: (i) Create a subclass of `SafeZoneFunction` and provide the code for the `F` method, (ii) Create a subclass of `NodeState` that provides the code for the `updateLSV` method, and (iii) Provide the method `computeSafeZones` that computes the safe zone to use whenever the estimate vector is updated. With this hierarchy, it is now possible to implement monitored conditions over functions with little implementation effort.

4.2.2 Mapping to the FERARI Architecture

The Storm topology we derived in our basic example in Section 4.1, can also be used to instantiate the more powerful approach of our function hierarchy. As the FERARI interfaces are used to express the algorithm, we can map the required runtime adaption to our Storm topology. New input data from the monitored streams reach the local nodes by the `update` method and are directed to the function we monitor accordingly. Local violations are communicated with the coordinator by the `sendToCoordinator` method, which is being mapped to a `global grouping` between the `LocalBolt` and `CoordinatorBolt`. In the other direction, the coordinator provides the local nodes with an updated estimate vector by the method `sendToLocal`. This is translated on the Storm topology to an `all grouping` between the `CoordinatorBolt` and the `LocalBolts`. The appropriate calculations required by the geometric approach, update of `lsv`, `dv` and `estimate`. are invoked by the `handleFromCoordinator` and `handleFromLocal` respectively. The same Storm, topology with `LocalBolt`, `CoordinatorBolt` and the connecting streams can be used, as runitme for the geometric monitoring hierarchy. Therefore, this topology can act as FERARI runtime adaption template.

5. CONCUSIONS AND FUTURE DIRECTIONS

In this paper we proposed an architecture for distributed detection and processing of events that allows the separation of application specific code from runtime dependent code. This is achieved by the introduction of the FERARI interfaces, which allow us to create applications for different execution environments. We evaluated established open source distributed streaming platforms, Akka, Spark, Storm and Trident and found that Storm best suits our requirements for the distributed monitoring scenario. We demonstrated how the proposed architecture can be used to implement an interesting example, monitoring when artists get popular by analyzing a stream of listen events. Additionally, we described how the powerful geometric monitoring approach can be implemented using our architecture while requiring tiny programming development efforts. Our code, in advance is public and available on-line [4].

Our future direction is to provide additional FERARI building blocks. Especially, we are interested in including the distributed online learning framework recently published in [6]. Another important block we are working on within the consortium is in providing the capabilities of the PROTON engine, that is open sourced by our partner IBM [5]. Finally, we work on creating a query planer, that will allow to formulate and dynamically optimize the monitoring task in a very convenient way.

6. ACKNOWLEDGMENTS

This research has been supported by the EU FP7-ICT-2013-11 under grant 619491 (FERARI).

7. REFERENCES

[1] Akka, `http://akka.io/`.
[2] Apache spark, `https://spark.apache.org/streaming/`.
[3] Apache storm, `http://storm.apache.org/`.
[4] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.
[5] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends databases*, 4(1–3):1–294, Jan. 2012.
[6] M. Kamp, M. Boley, D. Keren, A. Schuster, and I. Sharfman. Communication–efficient distributed online prediction by dynamic model synchronization. In *ECML PKDD*, 2014.
[7] D. Keren, G. Sagy, A. Abboud, D. Ben-David, A. Schuster, I. Sharfman, and A. Deligiannakis. Geometric monitoring of heterogeneous streams. *IEEE TKDE*, 26(8):1890–1903, 2014.
[8] D. Keren, I. Sharfman, A. Schuster, and A. Livne. Shape sensitive geometric monitoring. *IEEE TKDE*, 24(8):1520–1535, 2012.
[9] A. Lazerson, I. Sharfman, D. Keren, A. Schuster, M. Garofalakis, and V. Samoladas. Monitoring Distributed Streams using Convex Decomposition. *VLDB*, 2015 (to appear).
[10] G. Sagy, D. Keren, I. Sharfman, and A. Schuster. Distributed threshold querying of general functions by a difference of monotonic representation. *Proc. VLDB Endow.*, 4, November 2010.
[11] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. In *SIGMOD*, 2006.
[12] I. Sharfman, A. Schuster, and D. Keren. Aggregate threshold queries in sensor networks. In *IPDPS*, 2007.
[13] I. Sharfman, A. Schuster, and D. Keren. Shape sensitive geometric monitoring. In *PODS*, 2008.
[14] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy. Storm@twitter. In *SIGMOD*, 2014.

[4] `https://bitbucket.org/sbothe-iais/ferari`

[5] `https://github.com/ishkin/Proton`

Latent Fault Detection With Unbalanced Workloads

Moshe Gabel
Technion – Israel Institute of Technology
Haifa 32000 Israel
mgabel@cs.technion.ac.il

Kento Sato*
Lawrence Livermore National Laboratory
Livermore, CA 94551 USA
kento@llnl.gov

Daniel Keren
Haifa University
Haifa 31905 Israel
dkeren@cs.haifa.ac.il

Satoshi Matsuoka
Tokyo Institute of Technology
Meguro-ku, Tokyo 152-8552
Japan
matsu@is.titech.ac.jp

Assaf Schuster
Technion – Israel Institute of Technology
Haifa 32000 Israel
assaf@cs.technion.ac.il

ABSTRACT

Big data means big datacenters, comprised of hundreds or thousands of machines. With so many machines, failures are commonplace. Failure detection is crucial: undetected failures may lead to data loss and outages.

Recent health monitoring approaches use anomaly detection to forecast failures – anomalous machines are considered to be at risk of future failures. Our previous work focused on detecting latent faults in large web services, which are often characterized by scale-out architecture where load is dynamically balanced. We proposed a robust and unsupervised latent fault detector for such systems, with statistical bounds on the rate of false positives. That detector, however, is unsuitable for applications without dynamic load balancing, such as statically-balanced key-value stores, Hadoop jobs, and supercomputer applications.

We describe an improved latent fault detection method for unbalanced workloads. It retains the advantages of our previous methods: it is unsupervised, robust to changes, and statistically sound. Moreover, the statistical bounds for the new method scale better with the number of machines, and so dramatically reduce the number of measurements needed. Preliminary evaluation on supercomputer logs shows that the new method is able to correctly predict some failures, while our previous methods completely fail in this setting.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Fault tolerance

Keywords

Anomaly detection, health monitoring, data mining

*This work was produced while K. Sato was at Tokyo Institute of Technology, Meguro-ku, Tokyo 152-8552 Japan.

1. INTRODUCTION

Recent years have seen an increasing demand for computing power and storage. Large scale applications – whether offline batch computations or modern web services and clouds – are implemented on top of large datacenters, comprised of thousands of machines or more. For large cloud services, it is unreasonable to assume that all machines are working properly and are well configured [29, 28]. Unnoticed failures may cause data loss and service outages. Similarly, modern supercomputers and high-performance clusters are increasingly comprised of more and more individual components (multiple CPUs, drives, and recently multiple GPUs [32]), resulting in higher failure rates [31, 1]. In such systems, failures may delay long computation, even to the point where little useful computation is being done [30, 31, 27].

Many current failure detectors model normal behavior from historical data. Modeling can be manual, by setting static thresholds [14], or automatic, using supervised machine learning [2]. Modeling from historical behavior is suboptimal, however [9]. Workload changes, data-dependent computations, and software updates render learned models inaccurate [12, 9]: static thresholds must be adjusted by domain experts, and machine learning models must be retrained from recent data. This retraining requires relabeling data as exhibiting normal and abnormal behavior – an expensive process. Furthermore, supervised techniques often only detect problems that have been foreseen or encountered before.

More recent approaches [17, 18, 19] focus on unsupervised methods (mainly anomaly detection) which require no labeling and less domain expertise. Within this context we focus on latent fault detection [9]. *Latent faults* are subtle behavior deviations that may indicate problems or misconfigurations. The aim is to catch unforeseen faults that "fly under the radar" of monitoring systems, before they manifest as machine or software failures. In our previous work [9] we proposed a statistical latent fault detection framework for web services. It is robust to software changes and workload fluctuations, and provides statistical bounds on the rate of false positives. Our evaluation showed that latent faults are common and can precede failures by days. We have also extended that work for distributed settings [8], where the goal is to reduce communication and computational load.

Despite its advantages, our existing latent fault detection framework, like many other anomaly detection techniques,

assumed that workload is dynamically distributed over identical machines. Though this setup is common in scale-out, replicated services, it is not always the case in every setting. First, some large-scale web services are statically balanced or simply poorly balanced. Consider, for example, a key-value store where keys are statically assigned to machines by a hash function. If commonly used keys fall on a small number of machines, these machines are much more heavily loaded than the rest. Similarly, parallel computation frameworks such as Hadoop generally use key values to partition loads, resulting in unbalanced workloads [22]. Finally, large scale computations in compute clusters may distribute work to nodes unequally, due to data locality or because there is no easy way to predict how data distribution affects computation.

Our previous work also required a large number of measurements for a single run of the detection algorithm. The statistical bound grew linearly weaker with the number of machines: the more machines, the larger the required time window.

This work proposes a statistical latent fault detection test for unbalanced workloads, making it more practical in settings such as supercomputers, and in other large scale systems whose computational workload is not necessarily balanced. It also reduces the window size from a full day (roughly 300 measurements) to minutes (4 measurements). Since the new bound scales much better as the number of monitored machines grows, the new detector is much more responsive to immediate changes. It can therefore be used to monitor large systems when rapid response is important.

A preliminary evaluation on historical metric and failure logs from the TSUBAME2 supercomputer[1] shows that the new detector is superior: while our previous latent fault detectors fail completely in this setting, the new detector can predict some failures several days in advance.

2. IMPROVED ANOMALY DETECTION

Our previous latent fault detection framework [9, 8] relied on several assumptions, which we now revisit. First, machines in the system are homogenous in terms of hardware, software infrastructure and running code. Second, the majority of machines are not faulty; in a large system, most machines perform well most of the time. Finally, the monitored system uses dynamic load balancing – on average, workload is distributed evenly across machines. Thus when measuring aggregated performance counters, we could expect healthy machines to exhibit the same behavior, on average. We wish to keep the first two assumptions, but avoid the third.

As with web services, machines in compute clusters are often homogenous for logistical reasons. Where they are not, it is often possible to cluster to a few distinct configurations, using hardware and job data. Indeed supercomputers have strict, almost uniform hardware. For example, TSUBAME2 has 6 types of nodes[2] with well-documented hardware configurations.

The second assumption is also quite reasonable; it is hard to imagine an expensive datacenter running for lengths of time with a majority of faulty machines.

We can no longer assume dynamic load balancing though.

[1] `http://www.gsic.titech.ac.jp/en/tsubame2` .
[2] `http://tsubame.gsic.titech.ac.jp/docs/guides/tsubame2/html_en/overview.html#computing-nodes` .

Table 1: Hypothetical machine measurements. Node D has anomalous memory and CPU usage.

Node	Reqs	Memory	DB	CPU
A	3	630	9	6
B	5	650	15	10
C	4	640	12	8
D	3	**740**	9	**15**
E	8	680	24	16
F	7	670	21	14
G	5	645	15	11

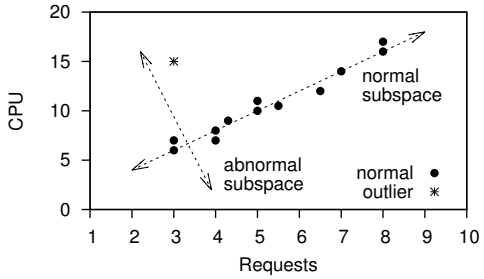


Figure 1: Scatterplot of the hypothetical requests vs CPU usage, with normal and abnormal subspaces. Machine D is visibly an outlier.

Instead, given our original assumption of homogenous machines running the same code, we assume there are common, inherent interdependencies, or correlations, between different counters, stemming from the fact that all machines do the same job, and run the same code. These correlations (not necessarily linear) between sets of counters are the result of what the machines are doing, and are not affected by workload. For instance, suppose task A requires k of some resource X and n of some resource Y for each unit of work. Increasing the workload to 5 units of work will require $5k$ of X and $5n$ of Y, but the correlation between k and n remains; it will remain even if we don't measure the actual workload. Generalizing to dependencies that may involve more than two counters, we can instead discuss correlations or relationships between or within sets of counters; for example, $2k + 3n + l^2 = m$.

Given the first two basic assumptions and the above, we can assume that similar machines doing the same task will result in the same correlations (relationships) between sets of counter values. The counters of machines with latent faults will not exhibit the same relationships.

For example, consider a hypothetical web service where for each client request we need 10MB of memory, 3 database transactions, and 2% CPU time. Machines with latent faults might have too few DB transactions, or too much memory use, or CPU usage that doesn't match the workload. Table 1 shows 5 such hypothetical machines. Machines A, B, C and E all exhibit the expected relationship between their counter values. Machine D, however, is anomalous, because its memory and CPU usage are far too high for its workload of 3 requests, for example due to a memory leak.

Our strategy is therefore to establish the linear correlations[3] within the aggregated counters at every time point,

[3] Not limited to the Pearson correlation, which is pairwise.

and find machines whose counters consistently (across several time points) exhibit different correlations.

2.1 PCA Subspace Decomposition

We will use Principal Component Analysis subspace decomposition [6, 23] to decompose the space of counters into a normal subspace and an abnormal subspace.

PCA is a statistical technique commonly used to automatically choose a smaller set of dimensions – the principal components – which captures most of the data variance. Since this subspace captures the variance in normal data, we can refer to it as the *normal subspace*. This normal subspace represents normal (healthy) linear correlations between the counters.

The residual components, on the other hand, define the complementary subspace that captures very little variance. In other words, when projecting a normal data point to the residual subspace, we can expect the projected vector to be very small, close to zero. The residual subspace is therefore the *abnormal subspace*, which represents violations of healthy relationships between counters.

Subsections 2.2 and 2.3 describe two variations using this basic idea. In the first variant, data vectors are projected into the abnormal subspace, and those vectors whose projection is above some threshold are declared to be abnormal. Jackson and Mudholkar [15] developed a way to infer the threshold from the data to guarantee a desired false positive rate. While their guarantee is only for multivariate Gaussian distributions, in practice the threshold is known to be robust even when the data is not Gaussian [16, 35]. Alternatively, we can apply the latent fault statistical framework [7] as is, using the projection to the abnormal subspace (normalized by the vector length) as the score function $S(m, x(t))$.

Figure 1 illustrates the technique. It shows requests vs. CPU usage of hypothetical machines from Table 1, including some additional healthy machines. The normal subspace is represented by the line $Y = 2X$, where X is requests and Y is CPU usage. The abnormal subspace is the perpendicular line. The outlier machine D clearly has a large presence in the abnormal subspace – projecting D's data to this space results in a large vector.

Ordinarily, historical data that is guaranteed to be "normal" is used to learn the normal and abnormal subspaces [23, 35]. In our case, we wish to detect small problems, and to support complex systems where we do not have a guaranteed error-free history. Instead we will make use of the large number of machines in the system. Since we assume most machines are fine at any given point in time, we can use this to extract the normal and abnormal subspaces.

One wrinkle in our plan is the presence of outliers in our data. Since we assume a small number of faulty machines (outliers), the resulting subspaces will include their faulty data. We therefore use a robust approach to PCA called HR-PCA, described by Xu et al. [34]. It is robust to outliers and arbitrarily corrupted data, and can recover the principal subspace even when the number of counters approaches the number of machines ($C \approx M$). HR-PCA is also quite efficient.

2.2 Formalizing the Algorithm

There are M machines, performing identical tasks, each periodically reporting C aggregated performance counters in a time window of length T. We standardize counter values in the time window across all machines to zero mean and unit variance in the time window. We denote by $x(m,t)$ the vector of standardized counter values for machine m at time t, and by $x(t) = \bigcup_m x(m,t)$ their union. Denote by X the $M \times C$ data matrix $x(t)$ after pre-processing and scaling at time t. Denote by x_m the row in X that came from machine m, meaning $x_m = x(m,t)$.

Using PCA we extract the normal subspace of X, comprised as the first k principal components $v_1, \ldots v_k$ that capture the most variance (say 95%). Denote by H_{no} the $C \times K$ normal subspace projection matrix built from the first k principal components, $H_{no} = [v_1, v_2, \ldots, v_k]$. Let the abnormal subspace projection matrix be the residual subspace $H_{ab} = I - H_{no}H_{no}^T$.

Given the projection H_{ab}, we can then map each machine vector $x(m,t)$ to its residual: $\tilde{x}_m = H_{ab}x_m$. Using the test statistic and threshold given in [15], define: $Q_m = \|\tilde{x}_m\|^2 = \|H_{ab}x_m\|^2$. A machine is declared abnormal at time t if $Q_m > Q_\alpha$, where Q_α denotes the threshold for the $1-\alpha$ confidence level:

$$Q_\alpha = \phi_1 \left[\frac{c_\alpha \sqrt{2\phi_2 h_0^2}}{\phi_1} + 1 + \frac{\phi_2 h_0 (h_0 - 1)}{\phi_1^2} \right]^{\frac{1}{h_0}} \quad ,$$

where

$$h_0 = 1 - \frac{2\phi_1\phi_3}{3\phi_2^2} \quad , \quad \phi_i = \sum_{j=k+1}^{C} \lambda_j^i \quad ,$$

λ_j is the variance captured by the j-th principal component, and c_α is the upper $1-\alpha$ percentile of the standard normal distribution. For a normal machine, $\Pr[Q_m > Q_\alpha] < \alpha$. In other words, α is the false alarm probability when testing a *single* machine.

Detecting one abnormal machine at time t is not sufficient, however. We are testing multiple machines, and must therefore guard against false positives. Hence we will only flag a machine if it is abnormal for T' consecutive times.

How big must T' be to guarantee a false alarm probability p when testing M machines? The probability of a false alarm for a specific machine m in T' consecutive time points is $\alpha^{T'}$. The false alarm probability in at least one machine after T' time points is therefore $1 - \left(1 - \alpha^{T'}\right)^M$, and so we require: $1 - \left(1 - \alpha^{T'}\right)^M \leq p$.

Thus for a desired false alarm probability p with M machines, we need a window size of:

$$T' = \left\lceil \log_\alpha \left(1 - \sqrt[M]{1-p}\right) \right\rceil . \qquad (1)$$

Note that the probability of false alarms drops roughly exponentially with T'. We discuss this below in Subsection 2.5.

The final algorithm for target false probabilities p and α:

1. Preprocess: select counters and scale to unit variance.
2. For each time t across T' consecutive times:
   - 2.1 Compute robust PCA (HR-PCA) from data $x(t)$.
   - 2.2 Choose k that captures most variance (say 95%).
   - 2.3 Build H_{no}, H_{ab}, Q_α.
   - 2.4 For each machine m, check if $Q_m > Q_\alpha$.
3. Report m if $Q_m > Q_\alpha$ for T' consecutive times.

2.3 Alternative to Thresholding

The threshold Q_α is determined from the actual data, and so may be too conservative. It is possible that, due to noisy

data, the resulting threshold is too high. The test is binary: $Q_m > Q_\alpha$ is either true or false; there is no middle ground. Hence, it is possible that even if Q_m is consistently high, much higher than the Q of other machines, it is still below the threshold. Our conservative design to limit false positives will result in too many false negatives, as few faulty machines are flagged.

Instead, we can use the statistical framework from [9, 7]. Let $S(m, x(t))$ be a *test*, a ranking function that assigns an *outlier score* (either a scalar or a vector) to machine m at time t. Given a test S, and desired false alarm probability $0 < \alpha < 1$, we can present the framework as follows:

1. Preprocess: select counters and scale to unit variance.
2. Compute for every machine m the vector: $v_m = \frac{1}{T}\sum_t S(m, x(t))$ (integration phase).
3. Compute the p-values (defined below) $p(m)$ from v_m.
4. Report every machine with $p(m) < \alpha$ as suspicious.

We use the normalized Q_m as the score function:

$$S(m, x(t)) = \frac{Q_m}{\|x_m\|^2} = \frac{\|\tilde{x}_m\|^2}{\|x_m\|^2} = \frac{\|H_{ab}x_m\|^2}{\|x_m\|^2} \quad .$$

We derive probabilistic bounds using the machinery from [7]. Note that $0 \le S(m, x(t)) \le 1$, thus even if we change all of $x(t)$ S cannot change by more than 1. Moreover, HR-PCA is robust, so changing just one vector $x(m, t)$ should not overtly affect H_{ab}, thus $Q_{m'}$ for any other machine m' should not change. Therefore S is $1, 0$-*bounded* [7, Definition 2.3.1], and we can apply [7, Lemma 2.3.3] to get a p-value:

$$p(m) = (M+1)\exp\left(-\frac{2TM\gamma^2}{\left(\sqrt{M}+1\right)^2}\right) \qquad (2)$$

where $\gamma = \max(0, \|v_m\| - \hat{v})$, and $\hat{v} = \frac{1}{M}\sum_m \|v_m\|$. This p-value is the probability that $\|v_m\|$ is larger than the mean $\hat{v}$ by γ when m is healthy, given that we are testing m machines; testing for $p(m) < \alpha$ guarantees false alarm probability α across all machines, equivalent to p in Subsection 2.2.

$p(m)$ is more flexible than Q_α – it is computed from data of T times (step 2, above), rather than tested each time separately. Consistently small deviations from the norm accumulate if T is large enough. The advantage of this "soft threshold" approach is that it is much more sensitive to smaller anomalies ($Q < Q_\alpha$) than the original "all or nothing" approach. The downside is that the improved window size described in Subsection 2.5 no longer applies.

2.4 Unbalanced Workloads and Robustness

Our previous framework required that workload be dynamically balanced, on average, across all machines. This was because we directly compared counter values between machines. Aggregating across a large time window helped us overcome, and take advantage of, temporal noise. Indeed, temporary workload imbalance is very likely, since it is difficult to guarantee that all machines are equally loaded for any short time interval. Averaged across a larger interval, these small random imbalances cancel each other out.

Conversely, the algorithm described above does not depend on the absolute counter values, but instead on the correlations between them at each point in time. We expect that these correlations will remain similar regardless of the load.

Table 2: Hypothetical machine measurements exhibiting unbalanced load.

Load	Reqs	Memory	DB	CPU
low	8	680	24	16
low	5	650	15	10
low	6	660	18	12
high	33	930	99	66
high	40	1000	120	80
high	37	970	111	74

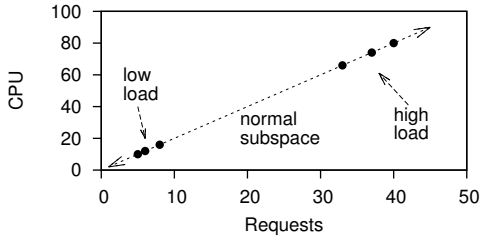


Figure 2: Scatterplot of the unbalanced hypothetical requests vs CPU usage. All machines lie on normal subspace.

For example, consider our hypothetical web service from above. For each client request, we need 10MB of memory, 3 database transactions, and 2% CPU time. Table 2 shows 6 such hypothetical machines. The first 3 machines are lightly loaded (few requests), while the last three are heavily loaded. Still, all exhibit the expected relationship between their counter values, and so lie on the normal subspace (Figure 2). A machine exhibiting anomalous CPU usage for the number of requests would lie outside this normal subspace.

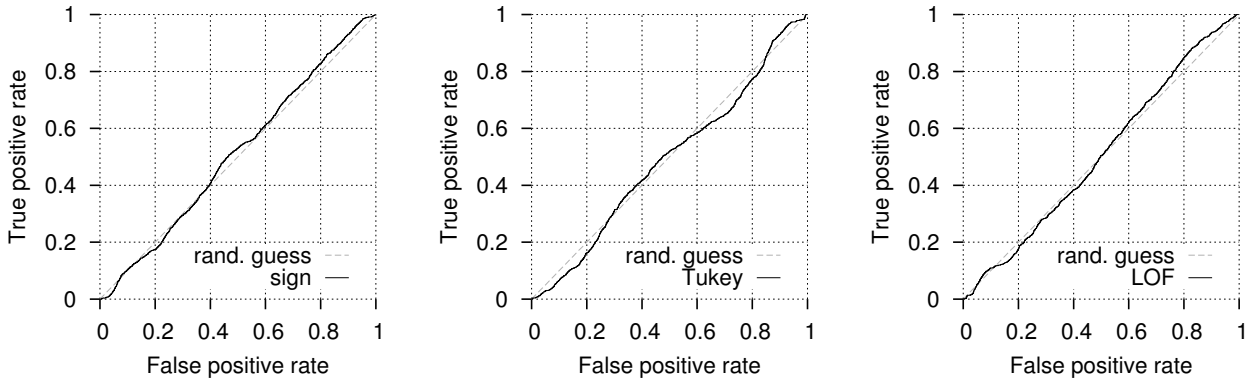
Moreover, as with our previous methods, the subspace decomposition approach is robust to changes in the monitored system. The normal and abnormal subspaces are recomputed using counter values measured at the same time, and we never compare such values across different times. If the software is updated, for example, the new behavior is never compared to the old one.

2.5 Improved Window Size

The bounds for our previous methods [9] required increasing window sizes as the number of machines grew. As illustrated by Equation (2), the framework bound grows linearly weaker with the number of machines M, meaning that we have to increase the window size T to compensate.

This can be intuitively be explained by the need to aggregate different measurements across many times to overcome temporal noise in counter values, such as short-term workload imbalance. The experiments described in [9] were performed with window size of $T = 288$, which translated to a full 24-hour day since counters were sampled every 5 minutes.

The PCA method has an improved window size. T' from Equation (1) is logarithmic in the number of machines M. The intuitive explanation is that we no longer need to track counter behavior across time to overcome minute random imbalances or random noise. For example, given $M = 10000$ machines, $\alpha = 0.01$, and overall false alarm probability $p = 0.01$, Equation (1) tells us we need a window size of only



(a) Sign test (b) Tukey test (c) LOF test

Figure 3: Performance of original latent fault detector on TSUBAME2 data with 7 day horizon.

3 time points, i.e., 15 minutes:

$$T' = \lceil \log_{0.01} \left(1 - \sqrt[10000]{1 - 0.01}\right) \rceil = \lceil 2.9989 \rceil = 3 \quad .$$

Even for a million machines, $M = 10^6$, a window size of 4 time points, meaning 20 minutes, is sufficient to guarantee a false positive rate of 0.01:

$$T' = \lceil \log_{0.01} \left(1 - \sqrt[10^6]{1 - 0.01}\right) \rceil = \lceil 3.9989 \rceil = 4 \quad .$$

3. PRELIMINARY EVALUATION

We used historical machine metric logs and failure records from the TSUBAME2 supercomputer to compare the new subspace decomposition approach to our existing latent fault detectors [9].

3.1 Supercomputer Workloads

Supercomputer workloads are very different from cloud workloads in many ways: long jobs rather than short requests; parallelization and load balancing are done within single jobs, not over all requests; and jobs are heterogeneous, so different nodes do not run the same code at the same time. Thus, the job uniformity assumption we make in Section 2 and our previous work [9] may no longer hold.

Computational jobs often perform many iterations of the same basic loop [30, 31]. For computations whose performance is not data-dependant (such as many common matrix operations), a single computation iteration will usually require the same amount of resources (CPU time, GPU load, etc.) as any other iteration. Moreover, required resources for one iteration will be the same for all nodes in the system with the same hardware configuration. This essentially brings us back to the PCA approach suggested in Section 2 – metrics of healthy nodes will lie in the same subspace.

Latent fault detection tests should be run on groups of machines partitioned by job. Scheduling logs that contain start and end times (along with the list of assigned machines) can be used to subdivide machines in this way.

3.2 The TSUBAME2 Dataset

We used 45 common machine metrics (e.g., cpu idle time, GPU utilization, user time, swap free, various temperatures), sampled every 1 to 10 minutes (depending on the metric), from one month of runs (roughly jobs, see below). We divided each run of 240 minutes or more, with at least 10 machines, to windows of length 240 minutes each. This resulted in 60 runs with a total of 252 windows, summarized in Table 3. We performed latent fault detection on each such window. The results of the detector were compared with the historical failure log within a 7 day horizon – a node is considered to have failed if it failed within 7 days from the time window; otherwise it is considered to have not failed.

Table 3: Statistics of inferred jobs (runs).

Statistic	Median	Max
Number of machines (M)	99	236
Length (minutes)	1016	5699

Since our TSUBAME2 logs did not include any job scheduling information, our preliminary experiments relied on CPU and GPU usage metrics to infer which machines were being used and how they were grouped. A group of machines that together became busy and then idle were considered to be a single job, or a "run". This method is ad-hoc and inherently inaccurate. For example, a failing machine might stop at the beginning of the computation and so would never be considered part of the run, as it did not finish with the rest. Section 5 discusses a potentially more robust alternative.

3.3 Results

We first evaluated the performance of our existing latent fault detectors: the sign, LOF and Tukey tests [9], with $T = 240$ and $\alpha = 0.01$. Figure 3 shows the receiver operating characteristic (ROC) curves for our existing latent fault detection tests. Ignoring computed p-values, we swept the threshold for anomaly scores $\|v_m\|$ (as in Eq. (2)) across a range of values and drew the resulting false positive and false negative rates. The performance of all three previous tests is no better than a random guess, where the false positive and false negative rates are equal.

We repeated the tests with the PCA approach suggested in Section 2, using the "soft threshold" variation described in Subsection 2.3. We used $T = 240$ and $\alpha = 0.01$. k was selected to capture 95% of variance. Finally, HR-PCA [34] was used as the robust PCA building block, with the maximum number of corrupted points set to 10% of machines

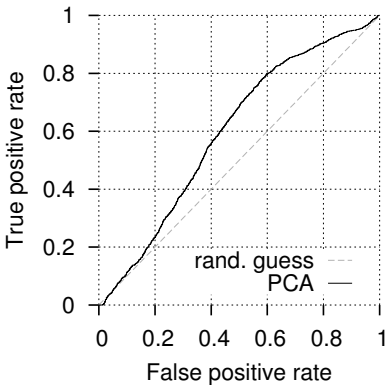


Figure 4: Performance of PCA latent fault detector on TSUBAME2 data with 7 day horizon.

($\hat{t} = 0.9M$). As can be seen in Figure 4, the subspace decomposition approach performs better than our original latent fault detectors. Though still no better than random guess at lower false positive rates, it is still able to predict some node failures several days ahead.

4. RELATED WORK

PCA residuals have been used in the past for monitoring tasks [6]. Lakhina et al. [23] famously used this approach to detect network traffic anomalies. Like many similar approaches, they monitor the network as a whole, and do not attempt to localize it to a specific node. Furthermore, they rely on historical data that is guaranteed to be normal, and assume that the system is unchanged, again a common theme. Xu et al. [35] analyze program source code to parse console log messages and use principal component analysis to identify unusual message patterns based on their frequency. As with Lakhina's work, this technique relies on error-free history and relatively stable systems. Console logs also tend to contain different sorts of data, and are likely to catch different sorts of anomalies. Similarly, Chen et al. [3] localize failures in software components of a Java application; they propose an online algorithm to update normal and abnormal behavior models. Chen et al. [4] also analyze the correlation between sets of measurements and track them over time. Their approach requires domain knowledge for choosing counters, and requires training on "healthy" periods to model baseline correlations.

Ling et al. [13] use Stochastic Matrix Perturbation theory to adapt Lakhina's work to distributed monitoring with PCA. Liu et al. [26], in turn, apply the distributed PCA monitoring approach on linear sketches of the network data to reduce running time and space costs.

5. CONCLUSIONS AND FUTURE WORK

Failure detection and prediction techniques are increasingly important in the era of clouds and compute clusters. Several recent approaches rely on anomaly detection techniques to detect failures ahead of time, while avoiding costly relabeling and retraining of models.

In this work we have presented a new latent fault detector suitable for settings where the workload is unbalanced. As with our previous methods, the new approach is robust to changes in the monitored system, it requires neither domain expertise nor labeled data, and it comes with statistical guarantees on the rate of false positives. Our preliminary evaluation showed that the new approach is clearly superior to the previous latent fault detectors in the supercomputer setting. Though the results obtained may not yet be practical in the supercomputer setting (possibly due to lack of scheduling logs), they show that the new detector does cope with unbalanced loads.

There are several avenues to pursue: more complete evaluation, communication-efficient and computation-efficient algorithms, and subspace clustering for job detection.

First, we wish to evaluate the new approach in additional settings where the workload is unbalanced. In the cloud setting, key-value stores are good candidates for our monitoring approach. In the compute cluster setting, Hadoop jobs have many machines running the same code in the reduce phase, but their computational load may be different.

Second, we can further combine PCA with distributed online detection as in [8], since collecting metrics from all nodes and computing PCA may be prohibitive for some large systems. We previously described [8] a communication-efficient approach using safe zones [21, 20] to standardize counter values across machines. Ling et al. [13] adapt PCA anomaly detection for distributed stream monitoring. We can combine their technique with recent distributed monitoring approaches [24, 11]. Beyond that, we can follow Liu et al. [26], who apply the distributed PCA technique on a linear sketch. Recent work by Liberty [25] introduces a better matrix sketching technique called Frequent Directions with improved bounds that is well-suited for PCA computation in a streaming setting. Indeed, streaming constructions of PCA using this approach are proposed by Ghashami and Phillips [10] and Cohen et al. [5].

Finally, more complete job and scheduling information in the supercomputer setting may help us improve results even further. A more robust approach to job detection might be *subspace clustering* [33]. Given a collection of vectors drawn from a union of (potentially disjoint) subspaces, subspace clustering algorithms cluster these data points according to the subspace they originate from. For similar reasons described in Section 2, and since hardware is uniform, we could assume that metrics of machines running the same job will lie in the same subspace. Thus, we might be able to use subspace clustering to identify machines that run similar code. Given such a group of machines, our anomaly detection techniques can be more effective. Moreover, the ability to cluster running code can be useful in monitoring settings such as virtual machine clouds, where operators have less information on what code runs inside virtual machines.

6. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme FP7-ICT-2013-11 under grant agreement Nº 619491 and Nº 619435. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-665287). This work was also supported by Grant-in-Aid for Research Fellow of the Japan Society for the Promotion of Science (JSPS Fellows) 24008253, and Grant-in-Aid for Scientific Research S 23220003.

7. REFERENCES

[1] G. Bronevetsky, I. Laguna, S. Bagchi, B. R. de Supinski, D. H. Ahn, and M. Schulz. Statistical fault detection for parallel applications with AutomaDeD. In *Proc. SELSE*, 2010.

[2] G. Bronevetsky, I. Laguna, B. De Supinski, and S. Bagchi. Automatic fault characterization via abnormality-enhanced classification. In *Proc. DSN*, 2012.

[3] H. Chen, G. Jiang, C. Ungureanu, and K. Yoshihira. Failure detection and localization in component based systems by online tracking. In *Proc. KDD*, 2005.

[4] H. Chen, G. Jiang, and K. Yoshihira. Failure detection in large-scale internet services by principal subspace mapping. *IEEE Trans. Knowl. Data Eng.*, 2007.

[5] M. B. Cohen, S. Elder, C. Musco, C. Musco, and M. Persu. Dimensionality reduction for k-means clustering and low rank approximation. *CoRR*, 2014.

[6] R. Dunia and S. J. Qin. Multi-dimensional fault diagnosis using a subspace approach. In *Proc. ACC*, 1997.

[7] M. Gabel. Unsupervised anomaly detection in large datacenters. Master's thesis, Technion I.I.T, 2013.

[8] M. Gabel, D. Keren, and A. Schuster. Communication-efficient distributed variance monitoring and outlier detection for multivariate time series. In *Proc. IPDPS*, 2014.

[9] M. Gabel, A. Schuster, R.-G. Bachrach, and N. Bjorner. Latent fault detection in large scale services. In *Proc. DSN*, 2012.

[10] M. Ghashami and J. M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *Proc. SODA*. 2014.

[11] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, and A. Schuster. Distributed geometric query monitoring using prediction models. *ACM Trans. Database Syst.*, 2014.

[12] C. Huang, I. Cohen, J. Symons, and T. Abdelzaher. Achieving scalable automated diagnosis of distributed systems performance problems. Technical report, HP Labs, 2007.

[13] L. Huang, X. Nguyen, M. Garofalakis, J. Hellerstein, M. Jordan, A. Joseph, and N. Taft. Communication-efficient online detection of network-wide anomalies. In *Proc. INFOCOM*, 2007.

[14] M. Isard. Autopilot: automatic data center management. *SIGOPS Oper. Syst. Rev.*, 2007.

[15] J. E. Jackson and G. S. Mudholkar. Control procedures for residuals associated with principal component analysis. *Technometrics*, 1979.

[16] D. R. Jensen and H. Solomon. A gaussian approximation to the distribution of a definite quadratic form. *Journal of the American Statistical Association*, 1972.

[17] S. Kadirvel, J. Ho, and J. A. B. Fortes. Fault management in Map-Reduce through early detection of anomalous nodes. In *Proc. ICAC*, 2013.

[18] S. Kavulya, S. Daniels, K. Joshi, M. Hiltunen, R. Gandhi, and P. Narasimhan. Draco: Statistical diagnosis of chronic problems in large distributed systems. In *Proc. DSN*, 2012.

[19] S. Kavulya, R. Gandhi, and P. Narasimhan. Gumshoe: Diagnosing performance problems in replicated file-systems. In *Proc. SRDS*, 2008.

[20] D. Keren, G. Sagy, A. Abboud, D. Ben-David, A. Schuster, I. Sharfman, and A. Deligiannakis. Geometric monitoring of heterogeneous streams. *Trans. on Knowl. and Data Eng.*, 2014.

[21] D. Keren, I. Sharfman, A. Schuster, and A. Livne. Shape sensitive geometric monitoring. *Trans. Knowl. Data Eng.*, 2012.

[22] Y. Kwon, K. Ren, M. Balazinska, and B. Howe. Managing skew in Hadoop. *IEEE Data Eng. Bull.*, 2013.

[23] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proc. SIGCOMM*, 2004.

[24] A. Lazerson, I. Sharfman, D. Keren, A. Schuster, M. Garofalakis, and V. Samoladas. Monitoring distributed streams using convex decompositions. In *Proc. VLDB*, 2015. To appear.

[25] E. Liberty. Simple and deterministic matrix sketching. In *Proc. KDD*, 2013.

[26] Y. Liu, L. Zhang, and Y. Guan. A distributed data streaming algorithm for network-wide traffic anomaly detection. *SIGMETRICS*, 2009.

[27] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proc. SC*, 2010.

[28] E. B. Nightingale, J. R. Douceur, and V. Orgovan. Cycles, cells and platters: An empirical analysis of hardware failures on a million consumer pcs. In *Proc. EuroSys*, 2011.

[29] N. Palatin, A. Leizarowitz, A. Schuster, and R. Wolff. Mining for misconfigured machines in grid systems. In *Proc. SIGKDD*, 2006.

[30] K. Sato, N. Maruyama, K. Mohror, A. Moody, T. Gamblin, B. R. de Supinski, and S. Matsuoka. Design and modeling of a non-blocking checkpointing system. In *Proc. SC*, 2012.

[31] K. Sato, A. Moody, K. Mohror, T. Gamblin, B. R. d. Supinski, N. Maruyama, and S. Matsuoka. FMI: Fault tolerant messaging interface for fast and transparent recovery. In *Proc. IPDPS*, 2014.

[32] U. Verner, A. Schuster, and M. Silberstein. Processing data streams with hard real-time constraints on heterogeneous systems. In *Proc. ICS*, 2011.

[33] R. Vidal. Subspace clustering. *IEEE Signal Processing Magazine*, 2011.

[34] H. Xu, C. Caramanis, and S. Mannor. Outlier-robust PCA: The high-dimensional case. *IEEE Transactions on Information Theory*, 2013.

[35] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting large-scale system problems by mining console logs. In *Proc. SOSP*, 2009.

What You See Is What You Do: applying Ecological Interface Design to Visual Analytics

Natan Morar
University of Birmingham
School of Electronic, Electrical and Computer Engineering, UK
0044 741 472 6845
nsm120@bham.ac.uk

Chris Baber
University of Birmingham
School of Electronic, Electrical and Computer Engineering, UK
0044 121 414 3965
c.baber@bham.ac.uk

Peter Bak
IBM Research Lab
Haifa / Israel
00972 4829 6537
peter.bak@il.ibm.com

Adam Duncan
University of Birmingham
School of Electronic, Electrical and Computer Engineering, UK
axd174@student.bham.ac.uk

ABSTRACT
In the SPEEDD project, we are developing approaches to the design and evaluation of Visual Analytics which are informed by Human Factors theories and methods. As part of this process, we are using the concept of Allocation of Function to inform the design of User Interfaces for Visual Analytics. The paper presents a case study of the development of a Road Traffic Management User Interface.

ACM Classification Keywords
H.1.2 User/machine systems; H.5.2 User interfaces

Keywords
Cognitive Work Analysis; Ecological Interface Design; Visual Analytics; Human Factors.

1. INTRODUCTION
Visual Analytics combines the power of data analytics with the insight and imagination of the human operator in response to the visualization of the output of these data analytics. In terms of output, visualization can be applied before the analysis on raw data (data visualization), or on output results (information visualization), or during the analysis phase (visual data mining), or on any combination of these [1]. This division of labor between an automated system, which mines massive data sets, and a human decision maker, who interprets the recommendations of the analytics, can be considered as Allocation of Function. One could allocate the analysis functions to the automation, leaving the human as the passive consumer of the system's outputs, and merely accepting the system's recommendations; anyone who has watched 'The Simpsons' will recall Homer pressing the 'any key' to confirm system status. Not only does the relegation of the human to an acceptor of system recommendations miss the point of the Visual Analytics concept, but it also removes the human operator from the analysis loop. A consequence of removing the person is that this can impair the person's ability to understand the meaning of the data, to interpret the system's recommendation or to intervene appropriately when required [2]. Furthermore, users tend to have more trust in their findings when involved in the discovery process than when the findings come from an automated system [1]. Thus, it would make sense to design the Allocation of Function between human and automation in such a way as to ensure both partners worked to their best potential. In general, the storage, transformation and processing of data is more suited to automatic systems, whereas hypotheses generation and interpretation of findings are considered more human led tasks [3].

2. ALLOCATION OF FUNCTION
Determining whether a particular function (in terms of system operation) should be performed by automation or human operator is known as Allocation of Function. While some functions (such as dealing with massive data sets) are clearly suite to automation and others (such as gaining insight from a collection of data) might be more suited to human operators, the challenge of Allocation of Function stems from the fact that some of the functions could be performed equally well by automation or human operator. Further, the way a function is performed is likely to change as a result of the task context. Thus, adaptive automation (in which Allocation of Function varies according to context) can improve operator ability in intervening in response to errors [4, 5, 6]. Moreover, by dynamically allocating tasks to either the user or the automated system user skills can be maintained [7.].

In this paper, we are interested in the question of whether it is possible to manage Allocation of Function through the visualization. In other words, the User Interface could indicate to the operator when and how they could intervene at particular stages in the process.

As the application of this work is linked to the traffic management use case, we start by presenting the requirements of the system as highlighted by traffic operators in our discussions with them. These requirements, in combination with the Cognitive Work Analysis (presented in the following section), informed the initial design of the User Interface (Figure 1). Following this, in order to appreciate how Allocation of Function might be applied to this use case, we turn our attention to the question of Situation Awareness and the design of Ecological Interfaces.

2.1 Requirements for Traffic Management Use Case

- Allow Operator to clarify and query notification
- Allow Operator to draw on experience of previous incidents
- Allow Operator to select Incident Type option
- Allow Operator to draw on several sources of information to confirm location
- Support Operator Situation Awareness
    - Of current incident
    - Of future conditions
- Allow Operator selection of response
- Allow Operator to challenge or negotiate response
- Support Operators in gaining Global and Local Situation Awareness of road user behaviour
- Supporting Operators in determining that the incident has no unexpected consequences.

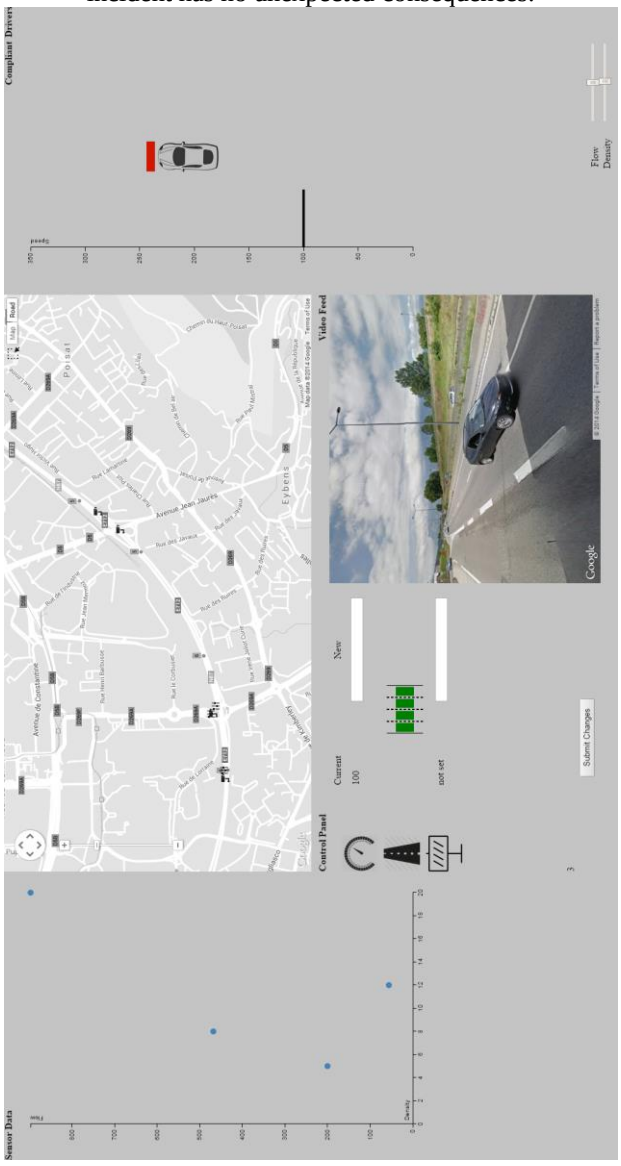


Figure 1: SPEEDD Initial User Interface for Road Traffic Management Use Case v1.0

3. COGNITIVE WORK ANALYSIS OF ROAD TRAFFIC MANAGEMENT OPERATIONS

CWA, Cognitive Work Analysis [10, 11, 12], involves a number of phases, each of which contributes to an understanding of how stakeholders *could* work with a given system. In this way, the problem space represented by the system can be explored in order to determine ways of supporting activity in that space. Figure 2 shows the Abstraction Hierarchy from CWA of Road Traffic Management; we have used the phrase 'manage road network' as the Functional Purpose of the system.

Having defined a Functional Purpose, the next step is to define the Value and Priority Measures of the system (the second row of Figure 2). These represent those aspects of performance that the system could use to indicate how well it is performing. Through observation and interviews, we defined the following aspects:

- To ensure minimal congestion in the road network
- To ensure minimal risk to road users
- To enable minimal journey times for road users
- To ensure informed road users
- To support maintained infrastructure
- To encourage compliant road users
- To support immediate response to incidents
- To produce an auditable record of activity

These aspects map on to the generally accepted set of objectives for traffic management [13.]:

- Maximize the available capacity of the roadway system
- Minimize the impact of incidents
- Contribute to demand regulation
- Assist in the provision of emergency services
- Maintain public confidence in operations and information provision

The main difference between the two sets concerns the issues of providing support to the emergency services (although we have 'immediate response to incident' which we suggest would include this), and maintaining public confidence in control center provision (which we do not include but which could relate to the priority for ensuring 'compliant road use').

In the control room environment (in the scope of the SPEEDD project), because of the introduction of the automated system, the goals (derived from CWA) are shared between the two entities – the operator and the automation. Each entity contributes to achieving the system goals through different means (see Figure 3). Besides the operators' role to deal with uncertainty and spot errors in the data and analysis outputs, the system should allow them

to inform (train) automation. The latter can be achieved through the action of overriding the automation outputs.

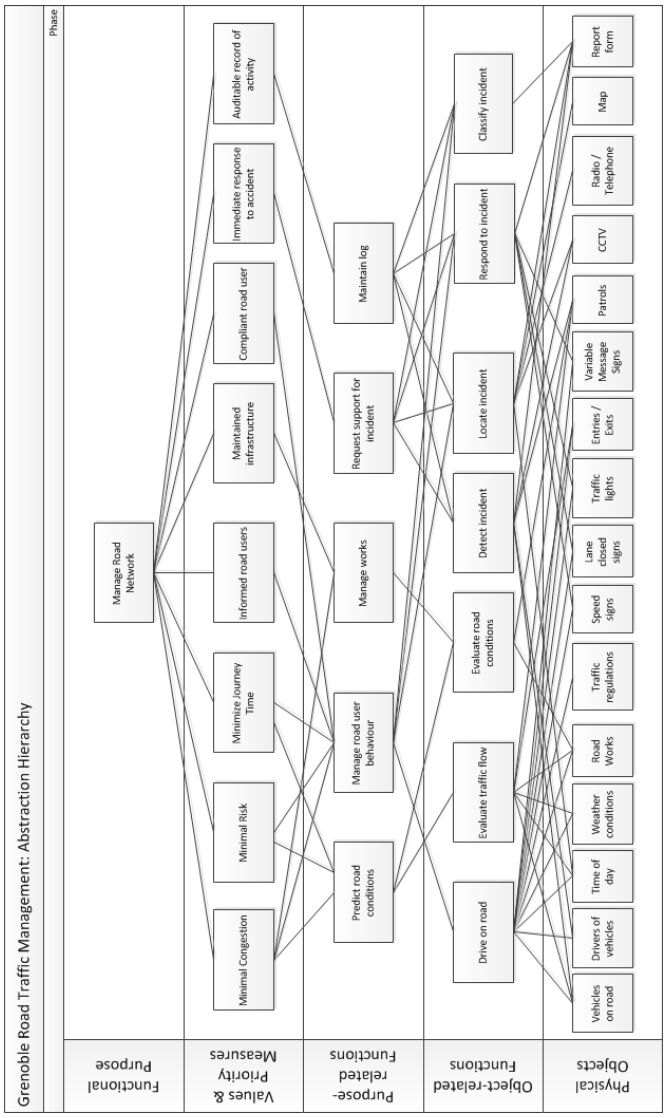


Figure 2: CWA Abstraction Hierarchy

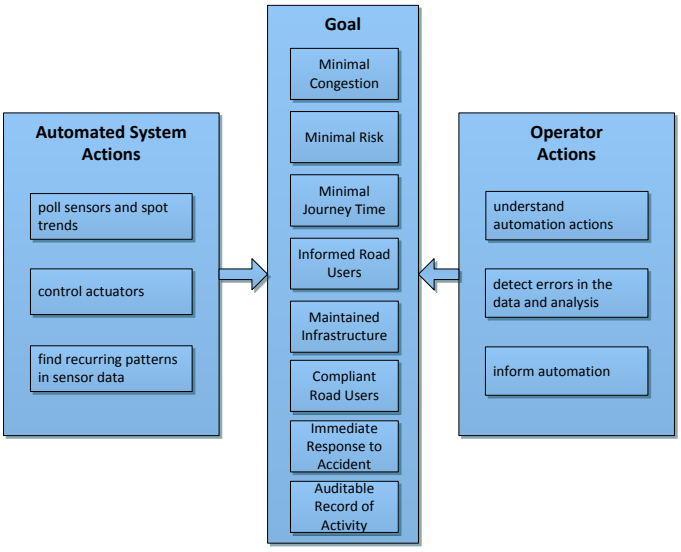


Figure 3: Contribution of System Components to System Goals

As in any Socio-Technical System, there will be a range of actors who will perform functions in order for the system to achieve its Functional Purpose. For instance, apart from automation and operators, there will be the individual road users who are driving vehicles through the road network and whose behavior the operators in a control room are seeking to influence. In addition, there might be specialized roles, dedicated to maintaining the infrastructure of the road network or to dealing with accidents and incidents, which are called upon at specific times. Figure 4 takes the Object-related Functions (from Figure 2) and shows how these can be performed by different actors (shown by color coding) and in different circumstances. In this Figure, the circumstances are presented as examples of different 'modes' in which the system could be assumed to operate, i.e., normal conditions (managed roads), disrupted conditions (response to incidents), or scheduled disruptions (planned works). Figure 4 shows how the different circumstances can lead to different distribution of these object-related functions across the range of actors.

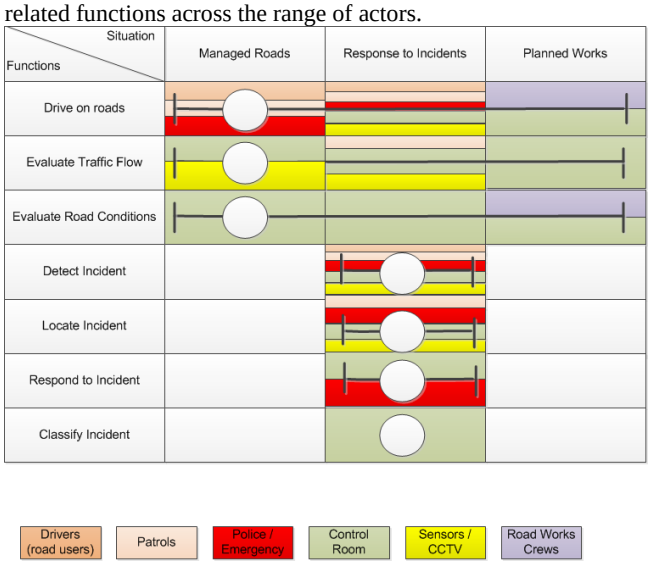


Figure 4: CWA SOCA

The Object-Related Functions in Figure 2 represent a form of task analysis. The 'decision ladder' in Figure 5 should be read from the bottom left (beginning with an input to the operator) up to the top (Functional Purpose, or overall goal of the operator / system). From the Functional Purpose, the right-hand leg of the ladder descends to the action that the operator will make. There are various 'short-cuts' that the experienced operator might apply (indicated by dotted lines), perhaps in light of particular patterns of data or reports from previous responses.

Figure 5: CWA 'Decision ladder'

4. DESIGN CONCEPTS AND IMPLEMENTATION

At the top level, the Abstraction Hierarchy (Figure 2) presents the Functional Purpose of the system. For example, if the system is intended to respond to incidents quickly, then this display could show the time spent responding to incidents, perhaps against targets or against historical data. The usefulness of such a display would depend on the nature of the work.

At the next level of the Abstraction Hierarchy, the Abstract Function (values and priorities) would be reflected by the parameters that the 'system' is seeking to balance. It can also be beneficial to present subgoals (Purpose-related functions), tasks (Object-related functions) or information sources (Physical objects). These can either provide cues for the operators to interact with the system at a lower level or can provide alternative means of alerting operators to change in system state. Thus, for example, the output of a CCTV (Physical Object) could be manipulated by the operator (Object-related function) in order to determine the location of an incident (Purpose-related function). In this situation, it might be useful for the operator to directly mark and record this information, say by marking this frame (and its associated metadata defining location, direction of view, time etc.) and capturing this directly into the report.

Table 1: Relating Information Requirements for different Stakeholders to the levels of Abstraction Hierarchy

	Control Room Operators	Road Works Crews	Drivers (Road users)
Domain purpose			
Values Priorities	Congestion Incident Record activity	Congestion Infrastructure	Congestion Risk Journey time Information
Knowledge semantics insight	Support to response Update log	Conditions {repair / works, weather, traffic flow / density, environmental}	Driver behaviour Compliance
Facts ideas opinions	Availability {signage, CCTV}	Availability {lane, road, exit / entry}	Movement Accident
Source objects	Location {signage, CCTV} Content {signage, CCTV}	Map	Vehicles Motion Compliance

.

In Table 1, relations are mapped and examples of the type of information which might be used in the system to support these relations are indicated. This provides a simple means of eliciting the information which *might* be useful for this system. Finally, taking the relationships defined in Table 1, we sketch the concept layout (Figure 6) for the User Interface.

4.1 Graphic Options

Figure 6 contains 8 regions. The following list outlines some of the options that are being considered in the design. Items in the list marked * correspond to existing information displays in the control room.

1. Road status (traffic conditions), e.g., displayed as a fundamental diagram. This could also compare current traffic conditions with the same time last week, or predicted traffic conditions and likely trends;
2. Values / trends / forecasts: this display could provide operators with views of the predicted traffic, or driver behavior, to allow comparison between alternative courses of action;
3. Road user goals: this display could indicate information which might be relevant to road user activity, for instance, alternative routes which drivers might take if there is congestion;

4. Driver behavior and compliance: this display could indicate how road users are behaving. This could include average speed in each lane or average distance between vehicles;
5. CCTV content / control*: this display would present the images from the selected CCTV camera to the operator, and allow the CCTV camera to be controlled;
6. Control activity, signage content*: this would show the actions that the operator is able to perform and the content which could be presented on variable message signs;
7. Log, open tasks, scheduled events*: this would show the log of the current incident that the operator is working on, together with open tasks or any scheduled events that need to be dealt with;
8. Map of road network*: displayed as a map of the ring road (either a schematic as in the current design or a more detailed map of Grenoble and the road network), with key Objects indicated, e.g., CCTV and sign locations, junction (ramps) etc. This could also be used to display the location of incidents, such as congestion.

5. SITUATION AWARENESS AND ECOLOGICAL INTERFACE DESIGN

For the operator, Situation Awareness involves selecting the most appropriate information source (or combination of sources) and then analyzing the information in order to make sense of the system being controlled. This raises questions such as what is the 'system' that is being worked with, and what constraints might affect interaction with this system. In other words, the focus of operator activity can be described in terms of the problem space in which humans make decisions, the sort of tasks and decisions that humans make, and the constraints which affect performance of these activities. Ecological Interface Design addresses these concerns [8.].

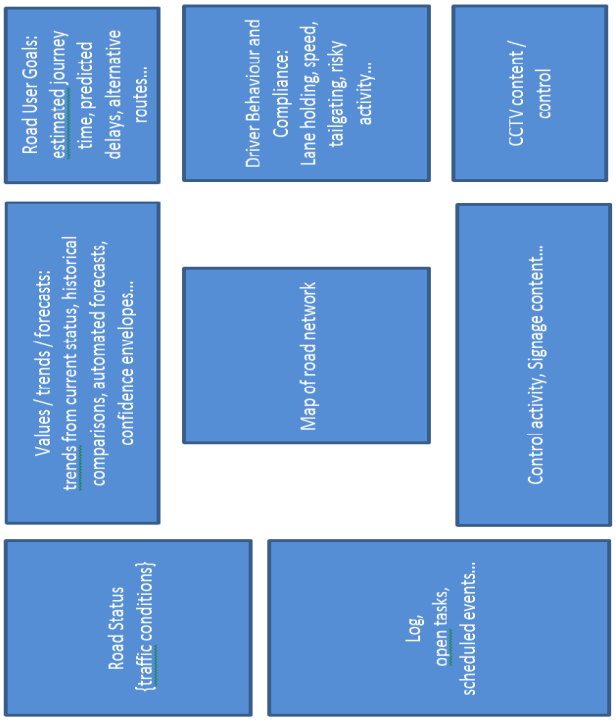


Figure 6: Schematic User Interface for Road Traffic Case

The concept of Ecological Interface Design (EID), developed from Cognitive Work Analysis (see next section), draws on Gibson's [9.] concept of direct perception (later encompassed by the 'ecological psychology' movement). For User Interface design, this leads to the assumption that people are able to perceive meaning of objects *directly* (i.e., with no need for cognitive intervention) when the situation in which they encounter those objects provide a suitable context for interpretation. A further assumption of EID is that the task constrains the ways in which information is interpreted and defined to be salient or meaningful. Within this 'task ecology', it is plausible to assume that different people will interpret the information in different ways (according to their current tasks, goals, experience and training). Thus, the 'task ecology' of a system is defined by the range of states in which it can develop and the constraints that these states place on people interacting with the system.

Relating Situation Awareness to EID, we might expect operators to be able to spot patterns in the data and then respond to these by selecting a course action. It is interesting to contrast guidance for the design of User Interfaces from the perspective of Situation Awareness with that presented for EID. As Table 2 shows, there are strong similarities between the approaches (even if the underlying theory and the terminology used differ). Both emphasize the benefit of 'direct' display of information and both imply the need to represent the system in terms of user goals and in terms of different levels of system operation and performance.

Table 2: Comparing EID and SA

Design for Situation Awareness	Ecological Interface Design
Relate to operator's major goals	Represent function and meaning in the task ecology
Present information directly	Design to support direct perception of visual information
Assist system projection	
Display global status	Reveal underlying system process and constraints
Support global-local trade-offs	
Support perception-action schemata	
Take advantage of human parallel processing capability	Integrated capabilities permit more work with less cognitive effort
Filter information judiciously	

5.1 User Interface for First Prototype Trials

While Figure 1 presents the User Interface derived from our analysis of operator activity and information requirements, the first prototype for the SPEEDD demonstration focuses on a specific subset of this use case. In the demonstration, the operator needs to monitor ramp metering and to accept (or challenge) the

automated systems control of ramps around the city. The User Interface for this task is presented in Figure 7. In addition to the User Interface supporting the demonstrator task, it also provides an opportunity for controlled experiments which will allow testing of the decision models and the eye-tracking metrics. For these experiments, participants will be presented with a series of ramp metering scenarios and will need to respond as quickly as possible to the automated system's recommendations. Using reaction time, it is possible to distinguish between different levels of performance, e.g., when all windows in the display contain corresponding information versus situations when information in one window conflicts with the others. In addition to reaction time, the experiments will also employ eye-tracking to ascertain which information sources participants tend to focus on under the different conditions.

Figure 7: User Interface for first SPEEDD demonstration

6. DISCUSSION

Key to the development of Visual Analytics is an appreciation of how Visual Analytics operates in a working environment in which other actors will share information with each other, or will interact with systems outside the core Visual Analytics system. This means that it important to appreciate the Socio-Technical Infrastructure in which the technology will be used (Figure 8). Consequently, the challenges this paper aims at addressing are the relating to information need, rather the information visualization. The latter is concerned by *how* the available information is presented, whereas the former shows *what* information shall be presented.

In this paper we demonstrate the application of Cognitive Work Analysis to the derivation of an Ecological Interface Design of the User Interface for the SPEEDD project's Road Traffic Management Use Case. Understanding operator tasks and information requirements (in terms of a Socio-Technical Systems) allows us to develop concepts for User Interface designs which reflect the job of the operator. This helps define the 'task ecology' in which operators perform their work, and helps define one aspect of the 'ecological' interface. The User Interface also reflects a desire to present information in formats which operators can spot patterns, trends and combinations of data using a form of 'direct perception'. The intention is to develop such designs so that operators can monitor system status by glancing at the displays during normal operations, rather than needing to engage in lengthy search and retrieval processes to discover information. The benefit of providing intuitive system overview is that it support operator Situation Awareness of steady-state, normal operations.

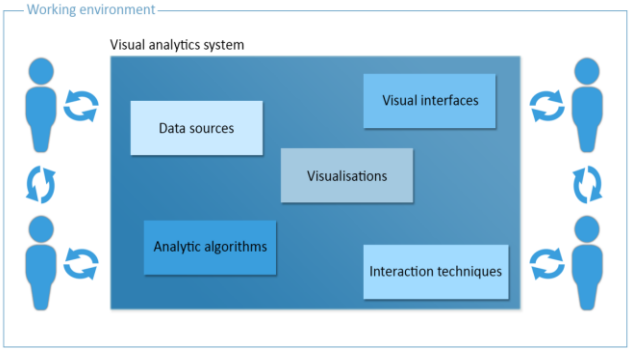


Figure 8: Visual Analytics in a Socio-Technical System

When operations deviate from normal, e.g., due to an actual or predicted incident, then the role of the operator changes from system monitor seeking to maintain Situation Awareness, to active responder seeking to ensure that system status returns to normal as quickly and efficiently as possible. In the SPEEDD project, this role is also performed by automated systems which detect system activity and perform responses to the activity. This means 'control' is now allocated between operator and automated system. We are developing the User Interface to not only inform the operator of system status, and automated system behavior but also to cue operators as to when (and how) they might need to intervene.

The User Interface shown in Figure 7, for instance, allows the operator to request that the system <explain> current settings and

decisions, at any time during the operation. However, if the operator feels that a setting or decision is not appropriate or correct, then the ramp being controlled can be selected and the decision can be queried, using the <challenge> button. This then allows the operator to either reset parameters or engage in some other form of intervention. While this is a simple example, it highlights how User Interface can be used to indicate the constraints under which the operator can act (where 'constraint' is seen as a positive means of shaping operator activity and indicating which function the operator is expected to perform).

7. REFERENCES

[1.] Keim, D., Kohlhammer, J., Ellis, G. and Mansmann, F. eds, (2010). Mastering the Information Age: Solving Problems with Visual Analytics, Goslar, Germany: Eurographics Association.

[2.] Bainbridge, L., (1983) Ironies of automation, *Automatica*, *19*, 775-779.

[3.] Sacha, D., Stoffel A., Stoffel, F., Kwon, B.C., Ellis, G. and Keim, D.A. (in press) Knowledge generation model for visual analytics, *IEEE Transactions on Visualisation and Computer Graphics*.

[4.] Byrne, E. A., & Parasuraman, R. (1996) Psychophysiology and adaptive automation, *Biological psychology*, *42*, 249-268.

[5.] Parasuraman, R., Cosenzo, K.A. and De Visser, E. (2009) Adaptive Automation for Human Supervision of Multiple Uninhabited Vehicles: Effects on Change Detection, Situation Awareness, and Mental Workload, *Military Psychology, 21*, 270–297.

[6.] Parasuraman, R., Mouloua, M. and Molloy, R. (1996) Effects of Adaptive Task Allocation on Monitoring of Automated Systems, *Human Factors, 38,* 665–679.

[7.] Johnson, A.W., Oman, C.M., Sheridan, T.B., Duda, K.R., 2014. Dynamic task allocation in operational systems: Issues, gaps, and recommendations, in: 2014 IEEE Aerospace Conference. Presented at the 2014 IEEE Aerospace Conference, pp. 1–15

[8.] Rasmussen, J. and Vicente, K. (1989) Coping with human errors through system design: implications for ecological interface design, International Journal of Human Computer Studies, 31, 517-534.

[9.] Gibson's (1969) Gibson, J. (1979) The Ecological Approach to Visual Perception, Boston, MA: Houghton Mifflin.

[10.] Jenkins, D.P., Stanton, N.A., Salmon, P.M. and Walker, G.H. (2009) Cognitive Work Analysis: coping with complexity, Avebury: Ashgate.

[11.] Vicente, K.J. (1999) Cognitive Work Analysis: towards safe, productive and healthy computer-based work, Mahwah, NJ: LEA.

[12.] Rasmussen, J., Pejtersen, A. And Goodstein, L.P. (1994) Cognitive Systems Engineering, New York: Wiley.

[13.] Folds, D., Brooks, J., Stocks, D., Fain, W., Courtney, T. and Blankenship, S. (1993) *Functional Definition of an Ideal Traffic Management System*, Atlanta, GA: Georgia Tech Research Institute.

Querying Graph Structured Data (GraphQ)

Federica Mandreoli (University of Modena and Reggio Emilia),
Riccardo Martoglia (University of Modena and Reggio Emilia),
Wilma Penzo (University of Bologna)

Using Graph Traversal in Scientific Data Interpolation

Alireza Rezaei Mahdiraji
Jacobs-University
Bremen, Germany
a.rezaeim@jacobs-university.de

Peter Baumann
Jacobs-University
Bremen, Germany
pbaumann@jacobs-university.de

ABSTRACT

In this paper, we present a topological neighborhood expression which allows us to express arbitrary neighborhood around cells in unstructured meshes. We show that the expression can be evaluated by traversing the connectivity information of the meshes. We implemented two algebraic operators which use the expression to compute neighbors of cells and approximate data fields of cells by aggregating their neighbors' information. We evaluate one of the operators on a real dataset using four queries and report the results.

Keywords

Graph Data Model, Halo, Hull, Regrid, Topological Neighborhood, Unstructured Meshes

1. INTRODUCTION

Many scientific domains such as oceanography and climatology have data stored on unstructured meshes. Weighted contribution from nearest-neighbor cells is known to improve accuracy of interpolation operations on unstructured meshes. Examples of such operations are smoothing a skewed data field, and computing partial derivative in a point of interest.

The common method to specify a neighborhood for a cell of interest is *stencil string* which is originally defined only for structured meshes. Stencil allows us to define the value of a cell as a function of its topological nearest-neighbor cells. In [3], the concept of stencil is generalized for unstructured meshes. A stencil string w.r.t. an unstructured mesh consists of a sequence of digits representing the dimensions of cells in the neighborhood of a cell of interest which needs to be accessed by an algorithm. The stencil string uses hard coded dimensions and thus contains no topological abstraction. Furthermore, it is not obvious from the string what is the result, i.e., union of elements visited in each intermediate layer (*hull*) or the elements only in the last layer (*halo*). Finally, it is not possible to filter intermediate cells using predicates.

In this paper, we present a neighborhood expression which uses topological functions instead of dimensions and allows filtering of intermediate results. We design two algebraic operators which use the expression to extract neighborhood and approximate information of a cell by interpolating information of its neighbors.

The paper is organized as follows: Section 2 introduces some mathematical concepts, Section 3 discusses the related work, Section 4 presents the new neighborhood expression, Section 5 shows the algebraic operators which use the expression to explore the neighborhood and interpolate data, Section 6 reports the experiments, and Section 7 concludes the paper.

2. MATHEMATICAL CONCEPTS

We define a mesh as a 4-tuple $M = (\mathcal{C}, \prec_{\mathcal{C}}, \Gamma, \mathcal{F})$ where $\mathcal{C}$, $\prec_{\mathcal{C}}$, Γ, and $\mathcal{F}$ are the set of cells, incidences, geometric embedding, and data fields of the mesh M.

$\mathcal{C}$ contains a set of k-cells ($0 \leq k \leq d$) which is closed under intersection, i.e., the intersection of two cells is either empty or another lower dimension cell of the mesh. Each cell in turn is formed by the union of lower dimensional cells (aka the cell boundary). Cells of dimension i are called i-cells. Cells of dimensions zero, one, two, and three are also called vertices, edges, faces, and bodies. Dimension of a mesh is defined as the maximal dimension of its cells (d).

The incidence set $\prec_{\mathcal{C}}$ specifies the topological structure of a mesh using the binary *incidence* relationship. The incidence relationship is a partial order between cells in $\mathcal{C}$, i.e., cells c and e are incident (i.e., $c \prec_{\mathcal{C}} e$) if one is located on the boundary of the other, i.e., $c \in \partial(e)$ ($dim(c) < dim(e)$) or $e \in \partial(c)$ ($dim(e) < dim(c)$) where $\partial()$ represents the boundary of the cell c. When k-cell c is on the boundary of m-cell e, then c is called a *k-face* of e and e is a *m-coface* of c or just *co-boundary* of c. When $k = m - 1$ then c is an *immediate boundary* face of e and e is an *immediate co-boundary* (coface) of c. Cells c and e ($k = m \neq 0$) are *p-adjacent* if they share a p-face. Two vertices v_1 and v_2 ($k = m = 0$) are adjacent if they share an edge. The adjacent relation can be expressed as nested application of the co-boundary and boundary relations, i.e., to find k-adjacent cells to the m-cell c, first we need to find all its boundary k-faces and then for each k-face find its m-cofaces.

The geometric embedding Γ maps each cell in $\mathcal{C}$ to its corresponding geometric realization such that $(c_1 \prec_{\mathcal{C}} c_2) \Leftrightarrow (\Gamma(c_1) \subset \Gamma(c_2) = \bigcup_{c \in \partial(c_2)} \Gamma(c))$. For instance, the geometric realization of vertices are their coordinates.

$\mathcal{F}$ contains a set of (partial) functions which assign data

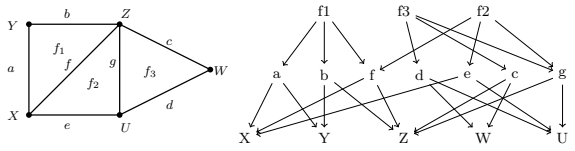


Figure 1: A 2D triangular unstructured mesh (left) and its incidence graph (right).

values to each cell. For instance, the function f_i assigns a value of type τ to each i-cells: $f_i : C^i \rightarrow \tau$ where C^i refers to i-cells.

We can represent mesh topology using *Incidence Graph (IG)* such that the nodes of the graph represents the cells of the mesh and the links between two nodes encode the incidence relationship. Figure 1 shows a simple 2D simplicial mesh (left) and it topological structure (right) as a incidence graph [13]. The incidence graph in Figure 1 shows that the boundary and co-boundary a cell can be extracted by following the links in the graph starting from the cell downward and upward, respectively. The immediate boundary and co-boundary cells of a cell are its children and parents. For instance, the immediate boundary and co-boundary of the edge d are $\{U, W\}$ and $\{f_3\}$, respectively and the boundary and immediate co-boundary of face f_3 are $\{d, c, g, W, U, Z\}$ and $\{\}$, respectively.

For detailed definitions, we refer the interested reader to [13], Chapter 3 of [3], and the references therein.

3. RELATED WORK

Stencil string specifies a subset of neighboring vertices of a given vertex on a structured mesh. The information of the neighbors are later used by numerical approximation algorithms to compute a weighted solution for the vertex. In other words, the stencils determine the region of influence of numerical algorithms. For example, commonly used neighborhood sizes by algorithms are three, five, and twenty five neighbors for 1D, 2D, and 3D structured meshes, respectively. The neighbors in a structured mesh can be easily specified by index arithmetic, e.g., the four neighbors of the point (i, j) in a 2D structured grid. In systems such as *Pochoir*, the neighboring vertices are explicitly listed [15]. Figure 2 (left) shows four neighboring vertices of the red vertex on a 3D structured mesh (5-point stencil).

In his thesis, Berti generalized the idea of stencils for unstructured meshes by abstracting topological structures of mesh algorithms as an *incidence sequence*. For instance, if the incidence sequence of a mesh algorithm is 010 (or alternatively shown as *vertex-edge-vertex*) w.r.t. an unstructured mesh, then it means any calculation for a vertex v needs to have access to all adjacent vertices of v (i.e., vertices sharing an edge with v) [3]. The snippet below shows how the stencil string 010 can be used to computes the data for each vertex v_1 as the sum of the data of its adjacent vertices (v_2).

```
forall vertices v1
  forall edges e incident to v1
      forall vertices v2 incident to e
         result[v1]+=data[v2]
```

Figure 2 (right) depicts 010 stencil for the red vertex in a 2D unstructured mesh. As it can be seen, the stencil string is implemented as nested for loops. Furthermore, Berti formally defined the concept of *incidence hull* as the union of all cells expressed by a stencil string. Gridfields uses stencil string in the same manner [7].

The proposed neighborhood expression in this paper extends the stencil string in [3]. The new neighborhood expression offers several advantages to the stencil string notation: it uses topological abstraction rather than dimensions, it can return two types of neighborhoods (i.e., hull or halo), and it is able to filter the intermediate results.

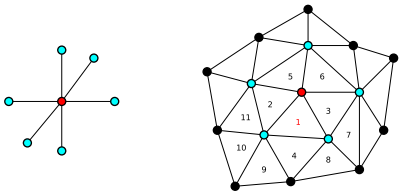


Figure 2: 5-point stencil for a vertex in 3D structured mesh (left) and adjacent vertices of the red vertex represented by 010 stencil in a 2D unstructured mesh (right).

The other related area to this research is *graph databases*. We use graph database *Neo4j* to implement the proposed neighborhood expression. Neo4j is a popular open source graph database implemented in Java which is operational since 2003. Neo4j is schema-free, supports full ACID transactions, and provides implementations for several graph algorithms out of the box. Neo4j uses *property graph* data model, i.e., data is stored in nodes and relationships of a multi-graph with pairs of key-value properties. Neo4j graphs can be queried using either its declarative query language (known as *Cypher*) or its Core Java API. We refer the reader to Neo4j documentation for further details on Neo4j and Cypher [1, 14].

The reason to choose graph databases is that they allow implementing the general purpose mesh data model (see Section 5.1). The reasons for choosing Neo4j rather than other graph databases such as *DEX* (implemented in C++) are two folds, namely, high performance on graph traversal queries and supporting spatial data. Recent research show that the performance of Neo4j drastically improved since its early version. The argument is supported by performance evaluation of several graph databases (e.g., such as Neo4j, DEX, orientDB, etc.) using data ingestion, traversal, non-traversal, and manipulation queries. Neo4j outperforms the other systems in traversal queries (the main focus of this paper) [8, 9, 11, 10]. Moreover, Neo4j is shown to have close performance to graph-processing frameworks such as *GraphLab* and *Giraph* [5]. The spatial layer is crucial for several mesh operations. This is beyond the scope of this paper and will not be covered here.

4. TOPOLOGICAL NEIGHBORHOOD EXPRESSION

The basic idea of the neighborhood expression is to use topological abstraction for expressing neighborhood traversal, i.e., to express a neighborhood as nested application of boundary, co-boundary, and adjacency functions. We define

these functions as follows:

Boundary Function. it is represented as $b(c,k)$ for a given cell c and it returns the boundary elements of dimension k ($k < dim(c)$) of the cell c:

$$b(c,k) = \{e|\ (e \in \partial(c))\ \wedge\ (dim(e) = k)\}$$

For a cell set C, it is defined as the union of k-cells on the boundary of each member of the set.

Immediate Boundary Function. it is represented as $ib(c)$ for a cell c and it returns the immediate boundary of c:

$$ib(c) = \{e|(e \in \partial(c)) \wedge (dim(e) = dim(c) - 1)\}$$

Co-Boundary Function. it is represented as $cob(c,k)$ for a given cell c and it returns the co-boundary of dimension k ($k > dim(c)$) of cell c:

$$cob(c,k) = \{e|\ c \in \partial(e)\ \wedge (dim(e) = k)\}$$

For a cell set C, it is defined as the union of k-cells on the co-boundary of each member of the set.

Immediate Co-Boundary Function. it is represented as $icob(c)$ for a given cell c and it returns the immediate co-boundary of c:

$$icob(c) = \{e|\ c \in \partial(e) \wedge (dim(e) = dim(c) + 1)\}$$

Adjacency Function. it is represented as $adj(c,k)$ for a given cell c and it returns the k-adjacent cells to the cell c, i.e., cells which share a k-face with c. Formally,

$$adj(c,k) = \{e|\ (\exists s : s \in (\partial(c) \cap \partial(e))) \wedge dim(s) = k\}$$

Note that $dim(c) = dim(e)$.
The definition of the adjacency relation for vertices is defined as follows:

$$adj(v) = \{c|\ \exists e : (dim(e) = 1) \wedge (v \prec e) \wedge (c \prec e)\}$$

where v and c are vertices. Note that in this case the function has only one argument.

4.1 Basic Neighborhood Expression

The basic neighborhood expression is semantically equivalent to the stencil string. The difference is that instead of using dimensions it uses the topological functions defined in the previous section. This means that arbitrary cell neighborhoods can be expressed as nested applications of the functions. Assuming the set C containing all initial elements which we would like to traverse their neighborhood (a.k.a. *seed*), the basic expression is defined as:

$$nex = e_n(e_{n-1}(\cdots e_1(\ e_0(C,k_0),k_1)\cdots,k_{n-1}),k_n)$$

where *nex* is the *neighborhood expression*, e_i is a (immediate) boundary, (immediate) co-boundary, or adjacency relationships, $0 \leq k \leq d$ (d is the mesh dimension), k_i shows an optional dimension argument for the functions, and $0 \leq i \leq n$. The expression repeatedly applies the functions to explore the neighborhood of the cells in the set C. Functions such as b needs a dimension parameter which specifies the dimension of the target boundary element. For the adjacency function, *adj* means p-cells which share a $(p-1)$-face with a given p-cell in the seed set.

The *nex* is a functional expression, i.e., its functions are applied from innermost to outermost, i.e., e_0 is applied to the seed set C and produces *layer* zero result L_0, e_1 is applied on L_1 and stored as *layer* one result L_1 and so on.

For instance, the stencil 010 (adjacent vertices of the seed vertices) can be expressed as $icob(ib(V))$ or simply $adj(V)$, where V is the seed set. As another example, a common stencil on 2D triangular mesh is 202, i.e., neighboring 2-cells which share at least one vertex with the seed 2-cell. In Figure 2 (right), the stencil for the triangle number 1 as seed contains triangles number 2 to 11. The stencil can be expressed as $cob(b(F,0),2)$ (or $icob(icob(ib(ib(F)))))$ where F is the seed set. The stencil 0102 can be succinctly expressed as $cob(adj(V),2)$ with V as the seed set.

The final result of the expression can be either the elements in the last layer or union of elements in all intermediate layers. In the latter case, the result forms a *stencil mesh* [7]. We define these two types of outputs as follows:

Halo Neighborhood. A halo neighborhood w.r.t. a given neighborhood expression *nex* contains the cells of the last layer only, i.e., before computing elements of the current layer it removes cells from the previous layer:

$$halo(C,nex) = L^n$$

Hull Neighborhood. A hull neighborhood w.r.t. a given neighborhood expression *nex* contains the union of elements in all intermediate layers of the *nex* evaluation:

$$hull(C,nex) = \bigcup_{i=0}^{i=n} L^i$$

4.2 Advanced Neighborhood Expression

The neighborhood expression from Section 4.1 has enough abstraction w.r.t. mesh topology. However, in comparison to stencil string, it does not offer anything new. Often applications need to filter the intermediate results of the traversal while searching the neighborhood. This is not possible with stencil string. Thus, we further extend the basic neighborhood expression to allow filtering of cells in intermediate layers using predicates on mesh components (e.g., field values, geometry, etc.). To this end, we extend the definition of the functions as follows:

Extended Boundary Function. It is represented as $b(C,k,p)$. It computes the k-cells in the boundary of each element in C and then it checks if the boundary elements satisfies the predicate p. It returns elements in the boundary which satisfy p. It is formally defined as follows:

$$b(C,k,p) = \bigcup_{c \in C} \{e|(e \in \partial(c)) \wedge (dim(e) = k) \wedge p(e)\}$$

where $p(e)$ means that the cell e satisfies p.

Extended Immediate Boundary Function. It is represented as $ib(C,p)$. For each cell c in C, it computes its immediate boundary and applies the predicate p on each cell in the immediate boundary of c. It returns cells in the immediate boundary which satisfy the predicate. It is formally defined as follows:

$$ib(C,p) = \bigcup_{c \in C} \{e|(e \in \partial(c)) \wedge (dim(e) = dim(c) - 1) \wedge p(e)\}$$

Extended Co-Boundary Function. It is shown as $cob(C,k,p)$. For each cell c in C, it computes its co-boundary elements of dimension k and checks if they satisfies the predicate p. It returns co-boundary cells which satisfy the predicate. It is formally defined as follows:

$$cob(C,k,p) = \bigcup_{c \in C} \{e|(c \in \partial(e)) \wedge (dim(e) = k) \wedge p(e)\}$$

Extended Immediate Co-Boundary Function. It is represented as $icob(C, p)$. For each cell c in C, it computes its immediate co-boundary and applies the predicate p on each cell in the immediate co-boundary of c. It returns cells in the immediate co-boundary which satisfy the predicate. $icob(C, p)$ is formally defined as follows:

$$\bigcup_{c \in C} \{e | (c \in \partial(e)) \wedge (dim(e) = dim(c) + 1) \wedge p(e)\}$$

Extended Adjacency Function. It is represented as $adj(C, p)$. For each p-cell c in C, it computes its $(p-1)$-adjacent cells and checks the predicate p. It returns $(p-1)$-adjacent cells which satisfy the predicate. $adj(C, p)$ is formally defined as follows:

$$\bigcup_{c \in C} \{e | (\exists s : s \in (\partial(c) \cap \partial(e))) \wedge (dim(s) = k) \wedge p(e)\}$$

With the definitions above, the new functional neighborhood expression is as follows:

$$nex = e_n(e_{n-1}(\cdots e_1(\ e_0(C\ , k_0, p_0), k_1, p_1) \\ \cdots, k_{n-1}, p_{n-1}), k_n, p_n)$$

where k_i and p_i are the (optional) dimension and the predicate arguments, respectively. The predicate p_i is defined on properties of cells returned by e_i, e.g., predicates on data fields or geometric features such as length, area, volume, etc.

For instance, the expression $icob(ib(T, f < 24.0))$ (or equivalently $adj(T, 2, f < 24.0)$) finds 2-adjacent 3-cells using 3-cells in T as seed set. The predicate selects only 2-cells where the value of the field f is less than 24.0.

Such a functional notation with many nested parentheses can become very tedious to read/write. Thus, we propose a notation inspired by *XPath* [2]. We use slash to separate the topological functions and brackets to express predicates. Assuming C as the seed set, the previous nex can be written as follows:

$$nex = e_0(k_0)[p_0]/e_1(k_1)[p_1]/\cdots/e_{n-2}(k_{n-2})[p_{n-2}]/ \\ e_{n-1}(k_{n-1})[p_{n-1}]/e_n(k_n)[p_n]$$

Note that k_i and p_i are optional and the seed set is not present in the expression. In the next Section, we show how to specify the seed.
For instance, the expression from the previous example can be written as $ib[f < 24.0]/icob$ using T as seed.

In comparison to the functional form, the evaluation of XPath-like neighborhood expression is done left-to-right, i.e., first applying e_0 to the seed set C and filtering the result using p_0, then applying e_1 to the result from layer zero and so on. More formally, the elements in each intermediate layer is computed as follows:

$$nex = \begin{cases} L^0 = \bigcup_{c \in C} \{e | e \in e_0(c) \wedge p_0(e)\} & i = 0 \\ L^1 = \bigcup_{c \in L^0} \{e | e \in e_1(c) \wedge p_1(e)\} & i = 1 \\ \cdots & \cdots \\ L^n = \bigcup_{c \in L^{n-1}} \{e | e \in e_n(c) \wedge p_n(e)\} & i = n \end{cases}$$

where L^i represents elements in the layer i and $p_i(e)$ means that the predicate p_i holds for cell e.

5. IMPLEMENTATION

We implemented a set of algebraic mesh operators which we call *AMQL* (Algebraic Mesh Query Language). The operators are declarative meaning that we only need to describe the information need and the AMQL engine will figure out how to find it. Two of AMQL's operators use the neighborhood expression, namely, the *neighbors* and *self-regrid* operators. Other operators are described in [13]. In the rest of this section, first we discuss the data model which we use to store unstructured meshes as graphs and then we explain the neighbors and self-regrid operators.

5.1 Graph Data Model for Meshes

In Section 2, we discussed that incidence graph can store connectivity information of unstructured meshes. The graph model allows us to find boundary, co-boundary, and adjacency relationships of cell using graph traversal.

The IG does not encode information about the data fields and geometric embedding of cells. Many mesh application domains have operations which needs to manipulate fields and geometry data. Furthermore, the IG stores only incidence relationship and the adjacency relationship needs to be computed on demand. Neighborhood queries use adjacency information extensively. Adjacency based data structure are proved to be more efficient [4].

Based on the above observations, we extend the IG model to a general purpose mesh model such that nodes in the graph contains data fields and geometries and each node stores information about its adjacent nodes. Figure 3 shows the graph data model for a linear 3D unstructured mesh. The nodes in the graphs represent the cells, e.g, vertex, edge, face, and body. $VField_i$, $EField_i$, $FField_i$, and $BField_i$ represent properties of vertices, edges, faces, and bodies, respectively. *Geom* is the geometric object of each node, i.e., *point* for vertex, *line-segment*, *polygon* for face, and *volume* for body. This allows us to compute geometric predicates such as length, area, volume, centroid, distance, etc. Furthermore, there are two types of topological relationships between nodes, namely, boundary (between cells of different dimensions) and adjacency (represented as self-loop in the Figure 3). These relationships allow us to navigate the topology using graph traversal algorithms.

5.2 Self-Regrid Operator

In domains such as climatology and oceanography, the regrid operator is used to transform data from source meshes to a target mesh [6]. The operator works in two steps: first, it assigns a set of target cells to each source cell (mapping step), then, it combines the data of the mapped cells to estimate data of the target cell (interpolation step) [7].

The *self-regrid* operator is a special case of the regrid operator (i.e., the source and the target meshes are the same) where the goal is to estimate the data of a cell using its neighbors information. The mapping step uses topological or geometric neighborhood functions. The geometric mapping function, which assigns neighbors to a cell based on their geometric distance to the cell (e.g., k-nearest neighbors), is beyond the scope of this paper. The neighborhood expression introduced in this paper can be used as a topological mapping function in the self-regrid operator to assign neighboring cells to each given cell, e.g., assigns all adjacent vertices to each vertex.

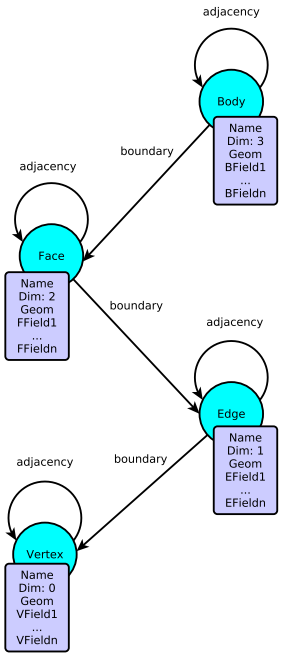


Figure 3: The graph data model for 3D linear meshes.

The notation for self-regrid operator is:

$$regrid(M,\ i,\ nex,\ agg\_f = aggFunc(f))$$

The operator assigns cells defined by nex to each i-cell of M using the i-cell as the seed. Then, it applies the aggregation function $aggFunc$ on the field f of the mapped cells and store the result as a new field agg_f for the i-cell. The algorithm 1 shows how the self-regrid algorithm works. The algorithm loops over i-cells of the mesh M (line 7) and by using each i-cell as seed evaluate the topological functions from left-to-right. Th topological functions is stored in L and $L[i]$ refers to the ith function in the expression nex. For instance, if the topological function is ib and has a corresponding predicate, it first computes the result of the topological function and then for each cell in the result check the predicate. If there is no corresponding predicate, it just returns the result of the function (line 12-17). The value of field f of the mapped cells is combined to estimate value for the target cell (line 23-26).

For instance, the following self-regrid operator can be used to smooth temperature field on each vertex of the mesh M using information from its adjacent vertices. To find adjacent vertices we need to compute the neighborhood expression adj (or $icob/ib$). This is equivalent to the stencil string 010.

```
regrid(M, 0, adj, agg_temp=avg(temperature))
```

In more structured and readable form, we can write it as follows:

```
FOR VERTEX v IN M
MAP v TO neighbors(M, v , halo, icob/ib ) AS m
RETURN v, avg(m.temperature)
```

The FOR loop iterates over all vertices of the mesh M. For each vertex, the operator maps the vertex to the output set

Algorithm 1: Evaluation of the Regrid Operator $regrid(M,\ i,\ nex,\ agg\_f = aggFunc(f))$

```
   input : Mesh M, dimension i, field f, aggregation function
           aggFunc, output field name agg_f, and neighborhood
           expression nex
   output: Mesh M with new field agg_f
 1 L ← List of (co-)boundary functions extracted from nex;
 2 P ← List of predicates extracted from nex;
 3 MappedCells ← ∅;
 4 Seed ← ∅;
 5 S ← ∅;
 6 fvals ← ∅;
 7 while (there are i-cells in C) do
 8 |   Seed ← next unused i-cell c;
 9 |   k ← i;
10 |   for (j from 0 to length(L))) do
11 |   |   for (e in Seed) do
12 |   |   |   if (P[i] ≠ null) then
13 |   |   |   |   I ← L[j](e);
14 |   |   |   |   add cell c from I to S where P[i](c) holds;
15 |   |   |   else
16 |   |   |   |   add ib(e) to S;
17 |   |   |   end
18 |   |   end
19 |   |   Seed ← S;
20 |   |   S ← ∅;
21 |   end
22 |   MappedCells ← Seed;
23 |   for (k-cell e in MappedCells) do
24 |   |   add e.f to fvals;
25 |   end
26 |   c."agg_f" = aggFunc(fvals) ;
27 |   MappedCells ← ∅;
28 |   fvals ← ∅;
29 end
```

m from the neighbors operator (using the MAP ... TO ... AS clause) and return the average of the temperature of the mapped cells. The dot notation is used to refer to the field temperature of a vertex, i.e., e.temperature.

5.3 Topological Neighbors Operator

The neighbors operator can create sub-meshes (a.k.a. stencil meshes). It is represented as $\mathcal{N}(M, E, ROI, nex)$ where its arguments are a mesh, seed set, the *Region Of Influence* (ROI), and a neighborhood expression, respectively. The *Region Of Influence (ROI)* is either *halo* or *hall*. The operator returns a set of cells by evaluating the expression nex on the seed set E w.r.t. to the ROI argument.
For instance, the hull mesh around a vertex v containing the vertex itself and all edges and faces can be expressed as $\mathcal{N}(M, \{v\}, hull, icob/icob/ib/ib)$.

Algorithm 2 shows how to construct result of the neighbors operator $\mathcal{N}(M, E, halo, nex)$. A similar algorithm can be written for the hull with small modification of the algorithm 2.

6. EXPERIMENTAL RESULTS

6.1 Experimental Setup

Experimental Design. We conducted experiments to evaluate the performance of the declarative self-regrid operator. The operator is implemented within AMQL which contains a collection of declarative operators for unstructured meshes implemented in Java. As explained in Section 3, the main reason to use Java is that Neo4j provides spatial data management (crucial for some of the operators) and either performs better or has close to existing graph

Algorithm 2: Algorithm Evaluation of Neighbors Operator $\mathcal{N}(M, E, halo, nex)$

input : Mesh M, seed cells E, $ROI = halo$, and neighborhood expression nex
output: Sequence N containing all cells which are in the defined neighbourhood by nex

```
1  N ← ∅;
2  L ← List of (co-)boundary functions extracted from nex;
3  P ← List of predicates extracted from nex;
4  Seed ← E;
5  S ← ∅;
6  I ← ∅;
7  k ← dimension of element in E;
8  for (i from 0 to length(L)) do
9      for (e in Seed) do
10         if (P[i] ≠ null) then
11             I ← L[i](e);
12             add cell c from I to S where P[i](c) holds;
13         else
14             add ib(e) to S;
15         end
16     end
17     Seed ← S;
18     S ← ∅;
19 end
20 N ← Seed;
```

databases and frameworks such as DEX or GraphLab. We compare the performance of our implementation with *GrAL* (a C++ mesh library) [3] by measuring the execution time (this does not include time of building internal data structures or indexes for each system). The reason to use GrAL is that it uses a generic approach to meshes which enables it to express virtually any combinatoric query using domain specific language of iterators [3].

We use four smoothing field queries in the experiments. The queries differ in the length of the neighborhood expressions and use of predicates. This allows us to observe the effect of expression length and predicates on the performance.

Implementation Details. We run the experiments on a system with four cores (2.4 GHz processor) with 8GB of RAM and XUbuntu 12.10 operating system.

We use *ANTLR* to parse AMQL operators including the self-regrid [12]. The operators are implemented in Java and use Neo4j database facilities, e.g., storage, indexes, traversal framework, etc. However, the AMQL implementation is storage neutral, i.e., the implementation is abstracted and can be used with any other system which provides implementations for the abstract methods.

We implemented the self-regrid operator using both Cypher and Neo4j core Java API. We refer to these two implementations as AMQL_Cypher and AMQL_Java, respectively. In particular, the implementation of AMQL_Cypher translates each regrid query to a Cypher query. We report performance of each implementation .

We use GrAL as of 1.11.2014 and Neo4j 2.1.6. GrAL is compiled using gcc 4.6.3 with setting -03 which controls depth of template instantiation. GrAL implements each query separately in C++. We run each query ten times and average the response times over ten trials for each pair of (query,dataset).

Dataset. We use a real dataset from oceanographic domain [7]. The dataset contains a 2D triangular mesh where each vertex has two data fields, namely, temperature and bathymetry. The number of vertices, edges, and faces in the dataset are 20736, 39133, and 59884, respectively. To see how the systems perform with the data set size, we applied the subsetting operator from AMQL to divide the dataset to three smaller datasets with different size of vertices, i.e., 4862 (D1), 10270 (D2), and 20736 (D3). As it can be seen, the second dataset has (almost) twice the number of vertices as in the dataset two and the dataset three has twice the number of vertices as in the dataset two. The reason behind the subsetting is that all the queries needs to iterate over all the vertices in the datasets. This means the workload for each dataset is twice the previous dataset.

Queries. We use four field smoothing queries for the experiment. The smoothing operation is commonly used to smooth a noisy data field or a data field with missing values. The queries are as follows:

Q1. *Compute temperature of each vertex as average of the temperature of its adjacent vertices.*

In the Section 5.2, we showed the regrid operator for this query and its Cypher translation is as follows:

```
MATCH (p0:M)
WHERE p0.dim=0
WITH p0
MATCH (p0:M)<-[:ADJACENCY]-(p1:M)
RETURN p0.cid, avg(p1.temperature)
```

The *MATCH* clause is used for graph pattern matching. The first MATCH clause defines an iterator variable $p0$ on all nodes of mesh M. The WHERE clause filters $p0$ to vertices where *dim* property is zero. The *WITH* clause chains several smaller queries. In the query, the WITH only passes the vertices to the next part of the query. The second MATCH clause does a path matching in the graph by selecting all $p1$ nodes which has ADJACENCY relationship with $p0$. Finally, the RETURN clause returns the identifier of each vertex in $p0$ and average value of temperature over all correspondent $p1$. We refer the interested reader to Neo4j documentation for elaborate details on Cypher [1].

The GrAL C++ code implementing the same query consists of 10 lines of codes which uses underlying GrAL abstraction such as Cell-On-Cell iterators and mesh functions.

Q2. *Compute temperature of each vertex as average of the temperature of its adjacent vertices with the bathymetry field greater than 5.0.*

The query can not be expressed by a stencil. The query is equivalent to the neighborhood expression *adj[bathymetry > 5.0]* or *icob/ib[bathymetry > 5.0]*. The implementation of the query in the AMQL is as follows:

```
regrid(M, 0, adj[bathymetry > 5.0],
       agg_temp=avg(temperature))
```

The regrid operator above is translated to the the following Cypher query:

```
MATCH (p0:M)
WHERE p0.dim=0
WITH p0
MATCH (p0:M)<-[:ADJACENCY]-(p1:M)
WHERE p1.bathymetry>5.0
RETURN p0.cid, avg(p1.temperature)
```

The GrAL C++ code implementing the same query consists of 10 lines of codes.

Q3. *Compute temperature of each vertex as average of the temperature of vertices which are exactly two edges (2-hops) away from the vertex.*

This is equivalent to the stencil string 01010 and the neighborhood expression *adj/adj* (or *icob/ib/icob/ib*). The self-regrid operator pertaining to the query is:

```
regrid(M, 0, adj/adj,
        agg_temp=avg(temperature))
```

The regrid operator is translated to the following Cypher query.

```
MATCH (p0:M)
WHERE p0.dim=0
WITH p0 MATCH (p0:M)<-[:ADJACENCY]-(p1:M)
        <-[:ADJACENCY]-(p2:M)
RETURN p0.cid, avg(p2.temperature)
```

The GrAL C++ code implementing the same query consists of 15 lines of codes. The code uses several abstraction concepts from GrAL [3] and contains three nested FOR loops.

Q4. *Compute temperature of each vertex as average of the temperature of vertices which are exactly two edges away from it. Consider immediate adjacent vertices only if their bathymetry field is greater than 5.0.*

There is no stencil equivalent to this query. The query is equivalent to the neighborhood expression *adj[bathymetry > 5.0]/adj* (or *icob/ib[bathymetry > 5.0]/icob/ib*). The implementation of the query in AMQL is as follows:

```
regrid(M, 0, adj[bathymetry > 5.0]/adj,
        agg_temp=avg(temperature))
```

The above regrid operator is translated to the following Cypher query.

```
MATCH (p0:M)
WHERE p0.dim=0 WITH p0
MATCH (p0:M)<-[:ADJACENCY]-(p1:M)
        <-[:ADJACENCY]-(p2)
WHERE p2.bathymetry>5.0
WITH p0, p2
MATCH (p2:M)
RETURN p0.cid, avg(p2.temperature)
```

Note that the complexity of the Cypher query grows with the length of the expression and the number of predicates. Moreover, some predicates such as geometric predicates can not be translated to Cypher.

The corresponding GrAL code for the query consists of 18 lines with three nested FOR loops.

6.2 Performance Evaluation

Figure 4, 5, 6, and 7 show the results of the **Q1**, **Q2**, **Q3**, and **Q4** queries. Clearly, the GrAL implementation outperforms AMQL in all the queries except **Q3**. A closer look on the performance data of the **Q1**, **Q2**, and **Q4** queries shows that GrAL on average is 150 (570), 140 (565), 500 (60) percent faster than AMQL_Java and AMQL_Cypher, respectively. However, in **Q3**, GrAL is on average 15 percent slower than AMQL_Java and 140 percent faster than AMQL_Cypher.

The performance of AMQL_Java increases by increasing the length of the neighborhood expression (see Figures 4 and 6). Its performance on Q3 even outperforms GrAL. This means that the traversal framework of Neo4j is very efficient on long expression. However, increasing the length and adding predicates cause a drastic increase in the performance of AMQL_Java (see Figures 6 and 7). This means that the traversal framework of Neo4j Java API does not perform well on a complex neighborhood expression with long length and predicates.

In comparison to AMQL_Java, the performance of AMQL_Cypher is better on longer expressions with predicates (see Figures 7). The reason is that we need to apply the traversal framework several times while evaluating an expression with predicates. Furthermore, the evaluation of predicates is done using Java code. However, AMQL_Cypher breaks down the query to shorter path and applies the predicates directly on Neo4j (which is faster than running on Java).

We conclude that both Cypher and Neo4j Java API should be used in implementing of the the expression depending on length of the expression and usage of predicates.

We observe that by increasing the length of the neighborhood expression and adding predicates the performances of AMQL and GrAL get closer (see Figure 6 and 7).

A common pattern in the performance of the both systems is that the response times increase linearly with the number of vertices. More precisely, for each pair of (system, query) the execution time on D2 is (almost) twice the execution on D1 and the execution time on D3 is (almost) twice the execution on D2.

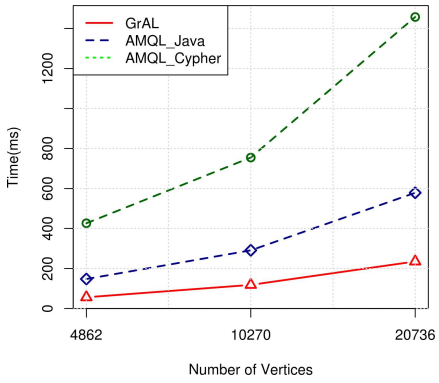


Figure 4: Performance of Q1 on AMQL and GrAL.

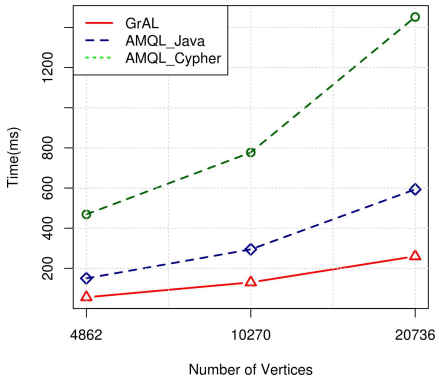


Figure 5: Performance of Q2 on AMQL and GrAL.

We believe that the poor performance of AMQL in comparison to GrAL has the following reasons. First, AMQL provides a generic solution which can accept any neighborhood expression as input while the GrAL implementations are query specific, i.e., any changes in the query requires changes in the implementation. The generic solution offers a declarative way of expressing the self-regrid but it has a cost which is the query parser overhead. Moreover, we use Neo4j's transactions in the implementation which introduce significant overhead. Also, in comparison to GrAL which uses a light and pure topological data structure, AMQL uses

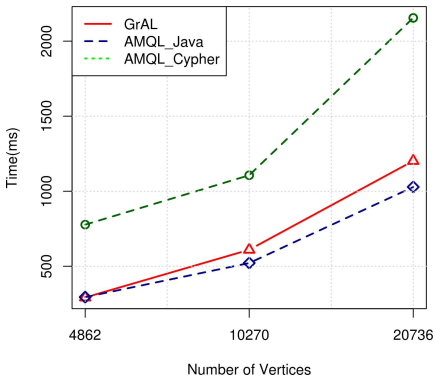


Figure 6: Performance of Q3 on AMQL and GrAL.

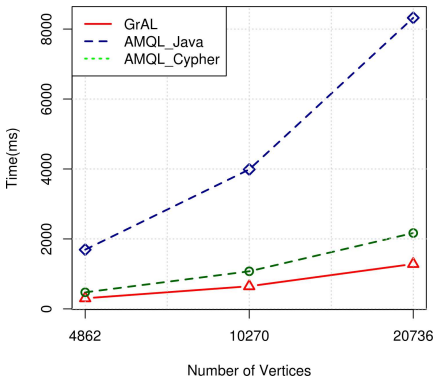


Figure 7: Performance of Q4 on AMQL and GrAL.

a general purpose mesh data model which contains the complete mesh information. This introduces further query processing overhead. Finally, it is known that Java language has inherent performance inefficiencies compare to C++.

To sum up, AMQL implementation works better on long neighborhood expression without predicates. In terms of expressiveness, the expression described in this paper is more expressive than the stencil string. Furthermore, it offers declarative querying (i.e., shorter and more readable than C++) and allows persisting of the computed data (i.e., the result of the regrid can be stored as a new field in the input mesh).

7. CONCLUSIONS AND FUTURE WORK

We presented a topological neighborhood expression which is more expressive than the stencil string. The implementation of the expression is declarative and generic. However, compare to a query-specific implementation in C++ it performs poorly (except on very long expressions).

In the future, we would like to measure the cost of the operator w.r.t. to the total cost of the queries and improve the operator implementation. Also, we would like to implement the operators on top of a graph database (framework) written in C++ such as DEX or GraphLab and repeat the experiments. We also want to use the expression in a structured query language for unstructured meshes similar to the example in the Section 5.2.

8. ACKNOWLEDGMENTS

The authors would like to thank Dr. Guntram Berti for his help in implementing the queries in GrAL library.

9. REFERENCES

[1] Neo4j - the world's leading graph database, Viewed December 2014.

[2] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Siméon. Xml path language (xpath). *World Wide Web Consortium (W3C)*, 2003.

[3] G. Berti. *Generic software components for Scientific Computing*. PhD thesis, BTU Cottbus, 2000.

[4] D. Canino, L. De Floriani, and K. Weiss. Ia*: An adjacency-based representation for non-manifold simplicial shapes in arbitrary dimensions. *Computers & Graphics*, 35(3):747–753, 2011.

[5] Y. Guo, M. Biczak, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke. Towards benchmarking graph-processing platforms. *Poster at Supercomputing*, 2013.

[6] H. Hinterberger, K. A. Meier, and H. Gilgen. Spatial data reallocation based on multidimensional range queries. a contribution to data management for the earth sciences. In *Scientific and Statistical Database Management*, pages 228–239. IEEE, 1994.

[7] B. Howe. *Gridfields: model-driven data transformation in the physical sciences*. PhD thesis, Portland, OR, USA, 2007. AAI3255425.

[8] S. Jouili and V. Vansteenberghe. An empirical comparison of graph databases. In *Social Computing (SocialCom), 2013 International Conference on*, pages 708–715. IEEE, 2013.

[9] V. Kolomičenko, M. Svoboda, and I. H. Mlỳnková. Experimental comparison of graph databases. In *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, page 115. ACM, 2013.

[10] P. Macko, D. Margo, and M. Seltzer. Performance introspection of graph databases. In *Proceedings of the 6th International Systems and Storage Conference*, page 18. ACM, 2013.

[11] R. C. McColl, D. Ediger, J. Poovey, D. Campbell, and D. A. Bader. A performance evaluation of open source graph databases. In *Proceedings of the first workshop on Parallel programming for analytics applications*, pages 11–18. ACM, 2014.

[12] T. Parr. *The definitive ANTLR reference: building domain-specific languages*. Pragmatic Bookshelf, 2007.

[13] A. Rezaei Mahdiraji. Toward unstructured mesh algebra and query language. In *Proceedings of the 2014 SIGMOD PhD Symposium*, pages 16–20. ACM, 2014.

[14] I. Robinson, J. Webber, and E. Eifrem. *Graph databases*. " O'Reilly Media, Inc.", 2013.

[15] Y. Tang, R. Chowdhury, C.-K. Luk, and C. E. Leiserson. Coding stencil computations using the pochoir stencil-specification language. In *3rd USENIX Workshop on Hot Topics in Parallelism (HotPar'11)*, 2011.

A Parallel Tree Pattern Query Processing Algorithm for Graph Databases using a GPGPU

Lila Shnaiderman
Computer Science Department, Technion
lilas@cs.technion.ac.il

Oded Shmueli
Computer Science Department, Technion
oshmu@cs.technion.ac.il

ABSTRACT

Large amounts of data are modeled and stored as graphs in order to express complex data relationships. Consequently, query processing on graph structures is becoming an important component in real-world applications. The most commonly used query format is that of tree pattern queries. We present a new parallel SIMD algorithm, GGQ (GPU Graph data base Query), for answering tree pattern queries on graph databases, using a GPU. We present the results of extensive experimentation of GGQ on large graph databases using known benchmarks that show that GGQ is an effective and competitive algorithm.

1. INTRODUCTION

Graph databases are widespread in many areas, including the semantic web and social/biological networks, as a graph is a more flexible and expressive structure than a tree. One of the most important and practically most interesting query formats for graph databases is a tree pattern query (TPQs - Tree Pattern Queries). In most known query languages for XML and RDF (such as XQUERY and SPARQL [19]), many queries can be regarded as TPQs on graphs. An example of a TPQ query is presented in Figure 2 (right end side). Finding all occurrences of matching a TPQ query to an isomorphic sub-graph of a given data graph is a fundamental operation in graph query processing. Related basic problems are (a) determining if a matching exists, and (b) providing part of the matched data nodes, corresponding to query target nodes, as the result.

Lately, there has been much research on using GPUs to speedup database operations. The standard use of GPUs is to render graphical information. GPUs are a cheap and ubiquitous source of processing power, as at least one GPU can be found in almost any computer. GPUs follow a SIMD (Single Instruction, Multiple Data) architecture, while multi-core systems follow a MIMD (Multiple Instructions, Multiple Data) architecture. In SIMD, multiple processing elements perform the **same** operation on multiple data elements, simultaneously.

We focus on processing TPQ queries and not on general graph structured queries as, in practice, TPQ queries seem to be the most frequently used type. Few research projects have addressed parallelizing query processing over graph databases. The main idea has been to use the data partitioning strategy, i.e., methods to partition the data between many computing elements, for example see [9]. To the best of our knowledge, there have been no attempts to parallelize the query processing of a *single* TPQ, or to design a parallel algorithm that exploits GPUs (or any other SIMD-based device) to accelerate the processing of a *single* TPQ.

In this paper we present the GGQ algorithm (GPU-Graph data base Query). The problem we address is how to use GPUs to accelerate the processing of a *single* TPQ query. The main idea underlying GGQ is to copy the relevant parts of the graph document, according to the input query, to the GPU global memory, to process the query using *all* the threads of the GPU in parallel, and to copy the query results back to the CPU memory. The key to parallelizing the query processing is in the ability to efficiently coordinate the query processing tasks between thousands of working units. In GGQ, each thread checks a different potential matching between the TPQ pattern and the data graph. In case that the checked potential matching actually exists, the thread reports this matching as one of the answers to the query.

GGQ is novel in that thread identifiers (IDs) are used to determine the choices made in attempting to match the tree pattern to actual database graph nodes and edges. As the space of possibilities that can be represented by an ID is limited, methods are presented to practically increase this space.

To minimize the amount of data that has to be copied to the GPU for a particular query execution, we designed a new graph lists storage scheme, GLS, that is based on a XML stream representation scheme [11]. A section describing GLS is not included in the paper due to lack of space.

For documents that can fully reside in the global memory, we gain speedup of up to 1000 times in comparison to Gremlin [17][1] (counting the time of copying the results from the GPU to the CPU but not counting copying from the CPU to the GPU). If a whole document is loaded to the GPU, many queries on this document can be processed one after the other, thus eliminating the need to copy the document, for each query, from the CPU to the GPU. For documents that can not fully reside in the GPU global memory, according to our experiments, we still gain a significant improvement of up to 100 times in comparison to Gremlin (counting the time of copying the data from the CPU to the GPU and the time of copying the results from the GPU to the CPU). In experiments with an extra-large query, we obtained speedup of up to 50 times in comparison to Gremlin (while counting just the copying time of the results from the GPU to CPU), and up to 35 times in comparison

[1]There may by now be tools for public use that are more efficient than Gremlin.

to Gremlin (while counting the time of copying the data from the CPU to the GPU and the time of copying the results from the GPU to the CPU).

2. BACKGROUND

In this section, we briefly introduce GPUs and CUDA (the underlying platform upon which the GGQ algorithm is implemented), and TPQ pattern matching.

2.1 Graphics Processors (GPUs)

GPUs were originally designed for dealing with graphics rendering. In recent years GPUs are also used as multi-threaded multi-core co-processors for CPUs. GPUs have a SIMD (Single Instruction, Multiple Data) architecture. In the SIMD architecture, there are multiple processing elements that perform the same operation on multiple data, simultaneously. Any algorithm for GPUs has to fit the SIMD scheme; hence an original CPU (or multi-core) algorithm **should not** be run as is on a GPU. If run as is, it will most probably be extremely inefficient. Programmers write their algorithms so that the part of the algorithm that does not have to be massively parallelized runs on the CPU and the other part, which can be massively parallelized, runs on the GPU.

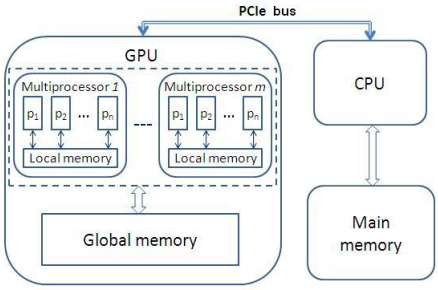


Figure 1: GPU architecture model

GPU Hardware Architecture. The GPU architecture is shown in Figure 1. This architecture is common to both NVIDIA [16] and AMD [13]. The NVIDIA GTX processor is a collection of multiprocessors (in GTX480 there are 15 multiprocessors), each with a group of processors (32 in GTX480). Each multiprocessor has its own shared memory which is common to all the processors within it. It also has a set of registers, texture, and constant memory caches. At any given cycle, each processor in the multiprocessor executes the same instruction on different data. A *warp* is a collection of threads that run simultaneously on a multiprocessor. The warp size is fixed for a specific GPU. Communication between multiprocessors (i.e., processors from different multiprocessors) is through the device memory (also called global memory), which is available to all the processors of the multiprocessors. The size of the global memory is limited. The GTX480, for example, has 1.5GB memory. The global memory has both a high bandwidth and high access latency. GPU threads have both low context-switch and low creation time as compared to CPU threads. The global memory is available to all the threads, so any thread can access any memory location.

CUDA Programming Model. Programmers use two types of code, the **kernel code** and the **host code**. The kernel code is executed on the GPU. The host code runs on the CPU. The host part is in charge of transferring data between the GPU and main memory, and starting kernel-code instances (kernels) on the GPU. A computation task on the GPU is divided into three separate steps. First, the host code allocates GPU memory for input and output data, and copies input data from the main memory to the GPU memory. Second, the host code starts threads each executing the kernel code, kernels, on the GPU. The kernels perform the required task on the GPU. Third, when the kernels finish their work, the host code copies results from the GPU memory to main memory. For the programmer the CUDA model is a collection of threads running in parallel. A collection of threads (called a *block*) runs on a multiprocessor at a given time. One can assign multiple blocks to a single multiprocessor and then the blocks execution on the multiprocessor is time-shared.

Execution. All threads of all blocks executing on a single multiprocessor share its resources. Each thread and block has a unique ID. In addition, each thread has a program counter, registers, per-thread private memory, and inputs that can be used by the thread during its execution. Each thread in a set of parallel threads executes an instance of the kernel code, in parallel. Blocks are further organized into grids of thread blocks by the programmer. Each grid is a 2 or 3-dimensional arrangement of blocks. When a block is executed, it is further divided into warps. Using the thread and block IDs each thread can perform the kernel code on different set of data. In some cases, during some operations, for example an *if else* statement, some of the threads in a multiprocessor are idle (during the *if* block or the *else* block), as according to the *if else* statement, they do not have to process the body of the *if* block or the *else* block of the statement.

2.2 TPQ (Tree Pattern Query) pattern matching

Tree Pattern Queries (TPQs) are represented as directed trees, where (1) the nodes and edges of a TPQ Q are labelled by labels from an alphabet $\sum$. The label of a node u is denoted by $\tau(u)$, and the root node of Q is denoted by $root(Q)$. The size of Q, denoted by $|Q|$, refers to the number of nodes in Q. (2) The nodes in Q are connected by parent-child edges (pc-edges) labeled by a label from $\sum$. Consider an edge $e = (u, v)$ with parent node u and child node v, we say that v is a child of u and u is the parent of v.

Given a TPQ Q with nodes (q_1,..., q_n) and a directed graph document D, a *match* of Q in D is a mapping from the nodes of Q to nodes (d_1,..., d_n) in D s.t.: (1) d_i is matched with q_i, $1 \leq i \leq n$, (2) d_i and q_i have the same label except that nodes labeled with the special label '*' may be matched with data nodes that can have any label from alphabet $\sum$. (3) the edges, i.e., structural (parent-child) relationships between query nodes are satisfied by the corresponding D nodes and the label of both of the edges (in Q and D) have to be exactly the same (again, with the '*' exception). The ordering of sibling nodes in a TPQ query imposes no constraints on the matching. Also, pattern nodes need not be mapped to *distinct* D nodes (the algorithm can be extended to enforce such distinct mappings).

The *TPQ pattern matching problem* is defined as finding all the possible matches of a given TPQ Q in a given graph document D.

3. THE GGQ (GPU GRAPH DATA BASE QUERY) ALGORITHM

The GGQ algorithm is a SIMD algorithm. The main advantage of the GGQ algorithm is the ability to divide the matching work to hundreds or even thousands of threads that run in parallel, and that the work of each thread is exactly of the same length. The idea of the basic version of the algorithm is to use the ID of a thread to determine the portion of the data to which a pattern matching attempt will be executed by the particular thread. Then, as the number of bits in a thread ID is bounded, we designed an extension that al-

lows the algorithm to be efficient also in cases when the query tree or the data graph are more complex. GGQ processes mainly the document parts that are relevant to the input query by processing only edge streams that are relevant for the input query.

The inputs of the algorithm are a labeled directed graph $G = (V, E)$, a TPQ Q, and a set of nodes V_q, subset of V, containing all data graph nodes which are part of legal possible matches for the root node of Q. The algorithm finds all possible matches between Q and G subject to the V_q constraint.

Next, we explain the main idea of the algorithm. For ease of explanation, assume that set V_q has just one node, v_1. Each GPU thread has a unique ID. For example, the ID of thread th is $thNum$, and in binary $thNumb =< b_m, b_{m-1}, ..., b_0 >$. Each node v_i in V has at most $outgNum(lbl)$ outgoing edges with label lbl for each lbl label where $outgNum(lbl)$ is the maximum number of edges labeled lbl connected to a node in the database. I.e., we need $log_2(outgNum(lbl))$ bits to represent $outgNum(lbl)$. For ease of exposition, we assume that $outgNum(lbl)$ for any label is a power of 2. The bits of $thNumb$ define which edge has to be chosen at each step of checking for a match against the data graph.

For example, assume that we have just two types of labels, $lblA$ and $lblB$, in the graph. $outgNum(lblA) = 4, outgNum(lblB) = 16$. Assume that we have a query pattern Q with 3 edges, the first and third edges are with label $lblA$, and the second edge is with label $lblB$. Assume that the maximal thread ID is 255, thus we have 8 bits $< b_7, b_6, ..., b_0 >$ to represent any possible thread ID. Bits b_0 and b_1 represent the index of all possible data edges with label $lblA$, for the first edge in the query pattern. Bits b_2, b_3, b_4, and b_5 represent the index of all possible data edges with label $lblB$, for the second edge in the query pattern. And finally, bits b_6 and b_7 represent the index of all possible data edges with label $lblA$, for the third edge in the query pattern.

We have also tried to reverse the ordering of enumerating the thread ID bits (i.e., to extract the bits from left to right - from MSB to LSB), so that the MSB bit will correspond to the top of the tree. However, the effectiveness of this will fully depend on the nature of the data tree. For the data we used which induces a flat structure on the data tree, this scheme turned out to have inferior performance.

To gain intuition about the algorithm, we start off with an example.

3.1 An Intuitive Example

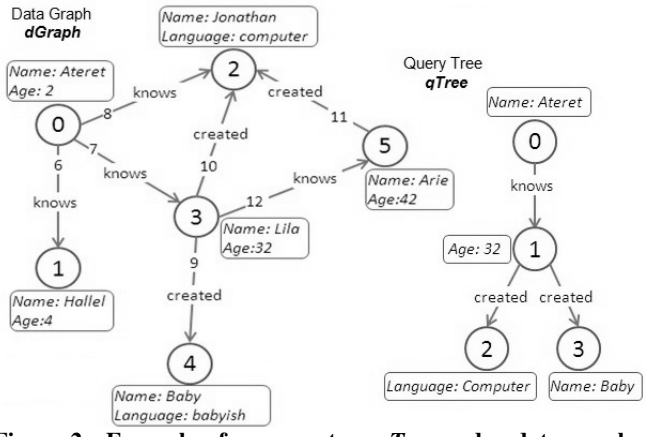


Figure 2: Example of a query tree *qTree* and a data graph *dGraph*

Consider[2] $qTree$ and $dGraph$ presented in Figure 2. Based on $dGraph$, we see that each node $v_i \in V$ has no more than $outgNum(created) = 2$ outgoing edges labelled $created$ and no more than $outgNum(knows) = 3$ outgoing edges labelled $knows$, thus we need 1 bit to represent $outgNum(created)$ and 2 bits to represent $outgNum(knows)$, here bit=0 means edge number 1 and bit=1 means edge number 2.

In total, we need 4 bits to represent all the possible potential matchings with graph $gData$ (shown in Figure 2). During the run, we have 16 different running threads. For ID with bits $< b_3, b_2, b_1, b_0 >$, bits b_0 and b_1 apply to the first edge (between query nodes 0 and 1), bit b_2 applies to the second edge (between query nodes 1 and 2) and bit b_3 applies to the last edge (between query nodes 1 and 3). V_q contains the data node with $ID = 0$. Now, we go over all the threads and explain what happens at run time with each of them.

The thread with ID 0000, finds that the data node with ID 0 has 3 outgoing edges labelled $knows$, according to the first 2 bits "00" of the thread's ID, we choose the first edge which leads us to the data node with ID 1. While checking the data of this node, we find that it does not have label and data "Age: 32". So, this partial matching is not part of an answer, thus the thread terminates with no match. The same behavior happens for threads with ID: 0100, 1000, and 1100.

The thread with ID 0010, chooses the 3-rd outgoing edge (corresponding to the "10" bits) of the data node with ID 0. Thus, it matches the query node with ID 1 to the data node with ID 2. While checking the data of this node, we find that it does not have label and data "Age: 32". Thus, the thread 0010 terminates with no match. The same behavior happens for the threads with ID: 0110, 1010, and 1110.

The thread with ID 0011, chooses the 4-th outgoing edge (corresponding to the "11" bits) of data node with ID 0, but such an edge does not exist. Thus, the thread terminates with no match. The same behavior happens for the threads with ID: 0111, 1011, and 1111.

The thread with ID 0001, chooses the 2-nd outgoing edge of the data node with ID 0. Thus, it matches the query node with ID 1 to the data node with ID 3. The data node with ID 3 has label and data "Age: 32"İ. Next, the thread chooses the 1-st outgoing edge, with label "created", of the data node with ID 3. Thus, it matches the query node with ID2 with the data node with ID 4. The data node with ID 4 has no label and data "Language: computer"İ. Thus the thread terminates with no match. The same behavior happens for the thread with ID 1001.

The thread with ID 0101, chooses the 2-nd outgoing edge of the data node with ID 0. Thus, it matches the query node with ID 1 to the data node with ID 3. The data node with ID 3 has label and data "Age: 32"İ. So, the thread now chooses the 2-nd outgoing edge of the data node with ID 3. Thus, it matches the query node with ID 2 with the data node with ID 2. The data node with ID 2 has label and data "Language: computer"İ. Now, the thread chooses the 1-st outgoing edge of the data node with ID 3 (corresponding to the leftmost "bit" with value 0). Thus, it matches the query node with ID 4 with the data node with ID 4. The data node with ID 4 has label and data "Name: Baby"İ. At this point the tread has finished to match all the nodes and edges. Thus the thread reports that the currently identified assignment of query nodes to data nodes is an answer to $qTree$ in $dGraph$, and terminates with a match. The last thread does not find a match.

3.2 Base algorithm

We specify how the algorithm operates for query Q, graph G, set

[2]In the experiments we used simpler documents in which a single label may be associated with a graph node.

V_q, and a thread with ID $thNum$. For ease of explanation, assume that set V_q has just one node, v_1.

Let $maxthNum$ be the maximal possible thread ID. Let p be $log_2(maxThNum)$, without loss of generality, assume that $maxThNum$ is power of 2. Sort the edges of Q: $e_0, ..., e_w$ so that if edge e_x is on the path from Q's root to the left vertex of edge e_y, then e_x precedes e_y in the order (i.e., "higher" in the tree).

Input: 1) Data graph G. 2) TPQ query Q. 3) V_q set.
Output: $ansSet$, the set of all thread IDs that encode patterns that are an answer to query Q in data graph G.
Method (runs on the CPU):
1. $ansSet = \{\}$
2. Invoke CUDA kernel call for function:
   $GpuGraphQuery(G, Q, V_q, ansSet)$
3. RETURN $ansSet$

// $GpuGraphQuery$ kernel function (runs on the GPU):
Input: 1) Data graph G. 2) TPQ query Q.
3) V_q set of nodes in G that match to the root of Q.
4) $ansSet$ set of all answers (thread IDs).
5) $idConst$. Has default value of 0. Used for algorithm extensions
Goal: In case that current thread's ID encodes an answer to query Q in data graph G, add it into $ansSet$.
Method:
1. Set $thID$ to a system assigned index of the current thread.
2. Set $maxID$ to a system value of the maximal thread index.
3. $thNum = (idConst * (maxID + 1) + thID)$
4. Set $cqIdx$ to 0. /* current query edge index */
5. Express $thNum$ in binary notation as $< b_p, b_{p-1}, ..., b_0 >$.
6. Set $bitIdx$ to 0. /*represents the currently processed bit in the binary notation of $thNum$*/
7. Create $dataNodeArray$ of size $|Q|$ and initialize all its entries to nil. /* the element with index a_i will be data graph node d_n, that corresponds to query node with index a_i */
8. $dataNodeArray[1] = v_r$ /* w.l.o.g. the root index of Q is 1 and v_r is some matching data node (according to V_q).*/
9. FOREACH edge $e_{cqIdx} = (q_a, q_b)$ in Q's edges in order
10. $lbl = e_{cqIdx}.getLabel()$
11. $k = Q.getNumBits(lbl)$ /* According to the graph definition, there are no more than 2^k outgoing edges labeled lbl from any node in G*/
12. Set num to the integer represented by bits $< b_{bitIdx+k-1}, b_{bitIdx+k-2}, ..., b_{bitIdx} >$ of $thNum$. /* These bits corresponds to edge number $cqIdx$ in Q */
13. $currV = dataNodeArray[q_a.idx]$. /*the data graph node to which q_a is mapped*/
14. $currE = currV.getEdge(lbl, num)$ /*gets edge number num out of outgoing edges labeled lbl of node $currV$*/
15. IF ($currE == nil$) THEN RETURN
16. $currV = currE.getTargetNode()$ /*find q_b*/
17. IF (NOT $isMatching(currV, q_b)$) THEN RETURN
18. $dataNodeArray[q_b.idx] = currV$ /*update the mapping array*/
19. $bitIdx = bitIdx + k$ /*prepare $bitIdx$ to read next edge data*/
20. END FOREACH
21. $ansSet.add(thNum)$ /*current thread encodes an answer*/

Figure 3: The base GGQ algorithm

Figure 3 presents the base version of the GGQ algorithm. The input to the algorithm are the data graph G, the TPQ Q and the set V_q that contains the matching data node of the TPQ query root node. Line 2 contains the invocation of the CUDA kernel function $gpuGraphQuery$ which is processed on the GPU. I.e., the query processing algorithm itself is executed on the GPU.

The $gpuGraphQuery$ kernel call finds all the matchings between the TPQ Q and the data graph G. The code of $gpuGraphQuery$ is run in all the threads. They process exactly the same code (i.e., the code of the $gpuGraphQuery$ function itself) at the same time ("Single Instruction") over different data ("Multiple Data") in G. As the possible number of pattern matchings between Q and G is very large, there is a potential for an enormous number of parallel threads. According to the GPU GTX 480 architecture, the maximum number of resident threads per MP (multiprocessor) is 1536 (i.e., 1536*15 for all the MPs), while the number of threads that are processed at any moment of time in the MP is 32 (other threads may be waiting for data from the global memory, or just waiting for their turn to be run). The threads in the GPU are arranged in blocks where each block can have a maximum of 1536 threads. If the requested (by the algorithm) number of threads exceeds 1536*15, then the GPU first handles the 15 first blocks, and then continues to process the next 15 blocks, and so on until all the blocks are processed. Note that maximum number of threads that can actually run in parallel at any point of time is 480 (32 on each of the 15 MPs). The potential number of pattern matchings between Q and G is very large, and is usually much larger than the number of compute units in the GPU. Thus the utilization of the GPU is usually very high, i.e., the throughput of processing the work is high in comparison to multi-threaded CPU systems. By running a profiling tool on GGQ, we found that the execution uses the coalesced memory access feature of the GPU[3]. This is due to an apparent matching between the structure of the storage and the way the algorithm traverses the data.

Next, we explain the $gpuGraphQuery$ kernel function. Line 1 computes the thread's ID, namely $thNum$, according to CUDA's semantics. In line 3 we compute the index for which the current thread is responsible. In the base algorithm, $idConst$ is always 0. Thus, $thNum == thID$. $cqIdx$ that is defined in line 4, indicates the index of the currently processed edge. Line 6 defines the $bitIdx$ variable. $bitIdx$ points to the bit that is currently processed in the binary presentation of $thNum$. The $dataNodeArray$ array which is defined in line 7 holds the data nodes that are matched against query Q by the current thread. I.e., the element with index a_i of the $dataNodeArray$ is the data graph node d_i that is matched to the query node with index a_i. The size of $dataNodeArray$ is the number of nodes in Q, i.e., $|Q|$. In line 8, $dataNodeArray[1]$ is initialized. This is the data node that is matched to the root node of Q. This data node named v_r is taken from set V_q which is one of the parameters of the $gpuGraphQuery$ function.

In line 9 the algorithm starts a FOREACH, that tries to perform a matching between the pattern that is encoded by $thNum$ (the index of the current thread) and G according to TPQ Q. Note that before starting the algorithm, the edges of Q are sorted in a way that if edge e_x is on the path from Q's root to the source vertex of edge e_y, then e_x precedes e_y in the order. And this is the order in which they are processed during the FOREACH. In lines 10-12, the algorithm finds the edge number num that has to be chosen out of the outgoing edges labeled lbl of node $currV$ (a value is assigned to $currV$ in line 13). $currV$ is the node that is matched to the q_a node, which is the source node of the e_{cqIdx} edge. $currV$ is taken out of the $dataNodeArray$ according to the index of the q_a node. The ordering of Q edges (described above) guarantees that $currV$ exists. To find num, the algorithm first extracts the bits of the binary representation of $thNum$ that correspond to the e_{cqIdx} edge. The decimal value that is encoded by these bits is inserted to num. In line 14 the algorithm gets the outgoing lbl labeled edge number num of node $currV$ and assigns it to edge $currE$. If the value of $currE$ is nil, it means that such an edge does not exist, thus according to line 15 the algorithm terminates the run, as this thread does not encode a matching pattern in graph G. In line

[3]This means that when many threads in a warp access consecutive global memory addresses, these memory accesses are grouped into one access.

16-17, using edge $currE$, the algorithm finds the data node that matches to the query node q_b (the target node of edge e_{cqIdx}) and inserts it to $currV$, then it checks the matching between the data of the new $currV$ and the data of q_b. In case it finds that there is no matching between the data of $currV$ and q_b, it terminates the run, as this thread encodes a pattern that does not exist in graph G. In lines 18-19 the data of $dataNodeArray$ and $bitIdx$ is updated, as preparation to the next iteration of the FOREACH. If the algorithm finishes successfully the FOREACH loop for all the edges, without returning in lines 15 or 17, it means that the current thread encodes a pattern that exists in the graph and that fully matches Q. That is why in line 21, the algorithm inserts $thNum$ to $ansSet$.

There are possible optimizations of the basic scheme. As pointed out by a reader, one can base thread addressing on a simple algorithm that takes into account the maximum number of edges with a particular label emanating from a node and, based on the query and the thread ID, deduce the thread's search pattern. This will often result in fewer threads.

3.3 First algorithm extension (Brute Force Looping)

Input: 1) Data graph G. 2) TPQ query Q. 3) V_q set.
Output: $ansSet$, the set of all thread IDs that encode patterns that are an answer to query Q in data graph G.
Method (runs on the CPU):
1. $ansSet = \{\}$
2. $maxIDbitNum = getBinBitsNum(getMaxID())$
   /*$getMaxIDbits$ is a system function*/
3. $maxQBinNum = getBinBitsNum(Q)$
4. FOR ($i = 0; i < 2^{(\lfloor \frac{(maxBin+maxIDbits)}{maxIDbits} \rfloor)}; i++$)
5. Invoke CUDA kernel call for function:
6. $GpuGraphQuery(G, Q, V_q, ansSet, i)$
7. END FOR
8. RETURN $ansSet$

Figure 4: The first extension of the GGQ algorithm

There can be situations in which the maximal number of bits that may be required to represent query patterns is larger than the number of bits of maximal thread ID. Thus, we extend the algorithm as presented in Figure 4. Assume that the maximal thread ID is $maxID$ and that we need $maxIDbits$ to represent it, that $maxBin$ bits are required to represent the query pattern, and that $maxBin > maxIDbits$. In line 4 we start a FOR loop. The number of iterations is: $2^{(\lfloor \frac{(maxBin+maxIDbits)}{maxIDbits} \rfloor)}$. At each loop iteration (line 5), we run the base algorithm, where each thread in the current iteration will take care of the pattern represented by the following number: $(i * (maxID + 1) + threadID)$, where $maxID$ is the ID of the maximal thread ID, and $treadID$ is the system ID of the current thread. This computation can be seen in line 3 of the base algorithm (Figure 3). Note that this way, conceptually, we extend the thread's ID bit representation to the left by placing there the bits corresponding to i in the current loop iteration.

Often, when a query is posed, the desired answer is whether there exist any matching between the query tree and the data graph. In such cases, it is sufficient to find one matching in order to provide a positive answer. In a slightly modified version of the algorithm, the run is stopped the moment a first match is found. This feature decreases the running time of the algorithm in such cases. Sometimes, the desired answer to a query corresponds to only one specific query node and not to all nodes corresponding to the whole set of query nodes. This does not affect the GGQ algorithm as an answer provided by GGQ to a query is an ID of the thread.

3.4 Second algorithm extension (Multi Phase)

A substantial possible improvement, in case that the number of possible patterns is larger than the maximal thread ID, is a two phase exploration (and, in general, a multi phase exploration using the same principle). Here, we first limit the pattern by removing subtrees (actually edges leading to the roots of subtrees) so as to be left with the original rooted pattern with portions removed so that the remaining new pattern Q' is a "prefix" of the original pattern Q. The idea is that we have sufficiently many bits in $maxIDbits$ to explore the smaller Q' (with no need to use the first extension). A Q' node is called a *contact node* if it is a node in Q from which an edge leading to a subtree was removed along with the whole subtree. When evaluating Q' we record for each solution the images in the data graph of the contact nodes of Q' which we call a *recorded solution vector*. Then, we run the second phase in which, for each recorded solution vector, we explore the rest of Q using all the threads we can utilize. In case two phases are not sufficient, we grow Q' to Q in more than 2 phases. Each such phase will produce a collection of recorded solution vectors in which additional Q nodes are assigned values. The advantage of this two phase (and in general multi phase) scheme is that (a) We employ many threads in the first phase working on a smaller query derived from the original query and obtain all the relevant prefixes, encoded in recorded solution vectors, out of the data graph. (b) In the second phase, for each recorded solution vector, we employ all threads on a relevant portion of the data graph that can potentially lead to a solution to Q.

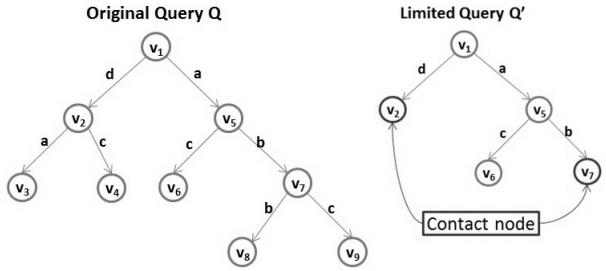


Figure 5: Example of limited Query

For example, consider the query Q as presented on the left side of Figure 5. Suppose that an edge representation requires 4 bits for any label, namely the whole pattern requires 32 bits. Suppose that $maxID$ requires 16 bits. So, we are "missing"İ 16 bits. We can transform Q to the limited query Q' with 4 less edges, as presented on the right side of Figure 5. This way we can handle Q' with all threads (whose $maxID$ requires 16 bits). Once we evaluate Q' we obtain recorded solution vectors. Each recorded solution vector encodes a partial matching of the full matching, and determines the *data contact nodes* v_a and v_b that are matching to the *query contact nodes* v_2 and v_7. When phase 2 is carried out for each recorded solution vector, each thread will operate on the subtrees rooted at v_2 and v_7 where the $dataNodeArray$ will be initialized with v_a in the location corresponding to v_2 and v_b in the location corresponding to v_7. As the subtrees rooted at the contact nodes have a total of 4 edges, 16 bits will suffice to represent all possible navigations. This means that in phase 2, when considering a particular recorded solution vector, **all** threads will be employed in checking possible continuations for this recorded solution vector. Thus, the computing power is fully utilized in (the short) phase 1 and later on throughout phase 2. Note that there is an advantage here over the loop scheme (that is presented in the first extension) in that for a loop index that corresponds to a non-prefix of the data graph, all

GPU threads are activated in vain. Here, the first phase guarantees that the sequence of GPU activations is done for recorded solution vectors that correspond to potentially extendable matchings. The disadvantage is that the recorded solution vectors need be stored so that they are available for the second phase.

Input: 1) Data graph G. 2) TPQ query Q. 3) V_q set.
Output: $ansSet$, the set of all thread IDs that encode patterns that are an answer to query Q in data graph G.
Method (runs on the CPU):
1. $prelimAnsSet = \{\}$
2. $prefixQ = getPrefixQ(Q)$ /*$getPrefixQ$ returns the "prefix" of the query Q*/
3. Invoke CUDA kernel call for function:
4. $\quad GpuGraphQuery(G, prefixQ, V_q, prelimAnsSet, 0)$
5. $ansSet = \{\}$
6. $remainQ = getRemainQ(Q, prefixQ)$ /*$getRemainQ$ returns $Q \setminus prefixQ$*/
7. FOREACH ans in $prelimAnsSet$
8. $\quad currAnsSet = \{\}$
9. $\quad$ Invoke CUDA kernel call for function:
10. $\quad\quad GpuGraphQueryExt(G, remainQ, currAnsSet, ans)$
11. $\quad ansSet = ansSet \cup currAnsSet$
12. END FOREACH
13. RETURN $ansSet$

// $GpuGraphQueryExt$ kernel function (runs on the GPU, just the differences from $GpuGraphQuery$ presented):
Input: 1) Data graph G. 2) forest $remainQ$.
3) $ansSet$ set of all answers (thread IDs).
4) $baseAns$ is the ID that encodes the matching between $prefixQ$ and G
Goal: In case that current thread's ID encodes an answer to query $remainQ$ based on the matching presented by $baseAns$ in data graph G, add it into $ansSet$.
Method:
...
3. $thNum = thID$
...
8. $initNodesArray(dataNodeArray, baseAns)$
/* $initNodesArray$ extracts $baseAns$, and fill all the nodes that already matched in dataNodeArray by answering $prefixQ$ in the first phase */
...

Figure 6: The second extension of the GGQ algorithm

Figure 6 presents the second extension to the algorithm. The function $getPrefixQ$ (line 2), decides which part of Q is going to be the "prefix" query. It makes the decision based on the number of bits $bLimit$ required to present the maximal thread ID, and on the structure of Q. Basically, it chooses the "upper" part of the tree (the part with the smallest depth), up to the limit of $bLimit$. I.e, it sums the number of bits that are required to present all the edges of the chosen part, and enlarges the chosen part up to the limit of $bLimit$. Lines 3,4 run the base algorithm on $prefixQ$, and insert the answer into $prelimAnsSet$. Line 5 initialize $ansSet$, the set of the final answers. $remainQ$ that is computed in line 6, is the remainder part of Q after removing $prefixQ$ out of it. Line 7 starts a FOREACH that computes the final answers for Q based on the preliminary answers from $prelimAnsSet$. The set of answers of the current iteration, $carrAnsSet$ is defined in line 8. Lines 9-10, contain the invocation of the CUDA kernel function $gpuGraphQueryExt$ which is processed on the GPU, and is slightly different from $gpuGraphQuery$ (as defined in Figure 3). In line 11 we add the answers that were found in the current iteration to the final answers set, namely $ansSet$.

Next we describe $gpuGraphQueryExt$. This function has slight differences from the base algorithm GPU function, $gpuGraphQuery$. Thus, we describe just these differences. The first difference is in line 8, in which the $thNum$ is defined. $thNum$ is equal to the system value of the ID of the current thread. The second difference is in line 8, in which the $initNodesArray$ function initializes $dataNodeArray$. The function extracts from $baseAns$ the matchings between nodes in Q and nodes in G that were found during the first phase, and assigns the found data nodes into the appropriate places in $dataNodeArray$. Except for the described two changes, the function operates exactly as the base $gpuGraphQuery$ function.

The number of edges that can be represented by one phase is n such that $\sum_{edge=1}^{n} outgNum(lbl(e)) \leq 2^{bitsNum(maxThreadID)}$ where $bitsNum(maxThreadID)$ is the number of bits that are used by the GPU to represent the maximal thread ID. For example, assume that a GPU thread ID is represented with 32 bits. Assume that for each edge e, on average, there are 16 potential $outgNum(lbl(e))$ from each node. Thus, on average, we need 4 bits to represent each edge of the query. Based on the above, each phase allows us to represent $32/4 = 8$ edges on average. Having 2 phases in the multi-phase extension described above allows us to represent fairly large TPQs with about 16 edges. The multi-phase extension can be easily extended to more than 2 phases. Based on this analysis, if we extend the multi-phase extension to 3 phases, we can represent a TPQ with 24 edges, which is a very large query. It is important to note that without the multi phase extension, experiments involving very large queries give very poor results that are worse than Gremlin's performance on these queries.

Input: 1) query edges ($qEdges$). 2) $maxQdepth$, the max depth of Q
3) $maxBitsNum$, the number of bits required to represent maximal thread ID
Goal: to set the field $phaseNum$ of each query edge
Method (runs on the CPU):
1. $currPhase = 1$
2. $currBitsSum = 0$
3. FOR $depth$ FROM 1 TO $maxQdepth$
4. $\quad$ FOREACH $edge$ IN $qEdges$
5. $\quad\quad$ IF $edge.depth == depth$
6. $\quad\quad\quad$ IF $currBitsSum + edge.bitsNum > maxBitsNum$
7. $\quad\quad\quad\quad currPhase++$
8. $\quad\quad\quad\quad currBitsSum = 0$
9. $\quad\quad\quad$ END IF
10. $\quad\quad\quad currBitsSum += edge.bitsNum$
11. $\quad\quad\quad edge.phaseNum = currPhase$
12. $\quad\quad$ END IF
13. END FOREACH
14. END FOR

Figure 7: Query phase ordering algorithm

Figure 7 presents the algorithm for breaking the query into phases.

4. EXPERIMENTAL EVALUATION

We compared GGQ to Gremlin [17] in terms of run time (to completion). Gremlin is the only query processor that we found that uses the *native graph* approach and that supports XPath-style queries over graph documents. Using Gremlin's query language, one can easily express TPQs. We are not aware of any parallel graph query processor to which we can currently compare our results. We used the GLS storage scheme to store the data. We implemented the GGQ algorithm from scratch on CUDA [7]. We experimented with GRR [10], a benchmark tool for generating random RDF documents. We also experimented with the Geospecies data document [2], and a representative data document example of the Census database [8]. We checked different TPQ query patterns. [4]

[4]Queries and data are available upon request.

	Path Q1	Path Q2	Speedup	
			Q1	Q2
$Gremlin$	114	84		
$GPU-full$	0.8	7.4	148	11
$GPU-ans$	0.09	0.08	1267	1050
$GPU-alg$	0.085	0.075	1425	1200

Figure 8: Results of GGQ on a document with size 125MB, for path queries with 5 nodes. The right two columns contain the speedup of GGQ run in comparison to a Gremlin run.

	Path Q1	Tree Q2	Speedup	
			Q1	Q2
$Gremlin$	81	75		
$GPU-full$	0.86	0.67	94	112
$GPU-ans$	0.09	0.12	900	625
$GPU-alg$	0.08	0.11	1012	682

Figure 10: Results of GGQ on a document with size 180MB, for a path query with 4 nodes and a tree query with 6 nodes. The right two columns contain the speedup of GGQ run in comparison to a Gremlin run.

All experiments were run on an 3 GHz Intel S5520SC ShadyCove 5520 12DDR3 6SATA/R 2LAN1000 EATX workstation having an NVIDIA GTX 480 GPU (with 1.5GB global memory), and having two Intel Xeon 6C X5650 processors (with 24GB of RAM in total). Each Xeon processor has 6 cores so altogether the workstation has 12 cores. We used the actual run time in various scenarios as the main metric of performance.

4.1 Experiments Description

Setting Up. An experiment run has two input files: an RDF document, and a text file with query (TPQ) patterns to run against the given document. An experiment begins with loading the input document into the *GLS storage* system by the parser. Then, we parse the queries, and process them against the input document. We used different TPQ patterns. The patterns we used have different length and of different tree structures.

Experiment Description. The document is first loaded to the GLS storage system (the time of loading is not measured, as it is a one time procedure). Every experiment has the following runs:

1. *Gremlin Run* - We process the queries in the queries text file using Gremlin [17]. Information regarding the run time of the query is collected in the result log file.

2. *GPU Run* - This run is performed using the GPU. We process the queries in the queries text file. The queries are processed by the GGQ algorithm as described in Section 3. Information regarding start and end times of processing the queries is collected in the result log file.

We compare the performance of GGQ to Gremlin by comparing the run time of these algorithms in three different ways. In the first way we start the time measurement for the GGQ algorithm before copying the data from the CPU to the global memory of the GPU, and stop after copying the result data from the GPU to the CPU (namely, *GPU-full*). In the second way we start the time measurement for the GGQ algorithm right after copying the data from the CPU to the global memory of the GPU, and before the query execution begins, and stop the time measurement right after finishing the query processing, but before copying the results data from the global memory of the GPU to the CPU (namely, *GPU-alg*). In the third way we start the time measurement for the GGQ algorithm right after copying the data from the CPU to the global memory of the GPU, and before the query execution begins, and stop the time measurement after copying the result data from the GPU to the CPU (namely, *GPU-ans*). *GPU-full* reflects the potential time improvement of the GPU for large documents that cannot fully reside in the global memory of the GPU. *GPU-ans* reflects the potential time improvement for documents that can fully reside in the global memory of the GPU. This is an important measurement as in a case that the document can fully reside in the GPU, we have to copy it to the GPU only once and then we can run many queries over this document in a row, by this eliminating the need for copying the document to the GPU per each query. *GPU-alg* is appropriate for GPUs in which the global memory and RAM are merged, i.e., in more recent processors such as NVIDIA's planned PASCAL GPU family. Time is measured is milliseconds. Each experiment is characterized by the size of the input RDF document. We experimented with documents sized as follows: 40MB, 125MB,180MB, 600MB. We did not use larger files, as the GRR benchmark tool was not able to create larger files. Also, the main factor that influences the complexity of GGQ is the size of the query and not the size of the RDF database document. Note that only the relevant edge streams have to be copied to the global GPU memory, so ordinarily the amount of data that is copied to the global GPU memory is usually much smaller than the document size.

	Path Q1	Tree Q2	Speedup	
			Q1	Q2
$Gremlin$	76	72		
$GPU-full$	3.2	3.17	24	23
$GPU-ans$	0.08	0.09	950	800
$GPU-alg$	0.075	0.095	1085	900

Figure 9: Results of GGQ on a document with size 600MB, for a path query with 5 nodes and a tree query with 6 nodes. The right two columns contain the speedup of GGQ run in comparison to a Gremlin run.

	Tree Q1	Tree Q2	Speedup	
			Q1	Q2
$Gremlin$	187	165		
$GPU-full$	49	5.4	3.8	30.6
$GPU-ans$	48	2.6	3.9	63.5

Figure 11: Results of GGQ on a document with size 180MB, for two different tree queries with 11 nodes. The right two columns contain the speedup of GGQ run in comparison to a Gremlin run.

4.2 Experiments

Figures 8 and 9 show the results of GGQ on GRR documents with sizes 125MB and 600MB respectively for different TPQ queries. The GGQ run with full memory transferring time (both directions) has speedup with respect to Gremlin of about 147 and 11 for a document with size 125MB and of 24 and 23 for a document with size 600MB, for Q1 and Q2 respectively. The GGQ run with result transferring time (from the global memory to the CPU) has speedup with respect to Gremlin of 1267 and 1050 for a document with size 125MB and of 950 and 800 for a document with size 600MB, for Q1 and Q2 respectively. The GGQ pure run (without transferring times) has speedup with respect to Gremlin of 1425 and 1200 for a document with size 125MB and of 1086 and 900 for a document with size 600MB, for Q1 and Q2 respectively. Due to lack of space, we shall not elaborate on all the experimentation Figures.

5. RELATED WORK

There are a growing number of initiatives to implement and commercialize Graph databases, such as Neo4j [6], HyperGraphDB [4] and DEX [3] and many RDF solutions such as Jena [5] and AllegroGraph [1]. There are other initiatives to create graph querying languages that enable a simplified user view of querying such as SPARQL [19] and Gremlin [17]. Another initiative for a graph query language is GraphQL that is presented in [12] in which the base node is a graph, so it deals with a graph of graphs. Thus, the answer to this query is a set of graphs; further, this work is not dealing with parallel query processing. Works in the area of parallelization of graph databases have started to appear. For example, parallelGDB [9] and papers that address parallelization that is based on graph partitioning. GPU-based work is [14] which proposes an efficient subgraph matching algorithm. It presents an implementation of the STwig algorithm [18] in which the third (join) step of the algorithm is performed in parallel on a GPU.

Lately, there are efforts to use GPUs to improve the performance of DBMSs. There are also new framework proposals, such as Medusa, a programming framework for parallel graph processing on GPUs. Medusa enables developers to leverage the massive parallelism and other hardware features of GPUs by writing sequential C/C++ code for a small set of APIs. Recent works, [15], propose efficient XML path processing algorithms using GPUs, which deal with path patterns. The current paper, on the other hand, deals with TPQs, which are more complex query patterns, looked for on more complex database structures.

6. CONCLUSIONS

We present the GGQ algorithm, a novel efficient algorithm for processing TPQ queries on graph documents. We use a new storage scheme, GLS, in a parallel multi-threaded computing platform, using a GPU as a CPU co-processor. GGQ employs techniques that allow it to run hundreds of threads in parallel.

We conducted extensive experimentation with GGQ. We compared, in terms of run time, GGQ to Gremlin [17], currently the only available tool for comparison, that supports XPath-style queries over graph documents. We checked performance for varying document sizes and for different queries. Experimental results indicate that using GGQ significantly reduces the run time of queries in comparison to Gremlin.

As part of future work, we plan to adapt the multi-phase scheme to oddly shaped graphs, e.g., ones with a few nodes, each having a multitude of edges. We also plan to extend GGQ to handle queries that are in the form of a directed graph. The idea is to first build a spanning forest out of the query graph. Then, to run the above algorithm on each tree in the forest. As the last step, to check all the answers for compatibility (namely, that the same query node is not mapped to different data graph nodes) and retain the answers that conform to the graph query structure.

7. REFERENCES

[1] Allegrograph: Modern, high-performance, persistent graph database. http://franz.com/agraph/allegrograph/.

[2] The apache xalan project. http://stats.lod2.eu/rdfdocs/769.

[3] Dex: High-performance and scalable graph database management system. http://www.sparsity-technologies.com/dex.

[4] Hypergraphdb. http://www.hypergraphdb.org/index.

[5] Jena: Java framework for building semantic web and linked data applications. http://jena.apache.org/.

[6] Neo4j: WorldŠs leading graph database. http://www.neotechnology.com/neo4j-graph-database/.

[7] Nvidia cuda c programming guide.

[8] Rdf data sets repository links. http://www.w3.org/wiki/DataSetRDFDumps.

[9] L. Barguñó, V. Muntés-Mulero, D. Dominguez-Sal, and P. Valduriez. Parallelgdb: a parallel graph database based on cache specialization. IDEAS '11.

[10] D. Blum and S. Cohen. Grr: Generating random rdf. In *ESWC (2)*, 2011.

[11] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal xml pattern matching. In *SIGMOD'02*.

[12] H. He and A. K. Singh. Graphs-at-a-time: Query language and access methods for graph databases. SIGMOD '08.

[13] J. Hensley. Amd ctm overview. In *SIGGRAPH'07*.

[14] X. Lin, R. Zhang, Z. Wen, H. Wang, and J. Qi. Efficient subgraph matching using gpus. In *Databases Theory and Applications*, Lecture Notes in Computer Science. 2014.

[15] R. Mousalli, R. Halstead, M. Salloum, W. Najjar, and V. J. Tsotras. Efficient xml path filtering using gpus. In *ADMS - VLDB Workshops*, 2011.

[16] NVIDIA. What is gpu-computing? http://www.nvidia.com/object/what-is-gpu-computing.html.

[17] M. A. Rodriguez. Gremlin, 2010.

[18] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *VLDB'12*.

[19] W3C. Sparql. http://www.w3.org/TR/rdf-sparql-query/.

Implementing Flexible Operators for Regular Path Queries

Petra Selmer
London Knowledge Lab
Birkbeck, University of London
lselm01@dcs.bbk.ac.uk

Alexandra Poulovassilis
London Knowledge Lab
Birkbeck, University of London
ap@dcs.bbk.ac.uk

Peter T. Wood
London Knowledge Lab
Birkbeck, University of London
ptw@dcs.bbk.ac.uk

ABSTRACT

Given the heterogeneity of complex graph data on the web, such as RDF linked data, a user wishing to query such data may lack full knowledge of its structure and irregularities. Hence, providing users with flexible querying capabilities can be beneficial. The query language we adopt comprises conjunctions of regular path queries, thus including extensions proposed for SPARQL 1.1 to allow for querying paths using regular expressions. To this language we add two operators: APPROX, supporting standard notions of approximation based on edit distance, and RELAX, which performs query relaxation based on RDFS inference rules. We describe our techniques for implementing the extended language and present a performance study undertaken on two real-world data sets. Our baseline implementation performs competitively with other automaton-based approaches, and we demonstrate empirically how various optimisations can decrease execution times of queries containing APPROX and RELAX, sometimes by orders of magnitude.

1. INTRODUCTION

The volume of graph-structured data on the web continues to grow, most recently in the form of RDF Linked Data. At the time of writing, there are 570 large datasets, spanning a variety of domains, such as the life sciences, geographical and government domains [2]. The prevalence of graph databases, such as Sparksee [21], Neo4j [14] and OrientDB [16], has also greatly increased over the past few years; they have been used in areas as diverse as social network analysis, recommendation services [20] and bioinformatics [1].

Graph-structured data in these domains may be complex, heterogeneous and evolving in terms of its structure, making it difficult for users to formulate queries that precisely match their information retrieval requirements. In this paper, we discuss the development of efficient algorithms for approximate matching and relaxation of conjunctive regular path (CRP) queries over such data, with the aim of assisting users in formulating queries and interactively retrieving results that are of relevance to them. Query results are returned incrementally to the user in order of their increasing edit or relaxation distance from the original query, with users being able to specify a limit on the number of results returned in each phase.

This paper extends earlier work in [9, 18], where the APPROX and RELAX operators were introduced, and in [17], where an initial prototype implementation was described, by describing our system implementation, called *Omega*, in detail. We also undertake here a performance study on real-world data sourced from adult further education [17] and from YAGO [10]. This study demonstrates that the performance of exact query evaluation is competitive with other automaton-based approaches, while a number of novel optimisations improve the performance of queries containing APPROX and RELAX, sometimes by orders of magnitude.

In Section 2 we give the necessary background and motivation, introducing our graph-based data model and query language. In Section 3 we discuss the implementation of *Omega*. We present our performance study in Section 4. In Section 5 we review related work in CRP query evaluation for graph-structured data. Section 6 summarises the contributions of the paper, gives our concluding remarks and directions for further work.

2. BACKGROUND AND PRELIMINARIES

Omega uses a general graph-structured data model comprising a directed graph $G = (V_G, E_G, \Sigma)$ and a separate ontology $K = (V_K, E_K)$. The set V_G contains nodes each representing an entity instance or an entity class, while the set $E_G \subseteq V_G \times (\Sigma \cup \texttt{type}) \times V_G$ represents relationships between members of V_G. For an edge $e = (x, l, y) \in E_G$, l is called the *label* of e, x the *source* of e, and y the *target*. We assume that the *alphabet* Σ is finite. The label `type` is used to connect an entity instance to its class, and can represent the corresponding notion in RDF/S (see below).

The set V_K contains nodes each of which represents an entity class or a property. We call a node in V_G or V_K that represents an entity class a *class node* and a node in V_K that represents a property a *property node*. So $V_G \cap V_K$ consists of all the class nodes of V_G.

The edges in E_K capture subclass relationships between class nodes, subproperty relationships between property nodes, and domain and range relationships between property and class nodes. Hence, $E_K \subseteq V_K \times \{\texttt{sc}, \texttt{sp}, \texttt{dom}, \texttt{range}\} \times V_K$. We assume that $\Sigma \cap \{\texttt{type}, \texttt{sc}, \texttt{sp}, \texttt{dom}, \texttt{range}\} = \emptyset$. We also assume that the set of labels of property nodes in V_K does not contain the label `type`.

This general graph model encompasses RDF data, except that it does not allow for the representation of RDF's 'blank' nodes; however, blank nodes are discouraged for linked data [7]. Our graph model also encompasses a fragment of the RDFS vocabulary: `rdf:type`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, and `rdfs:range`, which we abbreviate by the symbols `type`, `sc`, `sp`, `dom`, and `range`.

The query language underlying *Omega* is that of *conjunctive regular path queries* [3]. A conjunctive regular path (CRP) query Q, consisting of n conjuncts, is of the form

$$(Z_1, \ldots, Z_m) \leftarrow (X_1, R_1, Y_1), \ldots, (X_n, R_n, Y_n)$$

where $m, n \geq 1$, each X_i and Y_i is a variable or a constant, each Z_i is a variable appearing in the right-hand-side of Q, and each R_i is a *regular expression* over the alphabet from which edge labels in the graph are drawn. In our context, a regular expression R is defined as follows:

$$R := \epsilon \mid a \mid a- \mid \_ \mid (R1 \cdot R2) \mid (R1|R2) \mid R^* \mid R^+$$

where ϵ is the empty string, a is any label in $\Sigma \cup \{\texttt{type}\}$, $a-$ represents traversal of an edge in the reverse direction, "_" denotes the disjunction of all constants in $\Sigma \cup \{\texttt{type}\}$, and the operators have their usual meaning.

The (exact) *answer* to a CRP query Q on a graph G can be obtained in a standard way by finding all pairs of nodes in G satisfying each conjunct, joining the results, and projecting over the variables in the head of Q.

EXAMPLE 1. *Suppose a user wishes to find people who graduated from an institution located in the UK and poses the following query, Q, over the YAGO graph [10]:*

```
(?X) <- (UK,isLocatedIn-.gradFrom,?X)
```

(Variables in a query have an initial question mark.) This query returns no results since it requires that there is some entity y, located in the UK, which has graduated from some institution. However, no such y exists, since only people can graduate from an institution and only events and places can be located in a country.

The work in [9] investigated *approximate* matching of CRP queries, allowing edit operations such as insertions, deletions and substitutions of edge labels to be applied to the regular expressions R_i of a CRP query, each with some edit cost configurable by the user.

EXAMPLE 2. *In* Omega*, the user can submit a variant of Q in which the conjunct can be approximated:*

```
(?X) <- APPROX (UK,isLocatedIn-.gradFrom,?X)
```

`isLocatedIn-.gradFrom` *is approximated by* `isLocatedIn-.gradFrom-`*, at some edit distance* α*, by substituting* `gradFrom` *with* `gradFrom-`*. This query now returns results, matching the user's original intention by correcting the error in Q.*

The work in [18] also considered applying ontology-based *relaxation* to the regular expressions R_i. This allows query relaxations entailed using information from the ontology K, in particular: (i) replacing a class/property label by that of an immediate superclass/superproperty, at some cost β; (ii) replacing a property label by a `type` edge with target the property's domain or range class, at some cost γ.

EXAMPLE 3. *In* Omega*, the user can submit a variant of Q in which the conjunct can be relaxed:*

```
(?X) <- RELAX (UK,isLocatedIn-.gradFrom,?X)
```

This query allows `gradFrom` *to be relaxed to its parent property* `relationLocatedByObject` *at cost* β*, which now allows properties such as* `happenedIn` *and* `participatedIn` *to be matched, and answers to be returned at 'distance'* β*.*

3. IMPLEMENTATION OF *OMEGA*

Figure 1 illustrates the architecture of the *Omega* system. Sparksee [21] (formerly DEX) is used as the data store. The development was undertaken using the Microsoft .NET framework. The system comprises four components: (i) the *console* layer, in which queries are submitted, and which displays the results; (ii) the *system* layer in which query plans are constructed and executed; (iii) the Sparksee API; and (iv) the data store itself.

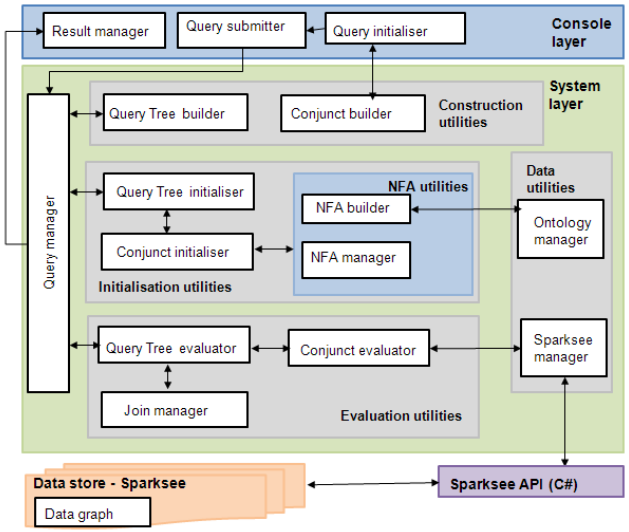


Figure 1: System architecture

The architecture of the *system* layer broadly follows that described in [17], with the major change being that the data store used in *Omega* is Sparksee [21] rather than XML. This layer is responsible for the construction of the automaton (NFA) corresponding to each query conjunct. Given a query conjunct (X, R, Y), a weighted NFA M_R is constructed to recognise the language denoted by the regular expression R. If the conjunct is prefixed by APPROX or RELAX, then M_R is augmented to produce an automaton A_R or M_R^K, respectively; we discuss this in Section 3.3. Further responsibilities of the *system* layer include the construction of the query tree, the incremental construction of a weighted product automaton H_R from each conjunct's automaton and the data graph G, and the evaluation of the overall query, including performing a ranked join for multi-conjunct queries. We make extensive use of data structures provided by the **C5 Generic Collection** library [15].

3.1 The Sparksee Data Model and API

The two main Sparksee structures used in our implementation are nodes and edges (which may be directed or undirected), each of which has a pre-created *type* (this is a label,

of *string* data type), and a unique object identifier (*oid*) of *long* data type. Associated with each node and edge are zero or more attributes, which are key-value pairs; values may be of any primitive data type. Further details regarding Sparksee may be found in [12, 13] and the User Manual[1]. The main Sparksee API functions used in *Omega* are as follows:

`Neighbors` takes as arguments a node n and edge type t, and returns the set of nodes connected to n via an edge of type t; the directionality of edges may also be specified.

`Heads` takes a set of edges E, and returns the set of nodes which are the target of an edge in E. `Tails` is analogous to `Heads`, except that nodes which are sources are returned. `TailsAndHeads` returns the union of `Heads` and `Tails`.

To store the data, Sparksee uses a combination of maps (inverted indexes) and associated bitmap vectors [13]. To improve the performance of the `Neighbors` function, an option may be set to index the neighbouring nodes when creating an edge type t. This means that an index entry is created when an edge of type t is created between any two nodes. Node- and edge-related attributes may also be configured to be indexed when they are created (the index stores all *oids* associated with each value of the attribute).

3.2 Omega data graphs

As it is mandatory in Sparksee for each node to have a type, and as our data model does not assume that nodes are typed, we create all of our nodes to be of the same type, 'node'. All of our nodes have one attribute, of *string* data type, representing the node label (which is unique in the data graph G). This attribute has indexing enabled.

We create multiple edge types, all of which are defined to be *directed* edge types with indexing enabled. Specifically, for each edge in G having label $l \in \Sigma$, two Sparksee edges are created: (i) one having type l, and (ii) one having type 'edge' with an associated indexed *string*-valued attribute corresponding to l. We introduce the generic 'edge' type to counter a limitation of the `Neighbors` function which requires the type of the edge to be provided as an argument, in order to allow us easily to retrieve multiple types of edges simultaneously. For each edge in G labelled `type`, only one edge is created, whose type is `type`. In cases which require the retrieval of *all* types of edges of a node, we retrieve all 'edge' edges, followed by all `type` edges.

3.3 Query conjunct initialisation

The initialisation of a query conjunct (X, R, Y) comprises the construction of the associated automaton (one of M_R, A_R or M_R^K), and the initialisation of its data structures prior to the evaluation of the conjunct. We discuss each here.

In all cases, an automaton (NFA) M_R is first constructed from regular expression R using standard techniques. Then, if the conjunct is prefixed by APPROX or RELAX in the query, additional transitions and states are added (see [18]), along with the removal of ϵ-transitions, to form A_R or M_R^K respectively. As the automaton is weighted, the removal of ϵ-transitions may result in final states having an additional, positive weight [5]. For state s, we denote this weight by $weight(s)$. The NFA is represented as a set of transitions (s, a, c, t), where s is the 'from' state, t is the 'to' state, a is the label, and c is the cost.

If the conjunct is APPROXed, the *insertion* edit operation would result in many additional transitions in the NFA, one for each label in $\Sigma \cup$ `{type}` and their reversals. To make our automaton more compact, we represent these as a single transition labelled by the wildcard label $*$.

In all cases, if X (respectively, Y) is a constant c, we annotate the initial (resp. final) state with c; otherwise we annotate the initial (resp. final) state with the wildcard symbol matching any constant.

The pseudocode for the initialisation of a conjunct is given in the Open procedure below. After constructing the appropriate automaton, the procedure evaluates the conjunct by traversing the automaton and the data graph simultaneously. This traversal is represented by tuples of the form (v, n, s, d, f), where d is the distance associated with visiting node n in state s having started from node v, and f denotes whether the tuple is 'final' or 'non-final' (see below).

The tuples are added to and removed from a dictionary D_R whose key is an integer-boolean variable (where the integer portion represents a distance and the boolean portion represents the final or non-final tuples at that distance). The value associated with each key is a linked list of tuples. Tuples are always added to, and removed from, the head of a linked list (at cost $O(1)$). We introduced the notion of final and non-final tuples in order to prioritise the removal of 'final' tuples (rather than 'non-final' ones) at the minimum distance (if any), so that answers may be returned earlier. Including this refinement improved the performance of most of our queries, and also ensured that some queries, which had previously failed by running out of memory, completed.

Procedure Open

```
   Input: query conjunct (X, R, Y)
(1) construct NFA M_R for R; initial state is s_0
(2) transform M_R into A_R (APPROX) or M_R^K (RELAX) if
    necessary
(3) visited_R ← ∅
(4) d ← 0
(5) if conjunct is of the form (C, R, ?X) then
(6)     //Let n be the node in G corresponding to C
(7)     if RELAX is being applied then
(8)         foreach node m ∈ GetAncestors(n) do
(9)             add(D_R, (m, m, s_0, d, false))
(10)    else
(11)        add(D_R, (n, n, s_0, d, false))
(12) else
(13)    //the conjunct is of the form (?X, R, ?Y)
(14)    if s_0 is final then
(15)        if weight(s_0) = 0 then
(16)            foreach node n in G do
(17)                add(D_R, (n, n, s_0, d, true))
(18)        else
(19)            foreach n ∈ GetAllNodesByLabel(s_0) do
(20)                add(D_R, (n, n, s_0, d, false))
(21)    else
(22)        foreach n ∈ GetAllStartNodesByLabel(s_0) do
(23)            add(D_R, (n, n, s_0, d, false))
```

We distinguish between 3 cases in the Open procedure: (*Case 1*) If the conjunct is of the form $(C, R, ?Y)$ where C

[1] `www.sparsity-technologies.com/downloads/UserManual.pdf`

is a constant, we begin the traversal at the node in G having the attribute value C.

(*Case 2*) A conjunct of the form $(?X, R, C)$ is transformed to $(C, R^-, ?X)$, where R^- is the *reversal* of R. This reversal can be accomplished in linear time starting from the NFA for R [23]. Thus, *Case 2* reverts to *Case 1*.

If the conjunct has the RELAX operator and C is a class node, we also add to D_R every node returned by $GetAncestors$ (line 8). This function returns all superclasses of C in order of increasing specificity so that they are added to the list in D_R in that order. We want to process more specific classes first, given that nodes representing more general classes will have larger degree (owing to transitive closure) and will lead to answers of greater cost.

(*Case 3*) For a conjunct of the form $(?X, R, ?Y)$, lines 14 to 23 are invoked. The function $GetAllNodesByLabel$ (line 19) takes as input a list of all labels on transitions whose 'from' state is the initial state s_0. Each label in the list is then processed as follows: (i) the directionality of the label is determined — i.e. whether it is an incoming or an outgoing edge, or whether both incoming *and* outgoing edges are required (as for the *-labelled transitions, introduced above); (ii) the set of object identifiers (oids) for the nodes having the relevant edge and directionality are retrieved using the Sparksee methods `Heads`, `Tails` and `TailsAndHeads`; (iii) Sparksee set operations are used to maintain a distinct set of nodes, so that the same node is not re-added to D_R at a higher cost (this can occur with the '*' label); and (iv) the remaining nodes in the graph G are returned. When adding to D_R, we iterate through the set of nodes in order of decreasing cost. The function $GetAllStartNodesByLabel$ (line 22) is identical to $GetAllNodesByLabel$, except that it does not include step (iv).

We have implemented the above two functions and that retrieving all nodes in G (line 16) as coroutines in conjunction with the $GetNext$ procedure (discussed in Section 3.4), incrementally obtaining nodes in batches (the default is 100 nodes at a time). We found that, as a result, the execution time of some queries was reduced by half, since nodes not required to answer the user's query are not added to D_R.

Procedure GetNext(X, R, Y)

Input: query conjunct (X, R, Y)
Output: triple (v, n, d), where v and n are instantiations of X and Y

```
(1)  while nonempty(D_R) do
(2)      (v, n, s, d, final) ← remove(D_R)
(3)      if final then
(4)          if ∄d'.(v, n, d') ∈ answers_R then
(5)              append (v, n, d) to answers_R
(6)              return (v, n, d)
(7)      else
(8)          if (v, n, s) ∉ visited_R then
(9)              add (v, n, s) to visited_R
(10)             foreach -d'-> (s', m) ∈ Succ(s, n) s.t.
                 (v, m, s') ∉ visited_R do
(11)                 add(D_R, (v, m, s', d + d', false))
(12)             if s is a final state and its annotation
                 matches n and ∄d'.(v, n, d') ∈ answers_R
                 then
(13)                 add(D_R, (v, n, s, d + weight(s), true))
(14)     //Incrementally add the next batch of initial nodes
(15)     if no distance 0 tuples in D_R and more initial nodes
         available then
(16)         foreach initial node n' do
(17)             add(D_R, (n', n', s_0, 0, false));
(18) return null
```

3.4 Query conjunct evaluation

The two algorithms concerned with the evaluation of a single query conjunct are $GetNext$ and $Succ$, which have previously been presented in [9, 18]. We now describe our physical implementation of these algorithms.

GetNext returns the next query answer, in order of non-decreasing distance from the original query Q, by repeatedly removing the first tuple (v, n, s, d, f) from the distance d list of D_R until D_R is empty. If the removed tuple is final (f is $true$) and the answer (v, n, d') has not been been generated before for some d', the triple (v, n, d) is returned after being added to $\texttt{answers}_R$. If the tuple is not final, we add (v, n, s) to $\texttt{visited}_R$, and add $(v, m, s', d + d', false)$ to D_R for each transition $\xrightarrow{d'}(s', m)$ returned by $Succ(s, n)$ such that $(v, m, s') \notin \texttt{visited}_R$. If s is a final state, its annotation matches n, and the answer (v, n, d') has not been been generated before for some d', then we add the weight of s to d, mark the tuple as final, and add the tuple to D_R.

For $visited_R$, we use a hashed set which has $O(1)$ lookup time. Lines 8 and 9 in practice are executed as a single step, and the logic in lines 10 to 13 is only executed if the item was added. This means that we never re-process a previously-processed (v, n, s) triple; this situation may arise when (v, n, s) triples of monotonically-increasing distances are created and added at lines 11 and 13 (we therefore never process 'duplicate' tuples at a higher distance).

In lines 15 to 17, we utilise a coroutine for $(?X, R, ?Y)$ conjuncts. If D_R no longer contains any tuples at distance 0, we retrieve and add the next batch of initial nodes from the functions initially invoked in the *Open* procedure.

The *Succ* function takes as input a node (s, n) of the weighted product automaton H_R and returns a set of of transitions $\xrightarrow{d}(p, m)$, such that there is an edge in H_R from (s, n) to (p, m) with cost d. The function $NextStates(s)$ returns the set of states reachable from state s on reading some label, along with the associated costs. We only retrieve those edges for node n in G whose label corresponds to one of those returned by $NextStates(s)$, thereby using the transitions in the automaton to guide the selection of neighbouring nodes in G.

$NeighboursByEdge$ takes as input the *oid* of a node n from G and a *label*, and returns a list of neighbouring node *oids*. If the label is not '*', we invoke the Sparksee method `Neighbors` in order to retrieve all neighbouring nodes for n connected by an edge labelled with *label*, taking directionality into account. If the label is '*', we invoke `Neighbors` once for edges labelled 'edge' and once for edges labelled `type`. We do this for both directions in both cases.

In each case, we iterate over the neighbouring nodes, adding their *oid* to W (lines 7 and 8). As it is possible for $NextStates$ to return identical labels consecutively (at the same cost), we store the results of $NeighboursByEdge$ for new labels in a

Procedure Succ(s, n)

Input: state s of NFA and node n of G
Output: set of transitions from (s, n) in H_R

(1) $W \leftarrow \emptyset$; $U \leftarrow \emptyset$
(2) $currlabel \leftarrow$ null; $prevlabel \leftarrow$ null
(3) **foreach** $(label, successor, cost) \in NextStates(s)$ **do**
(4) $\quad currlabel \leftarrow label$
(5) $\quad$ **if** $currlabel \neq prevlabel$ **then**
(6) $\quad\quad U \leftarrow NeighboursByEdge(n, label)$
(7) $\quad$ **foreach** *node* $m \in U$ **do**
(8) $\quad\quad$ add the transition $\stackrel{cost}{\rightarrow} (successor, m)$ to W
(9) $\quad prevlabel \leftarrow currlabel$
(10) **return** W

Class hierarchy	Depth	Average fan-out
Episode	2	2.67
Subject	2	8
Occupation	4	4.08
Education Qualification Level	2	3.89
Industry Sector	1	21

Figure 2: Characteristics of the class hierarchies.

set U (line 6), avoiding identical calls to $NeighboursByEdge$.

4. PERFORMANCE STUDY

In this section, we present performance results from two case studies. We also discuss two optimisations, showing how each results in improved performance for some of the APPROX/RELAX queries. All experiments were run on an Intel Core i7-950 (3.07-3.65GHz) with 6GB memory, running Windows 7 (64 bit).

4.1 L4All Case Study

Our first case study uses data from the L4All project [17]. Briefly, the L4All system aimed to support lifelong learners in exploring learning opportunities and in planning and reflecting on their learning. The system allows users to create and maintain a chronological record — a timeline — of their learning and work episodes. Each episode is (i) linked to an Episode category by an edge labelled `type`, (ii) linked to other episodes edges labelled 'next' or 'prereq' (indicating whether the earlier episode simply preceded, or was necessary in order to be able to proceed to, the later episode), and (iii) linked to either an occupational or an educational event, by means of an edge labelled 'job' or 'qualif', which in turn is classified in terms of Education Qualification Level or Industry Sector, respectively.

Figure 2 shows the class hierarchies used in the ontology accompanying the data; the *depth* is the length of the longest path from the root to the leaf nodes, and the *average fan-out* is the average number of children of each non-leaf class. There is only one property hierarchy: the super-property 'isEpisodeLink' has 'next' and 'prereq' as subproperties. These properties also have defined domains and ranges, but as these are not used in the our performance study, we do not discuss them further.

Our initial data comprised five detailed timelines from real users, to which we added 16 additional realistic timelines. Each of these timelines consisted of a mixture of educational and occupational episodes, and varied in terms of the number of episodes contained within them, as well as the classification of each episode.

	L1	L2	L3	L4
Nodes	2,691	15,188	68,544	240,519
Edges	19,856	118,088	558,972	1,861,959

Figure 3: Characteristics of the L4All data graphs.

Q1	(Work Episode, type$^-$, ?X)
Q2	(Information Systems, type$^-$.qualif$^-$, ?X)
Q3	(Software Professionals, type$^-$.job$^-$, ?X)
Q4	(?X, job.type, ?Y)
Q5	(?X, next+, ?Y)
Q6	(?X, prereq+, ?Y)
Q7	(?X ,next+|(prereq+.next), ?Y)
Q8	(Mathematical and Computer Sciences, type.prereq+, ?X)
Q9	(Alumni 4 Episode 1_1, prereq*.next+.prereq, ?X)
Q10	(Librarians, type$^-$, ?X)
Q11	(Librarians, type$^-$.job$^-$.next, ?X)
Q12	(BTEC Introductory Diploma, level$^-$.qualif$^-$.prereq, ?X)

Figure 4: The L4All query set.

We then scaled this data graph up by creating synthetic versions of the real timelines in order to obtain four data graphs of increasing size, called **L1** (143), **L2** (1,201), **L3** (5,221) and **L4** (11,416), where the number in brackets refers to the number of timelines. Figure 3 shows the characteristics of each data graph. The synthetic timelines were generated by duplicating a real timeline and using the ontology to alter the classification of each episode to be a 'sibling' class of its original class, for as many sibling classes as are present. Each duplicated timeline remained identical to the original in terms of the number of episodes, whether the type of the episode was educational or occupational, and the manner in which episodes were linked to each other. Thus, as the data graph increases in size, the degree of the class nodes (i.e. the nodes with incoming `type` edges) increases linearly. As the data graph size increases, the total number of edges also increases linearly with the number of nodes.

We execute a series of single-conjunct queries on this data in order to evaluate the performance of our APPROX and RELAX operators, shown in Figure 4. These 12 queries include actual queries used in the original L4All case study, as well as others designed to stress test our implementation. Each query is first run in 'exact' mode — i.e. neither APPROX nor RELAX is used — followed by versions of the same query containing either the APPROX or the RELAX operator. We therefore run 36 queries in total.

We used a cost of 1 for each approximation operation (insertion, substitution and deletion). For RELAX, we applied rules of type (i) (see Section 2), also at a cost of 1. We ran each query five times, discarding the first run as the cache-warm-up. After initialisation, each exact query was run to completion, in which *all* results are obtained. On the other hand, each APPROX and RELAX run comprises the following sequence: initialisation; obtain results 1–10 ('batch 1'); obtain results 11–20 ('batch 2'); ...; obtain results 91–100 (batch 10). For exact queries, the average time to return all answers was taken across runs 2 to 5. For APPROX and RELAX queries, we took the average of each of the 10 batches across runs 2 to 5 to obtain an average for each batch. We

	Q3	Q8	Q9	Q10	Q11	Q12
L1: Exact	58	0	1	1	2	0
L1: APPROX	100 **1** *(42)*	100 **2** *(100)*	100 **1** *(32)* **2** *(67)*	100 **1** *(7)* **2** *(92)*	100 **1** *(12)* **2** *(86)*	100 **1** *(100)*
L1: RELAX	100 **1** *(42)*	0	12 **1** *(11)*	100 **1** *(20)* **2** *(20)* **3** *(59)*	100 **1** *(40)* **2** *(40)* **3** *(18)*	59 **1** *(59)*
L2: Exact	1,090	0	1	1	2	0
L2: APPROX	100	100 **2** *(100)*	100 **1** *(32)* **2** *(67)*	100 **1** *(7)* **2** *(92)*	100 **1** *(12)* **2** *(86)*	100 **1** *(100)*
L2: RELAX	100	0	12 **1** *(11)*	100 **1** *(20)* **2** *(20)* **3** *(59)*	100 **1** *(40)* **2** *(40)* **3** *(18)*	59 **1** *(59)*
L3: Exact	3,104	0	1	1,024	2,048	0
L3: APPROX	100	100 **2** *(100)*	100 **1** *(32)* **2** *(67)*	100	100	100 **1** *(100)*
L3: RELAX	100	0	12 **1** *(11)*	100	100	59 **1** *(59)*
L4: Exact	3,104	0	1	1,024	2,048	0
L4: APPROX	100	100 **2** *(100)*	100 **1** *(32)* **2** *(67)*	100	100	100 **1** *(100)*
L4: RELAX	100	0	12 **1** *(11)*	100	100	100 **1** *(100)*

Figure 5: Results for each query and data graph.

then computed the average over all batches. Some of these queries yielded fewer than 100 results.

We show the number of results obtained for queries 3, 8, 9, 10, 11 and 12 per data graph in Figure 5. Queries 1 and 2 showed similar performance to Query 3, while queries 4–7 all returned well over 100 exact results on all the data graphs, thus negating the need to apply APPROX and RELAX to them for the purposes of this performance study. For APPROX and RELAX queries yielding non-exact answers, we also show in Figure 5 the distances of the non-exact answers, as well as the number of the answers at each non-zero distance in brackets (with the number of exact answers comprising the difference). For example, query Q9/APPROX on data graph **L2** returns 1 exact answer (100-(32+67)), 32 answers at distance 1 and 67 answers at distance 2.

Figures 6, 7 and 8 show the average execution times for the exact, APPROX and RELAX versions, respectively, of queries 3, 8, 9, 10, 11 and 12 over the data graphs **L1**–**L4**.

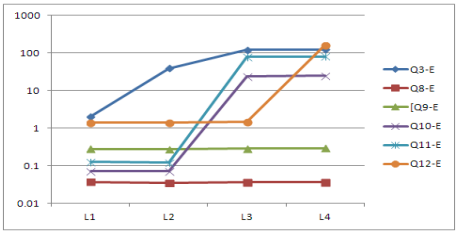


Figure 6: Execution time (ms) – exact queries.

For the exact queries, we see that queries 8 and 9 take constant time for all the data graphs since at most a single answer is returned. The jump in execution time from **L2** to **L3** for queries 10 and 11 is caused by the large increase in the number of answers; similarly for query 3. Query 12 shows a steep increase owing to the manner in which the synthetic timelines were generated, giving rise to the processing of nodes of ever-increasing degree. We note that the performance of all the queries is competitive with the expected behaviour of native NFA-based approaches to regular path query evaluation [11].

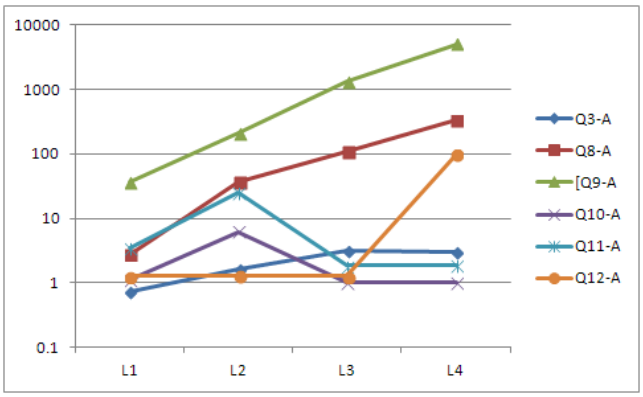


Figure 7: Execution time (ms) – APPROX queries.

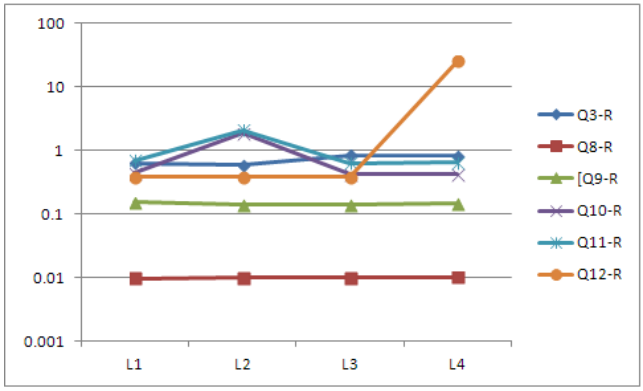


Figure 8: Execution time (ms) – RELAX queries.

For the APPROX queries, queries 10 and 11 show a decrease in the time taken for **L3** and **L4** compared with **L2** which is caused by the fast processing of sufficient exact results for the larger two data graphs; similarly for query 3. However, the APPROX versions of queries 8, 9 and 12 exhibit an exponential increase in time taken to retrieve the top 100 results. This is caused by a large number of intermediate results being generated (due to the *Succ* function returning a large number of transitions which are then converted into tuples in $GetNext$ and added to D_R). We discuss optimisation of query 9 in Section 4.3. Regarding queries 8 and 12, the time for query 8 decreased from 332ms to 272ms by applying the first optimisation of Section 4.3. Query 12 was not aided by the optimisations of Section 4.3.

The RELAX queries 3, 8, 9, 10 and 11 all exhibit a fairly constant execution time across the data graphs. Query 12 shows an increase from **L3** to **L4** for a reason similar to that for its APPROX version.

4.2 YAGO Case Study

For our second case study, we used data from YAGO [10]. The connectivity patterns in YAGO differ from the rather 'linear' timelines comprising the L4All data, so provide a contrasting basis on which to evaluate query performance. Additionally, the YAGO ontology differs from the L4All one in terms of its breadth and depth.

We downloaded the simpler taxonomy and core data facts from the YAGO website (the SIMPLETAX and CORE por-

Q1	(Halle_Saxony-Anhalt, bornIn$^-$.marriedTo.hasChild, ?X)
Q2	(Li_Peng, hasChild.gradFrom.gradFrom$^-$.hasWonPrize, ?X)
Q3	(wordnet_ziggurat, type$^-$.locatedIn$^-$, ?X)
Q4	(?X, directed.married.married+.playsFor, ?Y)
Q5	(?X, isConnectedTo.wasBornIn, ?Y)
Q6	(?X, imports.exports$^-$, ?Y)
Q7	(wordnet_city, type$^-$.happenedIn$^-$.participatedIn$^-$, ?X)
Q8	(Annie Haslam, type.type$^-$.actedIn, ?X)
Q9	(UK, (livesIn$^-$.hasCurrency)|(locatedIn$^-$.gradFrom), ?X)

Figure 9: The YAGO query set.

	Q2	Q3	Q4	Q5	Q9
Exact	2	0	0	0	0
APPROX	100 **1** *(98)*	100 **1** *(5)* **2** *(95)*	?	?	100 **1** *(100)*
RELAX	2	100 **1** *(100)*	0	100 **1** *(100)*	100 **1** *(100)*

Figure 10: Query results for the YAGO data graph.

tions) and imported these into our system[2]. The resulting data graph consists of 3,110,056 nodes and 17,043,938 edges. There is only one classification hierarchy in YAGO; its *depth* is 2 and *average fan-out* is 933.43.

Including the `type` property, YAGO uses 38 properties. There are two property hierarchies, containing 2 and 6 sub-properties respectively. The properties also have domains and ranges defined, not used in our performance study.

The queries we ran on the YAGO data are listed in Figure 9. The exact, APPROX and RELAX versions therefore give rise to 27 queries, for which we calculated the timings as described in Section 4.1, with the edit and relaxation costs the same as those used for the L4All case study.

The number of results obtained for queries 2, 3, 4, 5 and 9 for the YAGO data graph are shown in Figure 10. For each query, the exact version was run to completion, and the APPROX and RELAX versions were run until the top 100 answers were retrieved. The '?' indicates instances where the system ran out of memory and hence failed without returning any answers. Query 1 showed a similar performance to query 2; query 6 is similar to queries 4 and 5 in terms of query structure, but it terminated, unlike these; and queries 7 and 8 returned well over 100 exact results, therefore negating the need for APPROX and RELAX.

Figure 11 shows the average execution times for queries 2, 3, 4, 5 and 9. For the exact queries, queries 2 and 3 execute quickly. Queries 4 and 5 take longer to execute because their conjuncts are of the form $(?X, R, ?Y)$. Hence processing is initiated from a large number of nodes (41,811 and 33,834 respectively), and further traversal leads to large numbers of intermediate results; query 9 behaves similarly.

APPROX queries 2 and 3 exhibit poor performance due to a large number of intermediate results, while query 9 takes the same time as the exact version; we discuss these further in Section 4.3. Queries 4 and 5 failed to terminate as the system ran out of memory; this, too, is due to a large number of intermediate results.

RELAX queries 2, 3 and 9 performed competitively, returning more results for the latter two than their exact counterparts. Query 4's time was the same as for the exact version (with no extra results). Query 5 returned results and executed faster than the exact version; this is due to 100 results being found (by the application of rules of type i) and execution terminating sooner.

[2]http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/downloads/

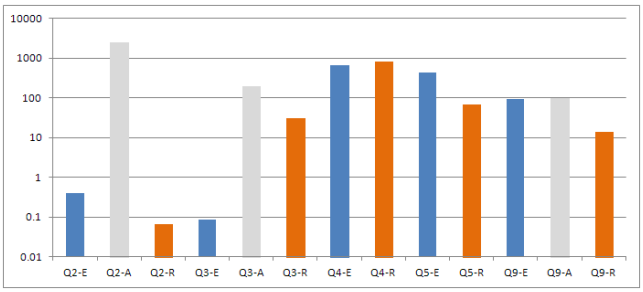


Figure 11: Execution times (ms), YAGO data graph.

4.3 Query execution optimisations

In this section we outline two optimisations which may improve the performance of APPROX and RELAX queries.

Retrieving answers by distance: We have implemented a distance-aware mode of query execution for APPROX and RELAX queries in order to prevent the unnecessary processing of data which yields answers at a cost higher than that required by the user. For example, if the user requests the top 100 answers in cases where there are over 100 answers at cost 0, using transitions of greater cost to traverse G and add tuples to D_R results in a slower query execution time overall, owing to the processing of redundant data.

We set a *current maximum cost*, ψ, to be 0 initially. No tuple having a cost greater than zero is processed (i.e. added or removed from D_R), and all answers of cost 0 are returned. Should more answers be required, we then increment ψ by the smallest cost, ϕ, of the edit or relaxation operations being applied. For each successive value of ψ, query evaluation commences from the beginning, as all tuples having a cost $c \leq \psi$ need to be considered (so this method is not suitable in cases where answers at high cost are required) but no tuple having a cost greater than ψ is processed.

Using distance-aware retrieval substantially improves the performance of some APPROX queries. For example, L4All queries 3 and 9 run three to four times faster with this optimisation. YAGO query 3 executes twice as fast, while query 2 takes 0.6ms instead of 2560ms, a dramatic improvement.

Replacing alternation by disjunction: Another optimisation for APPROX queries we have implemented is to decompose the NFA for a regular expression $R = R_1|R_2|...$ into sub-automata NFA_i for each R_i, providing the NFA has a single start state. These are processed in default order (NFA_1, NFA_2, ...) for the distance 0 answers, and we store the number of answers returned in each case, $n_{0,i}$. To compute the answers at distance ϕ, we evaluate the sub-automata by increasing $n_{0,i}$ value. In general, to compute the answers at distance $k\phi$, we evaluate the sub-automata by increasing $n_{(k-1)\phi,i}$ value.

For example, applying this to YAGO query 9, results in the sub-automata NFA_1 for `(UK, livesIn`$^-$`.hasCurrency, ?X)` and NFA_2 for `(UK, locatedIn`$^-$`.gradFrom, ?X)`. NFA_1 the returns the least answers at distance 0, so this is processed first for the distance 1 answers. This reduces the query execution time to 12.65ms compared with 101.23ms.

5. RELATED WORK

In this section we briefly review previous work on the implementation of regular path queries.

[11] presents a technique for the evaluation of exact queries which takes advantage of *rare* labels in a graph. A query containing one or more rare labels is broken down into a set of sub-queries such that each sub-query begins or ends with a rare label. Their method, using a bi-directional search utilising graph indexes, is shown to be faster than other automaton-based implementations. Our exact queries perform favourably compared with the results in [11].

[22] presents RPL, a regular path language for RDF data, whose implementation, like ours, uses an automaton-based approach. However, RPL is only able to process very small graphs efficiently [11].

[4] describes a framework allowing weighted RDF data to be queried in a cost-aware manner, and returning results ranked according to cost. This is accomplished by an extension to SPARQL, SPARankQL, encompassing the provision of novel predicates for expressing flexible paths between nodes and the capacity to define ranked queries (in which the weights are used). Our work allows the path to be expressed by a regular expression which may be mutated by edit operations, whereas SPARankQL can only be used to express either no restrictions on paths from a node or restrictions on specified labels of the path. The data graphs used in their performance study have, respectively, 9K nodes (24K edges) and 10K nodes (25K edges), and are both smaller and more sparse than our **L2** graph.

[6] discusses a SPARQL query graph model using transformation rules to rewrite queries. Experiments are run on RDF graphs of increasing size, with the largest comprising 1,272K triples. The rewritten queries run approximately twice as fast as the original ones. We do not yet make use of query rewriting, which is an area of future work.

6. CONCLUSIONS

Building on previous work on combining approximation and relaxation for regular path queries [18, 17], we have provided a detailed description of our implementation, *Omega*, focussing on low-level data structures and physical optimisations, both in terms of the interaction with the graph store, *Sparksee*, and our query processing layer within *Omega*.

We have presented a comprehensive performance study, using large graphs consisting of real-world data, in which we show that our baseline implementation performs competitively in terms of exact regular path queries. The benefits of our APPROX and RELAX operators have been shown in terms of additional answers being returned for queries returning few or no answers for the exact version. Many of the APPROX and RELAX queries executed quickly, but some either failed to terminate or did not complete within a reasonable amount of time. We discussed the reasons for this in each case, and showed how further optimisations, such as retrieval by distance and replacing alternation by disjunction, enabled several queries to execute faster.

For future work, we will consider the use of disk-based data structures to guarantee the termination of APPROX queries with large intermediate results (such as YAGO queries 4 and 5). We will also investigate using characteristics of the data graph and heuristics to reduce the amount of unnecessary processing. Other promising directions are query rewriting, and leveraging rare labels as in [11]. Distributed approaches [8, 19] are also relevant for flexible querying of larger-scale graphs than we have considered in our centralised approach so far.

7. REFERENCES

[1] Bio4j. http://bio4j.com/.

[2] C. Bizer, A. Jentzsch, and R. Cyganiak. http://lod-cloud.net/state/.

[3] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185, 2000.

[4] J. P. Cedeño and K. S. Candan. R2DF framework for ranked path queries over weighted RDF graphs. In *Proc. WIMS*, pages 40:1–40:12, 2011.

[5] M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer, 2009.

[6] O. Hartig and R. Heese. The SPARQL query graph model for query optimization. In *Proc. ESWC*, pages 564–578, 2007.

[7] T. Heath, M. Hausenblas, C. Bizer, and R. Cyganiak. How to publish linked data on the web (tutorial). In *Proc. ISWC*, 2008.

[8] J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL querying of large RDF graphs. *PVLDB*, 4(11):1123–1134, 2011.

[9] C. A. Hurtado, A. Poulovassilis, and P. T. Wood. Ranking approximate answers to semantic web queries. In *Proc. ESWC*, pages 263–277, 2009.

[10] G. Kasneci, M. Ramanath, F. Suchanek, and G. Weikum. The YAGO-NAGA approach to knowledge discovery. *SIGMOD Rec.*, 37(4):41–47, Mar. 2009.

[11] A. Koschmieder and U. Leser. Regular path queries on large graphs. In *Proc. SSDBM*, pages 177–194, 2012.

[12] N. Martinez-Bazan and D. Dominguez-Sal. Using semijoin programs to solve traversal queries in graph databases. In *Proc. GRADES*, pages 6:1–6:6, 2014.

[13] N. Martínez-Bazan et al. Efficient graph management based on bitmap indices. In *Proc. IDEAS*, pages 110–119, 2012.

[14] Neo4j. http://www.neo4j.com/.

[15] I. U. of Copenhagen. http://www.itu.dk/research/c5/.

[16] OrientDB. http://www.orientdb.org/.

[17] A. Poulovassilis, P. Selmer, and P. T. Wood. Flexible querying of lifelong learner metadata. *IEEE Trans. on Learning Technologies*, 5(2):117–129, 2012.

[18] A. Poulovassilis and P. T. Wood. Combining approximation and relaxation in semantic web path queries. In *Proc. ISWC*, pages 631–646, 2010.

[19] M. Przyjaciel-Zablocki, A. Schätzle, T. Hornung, and G. Lausen. RDFPath: Path query processing on large RDF graphs with MapReduce. In *ESWC Workshops*, 2011.

[20] Reco4j. http://www.reco4j.org/.

[21] Sparksee. http://www.sparsity-technologies.com/.

[22] H. Zauner, B. Linse, T. Furche, and F. Bry. A RPL through RDF: expressive navigation in RDF graphs. In *Proc. RR*, pages 251–257, 2010.

[23] D. D. Zhu and K. I. Ko. *Problem Solving in Automata, Languages, and Complexity*. Wiley, Newark, NJ, 2004.

Beta-algebra: Towards a Relational Algebra for Graph Analysis

Luiz Gomes-Jr*† gomesjr@ic.unicamp.br
Bernd Amann† bernd.amann@lip6.fr
André Santanchè* santanche@ic.unicamp.br

*Institute of Computing
State University of Campinas (UNICAMP)
Campinas – SP – Brazil

†LIP6 (UMR 7606/CNRS)
Université Pierre et Marie Curie (UPMC)
Paris – France

ABSTRACT

Graph analysis is an essential tool to understand natural and man-made networks, such as social networks, food webs, transportation infrastructures, etc. Although graph analysis has fomented the development of algorithms, visual tools, and distributed processing frameworks, there is still little support for analysis at the query language level. Current graph query languages are mostly concerned with flexible matching of subgraphs, while graph processing frameworks are mostly concerned with fast parallel execution of instructions.

Our goal is to provide analysis capabilities at the language level, allowing more interactive and explorative query-based analysis. In this paper, we present our ongoing efforts towards a relational algebra extension that offers an operator for graph-based data aggregation. The beta (β) operator is composed of four suboperators, which are used to control the path-based aggregations. The β-algebra allows seamless composition of queries that mix relational and graph-based aspects.

Here we introduce our current algebra and provide examples of its use. We also show how we are using the analysis strategy in query scenarios. Since the algebra-based query scenario allows for execution plan rewritings, we also discuss our first efforts on equivalence rules for query optimization.

Keywords

Graph algebra, relational algebra, Complex Networks, graph data models, graph query languages

1. INTRODUCTION

Graph analysis has become an important tool in diverse fields. Social, transportation, communication, and biological networks are examples of information often organized as graphs, which require specific tools and algorithms for proper data analysis.

The area of Complex Network Analysis [6] has advanced in the last decades and has produced several models, algorithms and techniques to study natural and human-made networks. In terms of database support, there has been a strong acceleration in the usage of graph databases and query languages, as well as in the development of the underlying algorithms and mechanisms. There is, however, still a big gap between graph query languages and the analysis techniques. Current graph query languages offer little support for the type of analysis required for complex networks. Such a gap is not present, for example, in traditional relational databases, which support query languages that offer aggregation operations that are the basis of more sophisticated multidimensional analysis.

Our goal is contributing towards bridging this gap. We aim at developing data management and querying mechanisms that offer a better support for network analysis. In this paper, we present our first steps towards creating an algebra that offers a graph-based aggregation operation. We expect this algebra to be the basis of more expressive queries, supporting declarative and interactive graph data exploration.

Our proposed algebra is based on Codd's relational algebra [4]. This has several advantages: (i) it provides a well established theoretical basis; (ii) it allows the combination of traditional relational operations alongside our proposed graph operation; (iii) it is a de facto standard in database research. Having an underlying algebra allows a better understanding of the semantics of the query language and, most importantly, allows the definition of rewriting rules for execution plan optimization. As an extra benefit, relational algebra compatibility also simplifies implementation in current relational database systems.

In simple terms, our goal with the algebra is to provide graph-based aggregation of values. The core of our proposal is the beta (β) operator, which encapsulates the graph traversal procedure and allows parametric control over the aggregation of values. We see graph-based aggregation as a generalization of relational aggregation over sets. Considering that most useful relational aggregations perform joins before applying an aggregation operation, we adopt this pattern of first deriving relationships between the data (joins or graph traversals) and then aggregating the values as the basis for our new constructor.

Combining the advantages of relational query languages and

graph analysis, the proposed algebra allows the construction of queries involving subgoals such as: obtain a subgraph based on given nodes properties and edge labels; calculate the reputation of the nodes in the subgraph; combine the reputation and the average distance to a given reference node in the general graph; order the resulting nodes based on the final score. In our envisioned scenario, such queries would be starting points for deeper explorative analysis, with goals such as: analyze the average node degree for the top-k nodes returned in the query; obtain the average value for a certain attribute for the bottom-k scoring nodes, etc.

This paper is organized as follows: Section 2 describes related work and fundamental concepts. Section 3 presents the definition of our algebra alongside with query and execution examples. Section 4 briefly describes our current approach for querying and initial query rewriting rules for execution plan optimization. Finally, Section 5 concludes the paper.

2. RELATED WORK

There is great diversity of graph query languages, which have pushed the boundaries for more expressive constructors [16]. Graph query languages are often based on conjunctive regular path queries (CRPQs). CRPQs are the basis for several graph languages, such as GraphLog [5] and SPARQL[1]. Recent developments have extended CRPQs in order to allow constraints over path properties. These types of queries have been described as extended conjunctive regular path queries (ECRPQs) [2]. ECRPQs also allow paths to be returned as query results. These queries are all focused on data selection and support only the simplest cases of analysis.

Query languages such as in GID [15] allow ranking based on pre-calculated metrics of importance which capture the dynamics of a snapshot of the network. Our goal is to enable a higher level of on-demand analysis in graph query languages. To that extent, we are working towards an algebra that can handle graphs and aggregations over paths. The goal is to use this algebra to build more flexible query languages.

Several algebras that support graphs have been proposed. To our knowledge, the algebra that is closer to our goals is the alpha-algebra [1], which also serves as inspiration for the name of our operator. The alpha operator derives the transitive closure for tables that express self-relationships – e.g. CONNECTS(from, to, distance). The algebra supports aggregation and filtering over paths through the delta (Δ) attribute, which is an internal relation containing each path history in the result set. Conceptually, the alpha operator has two main characteristics that make it unsuited for our needs: the operator always processes until reaching fix point, and there is a single point for value aggregation. In our algebra, we add more flexible stop conditions and split aggregation in four suboperations (Section 3). We also change the underlying model and add several elements for querying convenience. The changes allow more analysis algorithms to be represented in the algebra as well as providing more opportunities for optimization based on query rewritings (Section 4).

[1]http://www.w3.org/TR/sparql11-query

Frasincar et al. [8] propose an algebra for RDF, with some operators inspired by relational algebra. The algebra enables both querying and construction of RDF models. Although the algebra shares many of the goals in this paper, the focus is on the complete RDF-S model, which incurs considerable complexity when compared to our simpler graph model. Most importantly, the proposed algebra does not support graph-based aggregation, our main focus.

In a more recent and simpler proposal, Cyganiak [7] defines a relational algebra for the SPARQL language over the RDF model. The authors rely on a global reference table containing RDF triples (subject, property, object) as basis for the operations. We use a similar strategy for our underlying model as we employ global tables for nodes and relationships to represent the graph in the database. Their proposal, however, does not deal with aggregation or analysis issues.

The demand for graph analysis in large scale has motivated the development of several frameworks for distributed computation: Google's Pregel, the first NoSQL implementation in that scale, was followed by diverse proposals including GraphLab [13] and GraphX[2]. These models focus on parallelization of the API operations and do not provide declarative languages as means for data querying. GraphX shares some of our motivation since it aims at simplifying mixed analysis that include graphs and relations. The focus is, however, on general parallel computation and not on querying. Our goal is to provide a higher level, declarative language for graph querying and analysis, allowing a more interactive and explorative interface with the user. Although we are not concerned with distributed computation at the moment, we believe that would be a natural evolution for our framework.

The Complex Networks [6] field is a prominent area that would benefit from query-base graph analysis. Complex network formation is based on localized phenomena, which in a global scale determines emergent behavior that cannot be assessed based merely on the analysis of parts of the system. The researchers employ a variety of models and algorithms to derive knowledge from the structures. Among the frequently used algorithms are the well known PageRank [3] and HITS. Most of the analysis in the field is done using *ad hoc* applications with no database support.

The algebra that we propose in this paper has been developed in the context of our Complex Data Management System (CDMS) [11], which aims at providing a database-like framework for complex network analysis and management. Appropriate query languages (and underlying algebras) would allow a more fine-grained, exploratory and interactive interaction with the networks.

3. THE BETA-ALGEBRA

In this section we describe the requirements for our algebra and present the beta operator alongside example queries.

3.1 Requirements

The basic requirements for the proposed algebra are:

[2]https://spark.apache.org/graphx/

Allow traditional relational aggregation: given the widespread use and familiarity with relational database queries, it is important to build on and leverage this foundation. Moreover, a graph analysis workflow often contains routines that are typically relational (counting, ranking, statistics, etc).

Enable aggregation over path traversals: Most graph analysis tasks involve aggregating values along graph traversals. It is important to allow flexible and case-specific aggregation functions that are applied as the graph is traversed.

Preserve the closure of relational algebra: It is important that the aggregation operates over and produces relations. This allows compatibility with relational algebra properties studied for decades, as well as making the language more flexible and composable.

Support flexible selection of nodes of interest: To enable explorative analysis over graph data it is important to have efficient means to filter nodes and relationships that are to be part of the analysis. A declarative language would naturally allow this type of flexibility.

Offer means to express subgraphs to constrain the analysis: Currently, most complex networks analysis is done over graphs as a whole. We believe this is highly associated with a lack of convenient means to select a subgraph of interest and apply the analysis over the selection. An appropriate algebra needs to enable graph-based constraints over the execution of the algorithms.

Rewriting rules for cost-based optimization: One of the main advantages of building query languages over an underlying algebra is the possibility of rewriting the queries for faster execution. Including graph-specific operations allows the specification of semantically sound rewriting rules for the graph setting.

Other requirements, not covered in this paper but that we want to address soon are:

Include convergence criteria for operation termination: This would allow the implementation of complete versions of popular graph analysis algorithms (e.g. PageRank).

Provenance/Path information for query results: Like in the alpha algebra with its delta attribute, adding information about paths traversed by the operations allows more flexibility for query composition and allows optimizations such as avoiding loops in graphs.

3.2 Data model

In this paper we use a simple interpretation of graphs in the relational model. In this model, a labeled property multigraph[3] [14] G is represented in two relational tables, V (vertices or nodes) and E (edges or links). The tables contain attributes representing the properties for all nodes and edges (typically highly sparse tables). The V table is in the form $\langle node, a_1, a_2...a_n \rangle$, where *node* represents the node id in the database, and $a_1, a_2...a_n$ are node properties.

[3]a graph with labeled links and properties for nodes and links

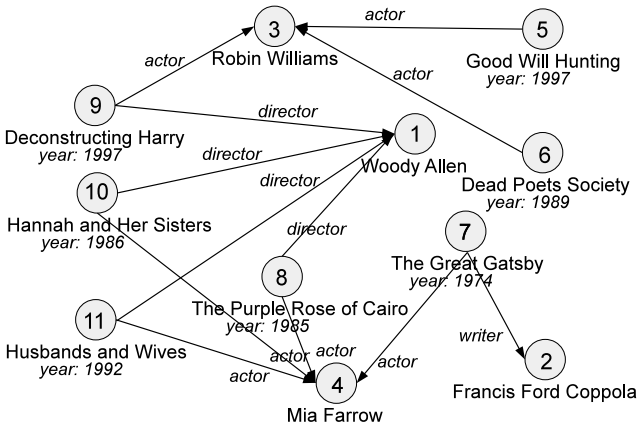


Figure 1: Subgraph from a movies dataset

The E table is in the form $\langle source, dest, label, a_1, a_2...a_n \rangle$. *source* and *dest* are ids representing the connected nodes for a given edge (the order implies the direction), *label* is the label according to the labeled property graph model, and the properties are as in the V table.

This data model is the reference for the proposed algebra. However, as with any other model, it does not impose implementation constrains (e.g. it can be implemented over a pure graph database and use efficient structures to represent the sparse tables). Our own implementation stack does not include any traditional relational component.

In the following paragraphs, the beta operator will be presented informally, in increasingly complexity as parameters and suboperations are introduced. Since the operator can be seen from either a graph or relational perspective, we will use equivalent terms interchangeably (e.g. join and traversal step). The semantics, however, is always relational.

3.3 The beta operator

Much like the alpha operator, the beta operator assesses recursive relations in the database. However, the main goal is not to derive transitive closure. Instead, the focus is on data aggregation along the traversal of the relations. In its simplest form, the beta operator performs a single join between a single column source table and the table E. An union operation is then applied to aggregate the original and new nodes. For the sake of readability, we omit extra join, projection, and renaming operations required to maintain the original schema after the execution of the operator. In a graph interpretation, the beta operator augments an initial set of nodes with all of their neighbors. For example, based on the graph in Figure 1, the beta operator applied to a source table containing one column with node ids $\{9, 10\}$ would produce $\{9, 10, 3, 1, 4\}$. This operation is represented as $\beta(\sigma_{id \in \{9,10\}}(V))$.

In general, we represent the beta operator as $n\beta_p(R)$, with $p = \langle s, dir, set, map, reduce, update, C \rangle$. Several parameters are used to control the behavior of the beta operator. s is the join condition, which accepts Boolean expressions just like its relational counterpart. n determines the number of recursive calls to the operator (i.e. consecutive joins).

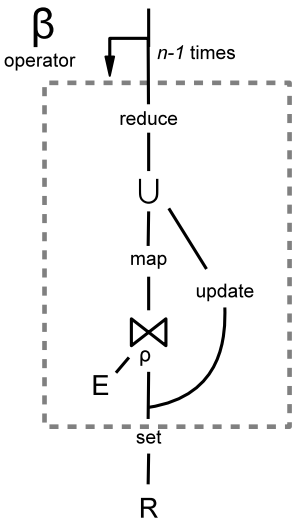


Figure 2: Simplified execution tree template

$dir \in \{inbound, outbound, both\}$ determines the order of the relations in the joins (or the direction of the graph traversal operations). The optional parameter *source* determines, for tables with more than one node column, the column over which the beta operator operates (the default is the fist column).

The operator keeps the algebra closed since (i) it always produces a table with at least the same columns as the input table, and (ii) it can be defined using standard relational algebra with aggregation. The design choices (such as encapsulating the table E inside the beta operator) are for convenience and to focus on what we think are the most important aspects of an aggregation operation. This aspect is inspired by the introduction of the relation join that despite being a redundant operation has directed the focus of the research on properties and optimizations of a central element of the model.

The most important elements of the beta operator are the aggregation suboperations. To allow full control of the computation as the graph is traversed, we define four operations: *set*, *map*, *reduce*, and *update*. *set* is a function that attributes a value to a new column before the join (traversal) operation is performed. *map* calculates a new value based on each node in the source relation. The new values are associated with the neighbors after the join operation. *reduce* is a function that aggregates over values for the same source node (equivalent to a group by). Finally, *update* redefines the aggregation values for the source nodes before the union. Figure 2 shows a simplified execution tree for the beta operator. As an example, the query:

$\sigma_{type=movie}(5\beta(\sigma_{id=1}(V)))$,
with {set: dist=0, map: dist=dist+1,dir=both}, obtains distances to movie nodes that can be reached from the initial node 1 (director Woody Allen) in up to 5 steps. Figure 3 shows the partial tables after the suboperations for the first iteration of the query. The query:

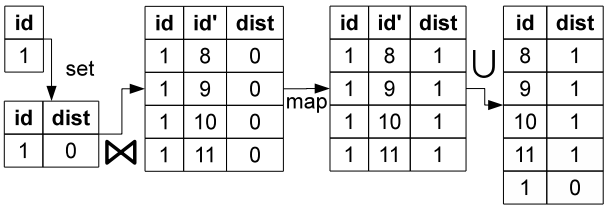


Figure 3: Snapshots of the resulting tables inside the beta operator for the first iteration of the example query.

$\sigma_{type=movie}(5\beta(\sigma_{id=1}(V)))$,
with {set: dist=0, map: dist=dist+1, reduce: dist=MIN(dist)} obtains the minimum distances in the same setting.

Another parameter is the optional graph constructor C. Its purpose is to specify the effective subgraph for the beta operator, constraining the search space for the traversals. The reference points for the constructor are the input nodes. C has its own parameters: *radius* is the maximum distance from the reference points; s is the edge selection expression (similar to the join condition in the beta operator). Nodes and links beyond the radius or that do not match the selection are ignored by the beta operator. The addition of the graph constructor in the algebra is also for convenience sake. The same effect could be obtained by a series of joins and selections over the E table. The query:

$5\beta(\sigma_{type=movie}(V))$,
with {set: rank=1, map: rank=rank/|e.out()|,
reduce: SUM(rank), reset: rank = 0, C: {radius:3}}, is a simplified PageRank algorithm executed for five iterations (not until convergence) over a graph of radius 3 around the source nodes in R. $|e.out()|$ represents the number outbound nodes (that can be obtained with traditional algebra aggregation). If the number of source nodes is known and its inverse is used in set function, the query obtains, for each node in the constructed graph from C, the probability that a random walker would stop at the node after five steps.

Other parameters and functionalities that we want to investigate are (i) specifying stop conditions for the beta operator, including simple test and convergence properties, (ii) recording path traversal information in a delta attribute, with functions that operate over it (similar to the alpha-algebra), and (iii) a modifier equivalent to the SQL *distinct*, that uses the delta attribute to avoid the computation of cycles.

4. QUERYING AND OPTIMIZATION

In this section we present our initial attempts with query language design and rule-based optimization.

4.1 Querying

We are developing the algebra presented here to support the query language that we have been developing as part of our CDMS system. The language that we are currently using is an extension of popular graph queries as shown in Figure 4. The language shows the type of queries we are envisioning, although it is less expressive than the algebra that we are

```
SELECT DISTINCT ?actor WHERE {
  ?film movie:director director:1 .
  ?film movie:actor ?actor .
  ?film movie:initial_release_date ?date .
  FILTER ( fn:starts-with(?date, "199") ) }
RANK BY RELEVANCE OF ?actor TO director:1
```

a

```
START diag=node(*)
   WHERE has(diag.Type) and diag.Type = "Diagnose"
   RETURN diag
RANK BY
   %r RELEVANCE OF diag TO node( %p ) WEIGHTED,
   %c CONNECTIVITY OF diag TO node( %p ) WEIGHTED
```

b

Figure 4: Query examples. a) extends SPARQL and b) extends Cypher

proposing. We plan to design a more expressive language following the definition of our algebra.

In the initial language, the graph-based aggregation is expressed in a RANK BY clause. The clause accepts metrics that aggregate values over graph traversals as in the algebra presented. For example, Relevance is a generalization of the notion of relevance in Information Retrieval, attributing higher scores to elements that have multiple and more specific connections (paths). This metric can be represented in our algebra by a beta operator with aggregation functions $\{set : score = 1, map : score = (score * 0.9)/e.out(), reduce := SUM(score)\}$. Details about the metrics and queries can be found in [10].

Figure 4a shows a query that retrieves actors whose careers are strongly correlated (relevant) with the director Woody Allen (id 1). Based on the subgraph in Figure 1, Mia Farrow would have a much higher score than Robin Williams. Another interesting query, that includes traditional relational aggregation, would be to find the pair (actor, director) with the maximum mutual relevance. This type of query would be hard to express using current graph or relational queries.

Figure 4b shows a query that we used for a nursing diagnosis task. Possible diagnosis are ranked based on their connections with the symptoms identified in the patients. This query contains a combination of two different metrics (Relevance and Connectivity).

4.2 Rewriting rules

An important motivation for introducing a new algebra is to better understand the computation complexity and define rewriting rules for query optimization. This work is still ongoing and we will only show some first examples for illustration.

The first rule is about the bidirectionality of the analysis. A beta operation that starts on a group of nodes and selects another group of target nodes can be reversed (changing the directions of the allowed edges). Reversing the direction can, in certain cases, reduce the search space by avoiding dense regions of the graph. For example, a three step undirected traversal from node 1 to node 6 in Figure 1 visits 9 nodes, while the traversal from 6 to 1 activates only 4 nodes. This rule can be represented as:

$\sigma_a(\beta(\sigma_b(R))) \equiv \sigma_b(\overline{\beta(\sigma_a(R))})$, where $\overline{\beta}$ represents the β operator with inverted directions (parameter dir). We are omitting, for sake of simplicity, extra joins and renamings that would make the outmost selections equivalent. We have tested this strategy for the query in Figure 4a against a comprehensive movie database [12]. The execution was reduced to half the time of the baseline query. The initial results, using a different formalization, were published as a technical report [9]. We still have to assess the subset of aggregation functions that allow the use of this rule. In practice, this rule requires cost-based planning, which we have not implemented yet.

Another rule, regarding compositionability of operators, combines aggregation functions that are applied over the same data by different beta operators. It can be represented as:

$\sigma_a(\beta_p(\sigma_b(R))) \bowtie \sigma_a(\beta'_{p'}(\sigma_b(R))) \equiv \sigma_a(\beta_{p \bullet p'}(\sigma_b(R)))$, where p represents the tuple of aggregation functions for the operator and $p \bullet p'$ combines the respective functions. The functions in p and p' must not make conflicting operations over the same attributes. We expect this type of rewriting to be very common, as multiple metrics can be used for the same target nodes (as in the query b in Figure 4). We have not yet implemented this rule in the system.

We have also explored options for speeding up queries beyond rule-based rewritings. We have tested, with positive results, caching paths between nodes for metrics that need to traverse the paths in both directions. We also explored a few query approximation strategies for specific metrics. These experiments are also reported in [9]. Another possibility is to materialize graphs constructed from the parameter C for use in multiple beta operators in the same query (rewriting the execution tree to take advantage of the materialized graph).

5. CONCLUSION

Graph analysis has become an important requirement in a wide range of modern applications and research fields. This type of analysis is currently highly specialized, employing ad-hoc applications or complex distributed frameworks. Graph databases offer little support for graph-based aggregations that would allow for query-based analysis.

Here we presented our ongoing work on the beta-algebra, which is intended to allow graph-based aggregations for declarative query languages. The algebra extends the relational model to support graph traversals and allows the control of several aspects of the aggregations.

We have shown examples of the use of the algebra and how it fits in our broader goal of developing data management and querying mechanisms specific for graph/complex network analysis. Also, we presented initial tests and directions for query optimization based on execution plan rewriting, along with our preliminary experimental results.

Our algebra allows more expressive querying when comparing to pure relation aggregation and CRPQs. The increased expressiveness enables explorative analysis and more interactive data manipulation. It also enables seamless integration of relational and graph-based analysis, which is a com-

mon application scenario. We believe the algebra is a good basis to build expressive query languages as well as useful optimization strategies.

Ongoing and future work include the expansion of the algebra to allow more flexible stop conditions (e.g. convergence), accumulation of traversal history (such as with the delta attribute in alpha-algebra), definition and tests of rewriting rules, specification of a query language that can take full advantage of the algebra, and implementation of a distributed query processor.

Acknowledgments

This work was partially financed by the Microsoft Research FAPESP Virtual Institute (NavScales project), CNPq (MuZOO Project and PRONEX-FAPESP), INCT in Web Science (CNPq 557.128/2009-9) and CAPES-COFECUB, with individual grants from CAPES and FAPESP (Proc. 2012/15988-9 and 2014/01419-8). The opinions and conclusions expressed in this work do not necessarily reflect the funding agencies'.

6. REFERENCES

[1] R. Agrawal. Alpha: An extension of relational algebra to express a class of recursive queries. *IEEE Trans. on Softw. Eng.*, 14(7):879, July 1988.

[2] P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst*, 37(4):31, 2012.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[4] E. F. Codd. A relational model of data for large shared data banks. *Comm. ACM*, 13(6):377–387, June 1970.

[5] M. Consens and A. O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *In Proceedings of the Ninth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 404–416, 1990.

[6] L. Costa, O. Oliveira Jr, G. Travieso, F. Rodrigues, P. Boas, L. Antiqueira, M. Viana, and L. Rocha. Analyzing and modeling real-world phenomena with complex networks: A survey of applications. *Advances in Physics*, 60:329–412, 2011.

[7] R. Cyganiak. A relational algebra for SPARQL. Technical Report HPL-2005-170, Hewlett Packard Laboratories, May 21 2005.

[8] F. Frasincar, G.-J. Houben, R. Vdovjak, and P. Barna. RAL: An algebra for querying RDF. *World Wide Web*, 7(1):83–109, 2004.

[9] L. Gomes-Jr, L. Costa, and A. Santanchè. Querying complex data. Technical Report IC-13-27, Institute of Computing, University of Campinas, October 2013.

[10] L. Gomes-Jr, R. Jensen, and A. Santanchè. Towards query model integration: topology-aware, ir-inspired metrics for declarative graph querying. In *GraphQ-EDBT*, 2013.

[11] L. Gomes-Jr and A. Santanchè. The Web Within: leveraging Web standards and graph analysis to enable application-level integration of institutional data. *to appear in Transactions on Large Scale Data and Knowledge Centered Systems*, 2014.

[12] O. Hassanzadeh and M. Consens. Linked movie data base. In *Proceedings of the 2nd Workshop on Linked Data on the Web (LDOW2009)*, 2009.

[13] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5:716–727, 2012.

[14] M. A. Rodriguez and P. Neubauer. The graph traversal pattern. *CoRR*, abs/1004.1001, 2010.

[15] R. Varadarajan, V. Hristidis, L. Raschid, M.-E. Vidal, L. D. Ibáñez, and H. Rodríguez-Drumond. Flexible and efficient querying and ranking on hyperlinked data sources. In *EDBT*, pages 553–564, 2009.

[16] P. T. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, 2012.

Graph Search of Software Models Using Multidimensional Scaling

Bojana Bislimovska[1], Güneş Aluç[2], M. Tamer Özsu[2] and Piero Fraternali[1]

[1] *Politecnico di Milano,* [2] *University of Waterloo*

{bojana.bislimovska, piero.fraternali}@polimi.it {galuc,tamer.ozsu}@uwaterloo.ca

ABSTRACT

Software models formalize the requirements, structure and behavior of a system or application. They represent essential artifacts that simplify the process of software development. Software repositories have been developed to store models in order to facilitate the reuse of know-how from software projects; however, methods for searching these model repositories are not very efficient. Specifically, while being more scalable, general-purpose keyword search is not suitable for model search because it does not consider the structure that is inherent in software models: a good search algorithm should consider the model structure as well as the knowledge concentrated in the metamodel. On the other hand, existing approaches that consider the structure while querying software models are limited to only specific domains such as Business Process Models (BPMs).

In this paper, we introduce MultiModGraph, an efficient approach for indexing and searching model repositories. MultiModGraph preserves the model structure and metamodel information by representing models as graphs. To enable efficient search, the approach employs multidimensional scaling to approximately map vertices of the model graph to points in space. We evaluate MultiModGraph both with respect to speed and quality of results using a real-word repository of web application models.

1. INTRODUCTION

Models facilitate software development in multiple ways: They raise the level of abstraction to help deal with the increasing complexity in software development; they help organizations improve source code quality and adapt faster to changes in the requirements of a project; and they improve communications within an organization. Models have a specific structure, which is expressed using a well-defined syntax of a modeling language. Each modeling language conforms to a metamodel, which defines the structure, semantics and constraints for building a model [10].

Model repositories are used for storing collections of software models. Most (model) repositories offer elementary tools to search these collections of models, which is particularly important for model re-usability [12]. For example, instead of designing a model from scratch, developers can retrieve an already existing modeling pattern (from the repository) and tailor it according to their needs to build new software models. This can significantly improve the model development process by decreasing its time and cost, while at the same time improving its quality.

Broadly speaking, model repositories employ two techniques for search: general-purpose keyword search [13], or content-based search that incorporates the model structure in the query [16, 19]. However, each technique has its shortcomings. While being more scalable, general-purpose keyword search is not suitable for model search because it does not consider the structure that is inherent in software models [2]. Furthermore, most keyword-based approaches allow only exact matching of keywords, where a set of keywords is matched against the models' description (e.g. model element labels). On the other hand, those approaches that consider the structure while querying software models are limited to only specific domains such as Business Process Models (BPMs). In contrast, methods are needed that are (i) more general, (ii) sensitive to the knowledge about the model structure and (iii) are at the same time scalable. Specifically, these solutions should allow users to pose queries (to the model repository) in the form of a model sketch, which captures the intended requirements in a native modeling language supported by the model repository. Then, the repository should rank and return a sub-collection of the "most relevant" modeling patterns from these models.

In this paper, we propose an algorithm for efficient search of Web Modeling Language (WebML[1]) models, namely, MultiModGraph. The algorithm uses a representation of models as attributed graphs, which allows mapping of the model structure and hierarchies among the model elements to a graph. Queries, which represent model fragments, can also be transformed into graphs, and they can be used for searching similar models in a model repository. Our algorithm uses multidimensional scaling to represent the graph vertices as points in a multidimensional space. These points are used to build an index that allows for efficient pruning during search. This way, given a query vertex, those vertices in the graph that are relevant to the query can be located efficiently (i.e., they correspond to points within a specified distance in the

[1]WebML is a modeling language for Web application frontends, recently generalized into the OMG IFML standard (www.ifml.org)

multidimensional space). Furthermore, the algorithm considers neighborhood information for each graph vertex in order to locally expand already matched vertices. Metamodel information is incorporated in the search and indexing, as well as in the ranking function which sorts retrieved models with respect to their similarity.

The paper is organized as follows: Section 2 presents related work; Section 3 describes the WebML modeling language and the process of model to graph transformation; Section 4 gives the system architecture; Section 5 describes the proposed approach; Section 6 illustrates our results, and finally, Section 7 concludes and proposes directions for future work.

2. RELATED WORK

Existing works can be classified into 3 areas: (i) keyword-based model search, (ii) content-based model search, which present specific techniques for search of models, and (iii) graph databases whose indexing and querying approaches can be employed for model search.

2.1 Keyword-Based Model Search

Keyword-based approaches for model search use a set of keywords for querying models. Their main limitation is that they do not consider the model structure in the query or the hierarchies and relationships among model elements. Furthermore, they return exact but not approximate matches to the query, which may be relavant to the user.

Moogle [13] is a keyword-based model search engine that uses metamodels to create indexes for the evaluation of keyword queries. In comparison to our approach, Moogle supports only textual queries with just a simple filter on the type of the model element to be returned. Another keyword based search solution for WebML models is presented in [2]. It incorporates metamodel information in the search process, used only in deciding how weights are assigned to different index terms. In contrast, MultiModGraph supports relationships among model elements through graph model representation, as well as some additional metamodel information such as references to the Data Model.

2.2 Graph-Based Model Search

Existing approaches that rely on a graph-based representation of models predominantly target Business Process Models (BPM) and their corresponding notations. However, BPMs are not as rich as WebML models in terms of syntax and semantics. Moreover, they are not suitable for searching large collections of models, since they only rely on a scan of the set of models without any indexes. One example of such technique is [6], which proposes discovering and ranking of BPEL process models. This is achieved by using behavioral similarity measure and a graph matching algorithm.

The approach in [16] retrieves process models by combining related pairs' clustering and a set of metrics for comparison of vertex labels. The main limitation of this approach is that the similarity between two process models is mainly based on the similarity of vertex labels rather than the structural similarity of the model graphs.

Some recent approaches [19, 8] exploit indexing for more efficient retrieval of business process models. These indexes are mostly feature-based, containing subgraphs that are most representative features of the model graphs in the repository. However, this type of indexing cannot be applied to models with complex metamodels, such as WebML.

In [2], we compare keyword-based approach with a graph-based approach for searching web application(WebML) models, but we do not apply any indexing in the graph-based approach.

2.3 Graph Databases

The approaches for indexing and querying that allow efficient search of large graph databases can be employed for efficient search of models. NeMa (Network Match) [9] is a neighborhood-based top-k subgraph matching technique that uses a minimal cost function to evaluate a goodness of a match. It considers structure and label similarities and it uses a neighborhood-based vector index to improve efficiency. Unlike NeMa, MultiModGraph uses different kind of indexing based on multidimensional scaling. TALE (Tool for Approximate Subgraph Matching of Large Queries Efficiently) [18] is a general tool for approximate subgraph matching. It employs neighborhood-based indexing. TALE allows for vertex mismatches and vertex and edge gaps. The basic differences are that in MultiModGraph the queries are small model fragments, and the graphs are attributed.

There also exist some approaches for search of attributed graph databases [11, 20] whose graphs contain multiple labels for both vertices and edges. Attributed graphs are used to represent WebML models, because they exploit the richness of the WebML metamodel. These techniques for matching attributed graphs use indexing methods that contain neighborhood information for each vertex. MultiModGraph also uses neighborhood information, but for a different purpose, i.e., to expand the matching candidates. The main limitation of these approaches is that they do not consider approximate, but only exact graph matching.

3. BACKGROUND AND PRELIMINARIES

In this section, we give a brief introduction to WebML, and describe the transformation from WebML models to attributed graphs.

3.1 Web Modeling Language (WebML)

WebML is a Domain Specific Language (DSL) for designing complex web sites [4], recently generalized into OMG IFML standard. It consists of two parts (i) data model and (ii) web model. The data model describes the data requirements of an application, using entity-relationship notation. The web model describes the organization of the front-end interfaces of a web application. It contains three main building blocks, namely pages, units and links which are organized hierarchically into larger container elements such as areas and site views. A *site view* represents a model element that includes a well-defined set of requirements for a specific category of users. Site views can contain *areas*, container elements that cluster pages with a homogeneous subject and can be nested recursively [3]. Pages are the actual interface elements delivered to the user and they contain *content units* which represent atomic elements for specifying the content of a web page. Another type of units is an *operation unit*, contained in the areas and site views. Operation units denote operations on data or arbitrary business actions; they can be activated as a result of a link navigation, performing manipulation with data, or execution of an external service. Content and operation units are connected by links. Links allow sequencing of units, passing parame-

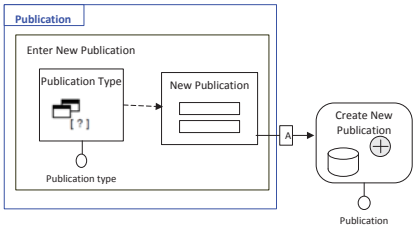


Figure 1: Example of a WebML model

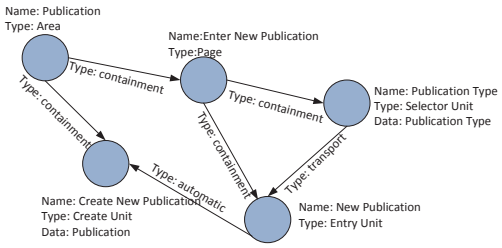


Figure 2: Graph representation of the WebML model in Figure 1

ters, navigating the hypertext front-end, changing the page content or accessing a page.

Each WebML model has a structure determined by the WebML metamodel, and an inherent hierarchy determined by the container WebML model elements. Figure 1 shows an example of a part of a WebML model. The area *Publication* contains a *Enter New Publication* page, which allows the user to insert a new publication through the *New Publication* entry unit. The selection of a publication type is enabled by the selector unit *Publication Type*. After the insertion of the data for a publication, a new publication instance is created, which is performed by the *Create New Publication* create unit.

3.2 Model to Graph Transformation

We represent WebML models as attributed graphs such that every model element is represented as a vertex in the graph, while containment relationships and links among the model elements are represented as graph edges. This type of (graph) representation preserves as much as possible the model structure and the hierarchies present among the model elements. Specifically in this case, each graph vertex is annotated with three attributes: (i) name, (ii) type and (iii) data, (Figure 2), where:

- *Name* represents the textual label of the model element;
- *Type* represents the corresponding model element type, derived fro mthe metamodel;
- *Data* represents the entity or relationship to which the model element refers, in case such reference exists.

Likewise, each edge is annotated with the attribute *type* which refers to the type of the corresponding relationship/link in the model, as represented in Figure 2.

Thus, after the transformation, each WebML model from the repository yields an attributed graph, and the model search becomes the problem of searching over a collection of attributed graphs. Since queries also represent model fragments, they can be transformed in the same way into graphs.

4. SYSTEM ARCHITECTURE

Figure 3 presents the architecture of our graph-based model search system. The *Content Processing* component takes every model from the repository and transforms it into a format suitable for indexing. First, the *Project Analysis* sub-component extracts general informaton from the model such as the model name and id. Then, the *Model to Graph Transformation* sub-component transforms each model into a graph considering its metamodel features, as explained in Section 3 for the case of WebML models. These model graphs are used to build the index in the *Indexing* component, which is elaborated in more detail in Section 5.1.2.

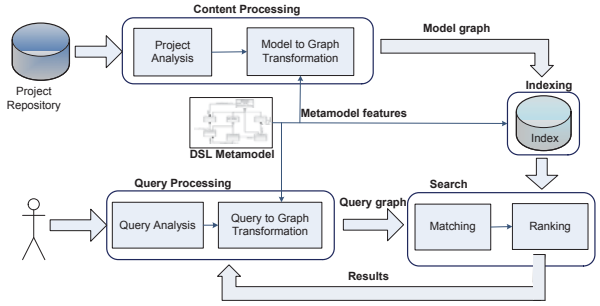


Figure 3: Architecture of a graph-based model-driven information retrieval system.

On the user side, a query is expressed as a model fragment, which can be formulated in the same modeling language in which models in the repository are encoded. The *Query Processing* component transforms the query model fragment into a format that is more suitable for search, the same way models are transformed into graphs by the *Content Processing* component. When the query is transformed into a graph, the system is ready for search.

The *Search* component has two tasks, which are discussed in more detail in Section 5.1.3. The *Matching* sub-component uses a specific algorithm and the help of the index to find model fragments (subgraphs) from the repository that match the query graph under certain criteria. Finally, the *Ranking* sub-component performs sorting on the retrieved model subgraphs with respect to their relevance to the query. These ranked subgraphs (along with their computed ranking scores) are returned to the user.

5. DETAILS OF MULTIMODGRAPH

Our main objective is, for a given query, to find a ranked list of modeling patterns in the repository such that the returned patterns are as similar as possible to the modeling pattern that represents the query. In our approach, the problem can be rephrased as discovering a ranked list of subgraphs in the set of project graphs similar to the query graph. We define the notion of similarity as follows: A query graph is similar to a retrieved project subgraph based on the similarity of the textual content represented by the labels of the vertices and edges in the attributed graphs, as well as the similarity in their corresponding graph topolo-

gies. Moreover, the size of these similar subgraphs should be comparable to the size of the query graph, so that upon retrieval, these subgraphs (or the modeling patterns they represent) can be reused to build new software models with as few modifications as possible. One may note that in one large project graph there might be multiple subgraphs similar to a query graph, since a given task can be presented with different modeling patterns, and we would like to capture also those kind of modeling patterns.

Our approach consists of an indexing phase, in which vertices of the model graphs are indexed for efficient search, and a search phase, in which (i) an index lookup is performed on the vertices to find potentially matching candidates, (ii) the matched vertices are expanded to form subgraph patterns that are similar to the query graph, and (iii) the subgraphs are ranked with respect to their similarity to the query graph.

In the indexing phase, illustrated in Figure 4, we build three types of indexes. The first index is a grid index, that uses multidimensional scaling to cluster similar graph vertices from all of the project graphs for each attribute that represents a different model feature: name, type and data (cf., Section 3.2).

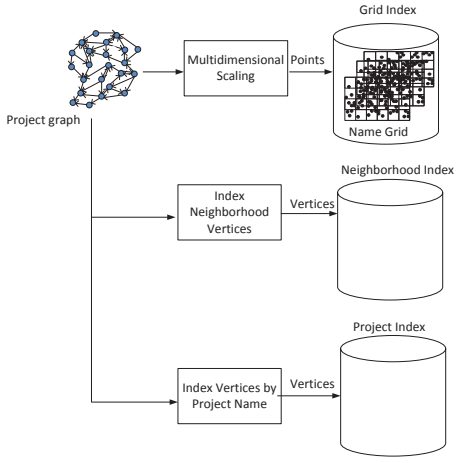


Figure 4: Indexing in MultiModGraph.

We use multidimensional scaling because our preliminary evaluations (detailed results are available in [1]) showed that it allows efficient pruning of most of those vertices that are beyond a distance threshold from a given query vertex. For expansion of the vertices matched using the grid index, two more auxiliary indexes are built: (i) a neighborhood index that considers the vertex neighborhood, and (ii) a project index that considers the vertices belonging to a graph (project), specified by their name. The indexing phase is discussed in more detail in Section 5.1.2.

In the search phase, shown in Figure 5, query vertices are also transformed into points in space through the same multidimensional scaling algorithm [5]. These points are used to search the grid index, retrieving only those vertices similar to the corresponding query vertices with respect to a specific attribute. Then, the project and the neighborhood index are searched to expand the matching candidates and form local subgraph matches, which are subsequently ranked considering a graph-edit distance metric as a similarity measure. Further details of search and match expansion can be found in Sections 5.1.3 and 5.1.4, respectively.

5.1 Graph search using multidimensional scaling

In this section, we present MultiModGraph in more detail. We illustrate the concept of graph similarity, i.e. how a query graph is defined to be similar to a subgraph of the project graph (Section 5.1.1); we describe the process of indexing the project graphs (Section 5.1.2), the search of similar query vertices to find matching candidate vertices (Section 5.1.3), the expansion of the matching candidate vertices to produce local subgraph matching patterns, and the ranking of the matching patterns with respect to the query (Section 5.1.4).

5.1.1 Graph Similarity

The similarity between the query graph and each of the project subgraphs is computed through graph-edit distance, a measure that specifies the number of graph-edit operations that transform one graph into the other. The considered operations are:

- *Vertex substitution*: a vertex in the project subgraph is substituted with a vertex in the query graph if they are similar. Two vertices are similar if the project graph vertex is retrieved as a result of searching the grid index for a specific query vertex considering at least one attribute.
- *Edge substitution*: an edge in the project subgraph is substituted with an edge in the query graph if their type labels belong to a similar type, and if their incident vertices are substituted. Two edge labels are similar if they are identical, or if they both belong to the set of WebML links, excluding the containment relationships.
- *Vertex deletion*: a vertex from the query graph that does not have corresponding similar vertices in the project subgraph is deleted from the query graph.
- *Vertex insertion*: a vertex from the project subgraph that does not have corresponding similar vertices in the query graph is inserted in the query graph.
- *Edge insertion*: an edge from the project subgraph that does not have corresponding similar edges in the query graph is inserted in the query graph.
- *Edge deletion*: an edge from the query graph that does not have corresponding similar edges in the project subgraph is deleted from the query graph.

5.1.2 Indexing

A. *Grid Index*

Given a set of data objects and the distance values between each pair of objects, multidimensional scaling assigns coordinates to each data object, such that distances computed from the assigned coordinates are as representative as possible to the actual distances. While this technique is used mainly in data visualization [15], we exploit this idea for clustering and then efficiently indexing the vertices of the model graphs.

We perform clustering of graph vertices with respect to the vertex attributes that correspond to the metamodel attributes in a model element. For our specific context, we consider the name, type and data attributes. Clustering

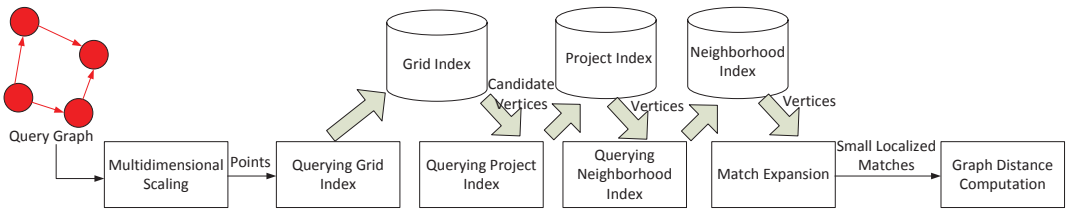


Figure 5: Search and Match Expansion in MultiModGraph.

is achieved by transforming vertex attributes' values representing a specific attribute class (name, type and data) as points in multidimensional space. The distance between points, preserved by the multidimensional scaling, is computed by the Euclidian distance. These computed distances help to find for a graph vertex, its "nearby" graph vertices with respect to a single attribute. The transformed points are placed into multidimensional grids, as shown in Figure 6. Each grid corresponds to a metamodel attribute, i.e., name, type and data. Therefore, the total number of grids is the same as the number of attributes (in our case three). The number of dimensions of each grid is equal to the number of dimensions of the points representing a specific attribute. These grids are used to build the grid index which allows for efficient pruning of all vertices that are not within a specified distance from a query vertex. In this work we chose the Chalmers algorithm [5] for performing multidimensional scaling, because it has lower computational overhead (quadratic) than other multidimensional scaling approaches without introducing too much noise. It is a heuristic-based approach, hence it does not provide tight error bounds. The quality evaluation of the algorithm is presented in [1].

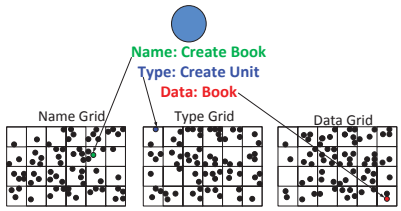


Figure 6: Grid Index in MultiModGraph.

B. *Auxiliary Indexes*

Besides the grid index, two other index structures are constructed and used in the search algorithm.

- The neighborhood index is an inverted index that keeps track of the neighborhood of each vertex, where for each vertex, all the vertices wthin its 2-hop neighborhood considering both ancestors and descendants are stored. This index considers the local structure around a graph vertex for expansion of already matched vertices. The 2-hop neighborhood has been selected for two reasons: (i) to better respond to the diversity of modeling patterns expressing a given task; (ii) to allow vertex mismatches between a query graph and a local subgraph match, since we perform approximate, and not exact matching. For scalability reasons, we do not exceed 2-hop neighborhood.
- The project index is an inverted index that for each vertex stores the corresponding project name. It is used to form subgraphs of vertices that belong to the same project.

5.1.3 Search

The search algorithm is described in *Algorithm 1* and presented in Figure 5. V_Q represents the set of vertices of the query graph, while *attribute* is the set of attribute types, namely, name, type and data attribute. The search process, takes each vertex v_q of the query graph and transforms it into a set of points in space $queryPoint$, using the Chalmers algorithm, where each point represents a specific attribute a. These points are used to query the grid index. Each query point is positioned in the corresponding grid in a similar way as the points that represent vertex label attributes from the project graphs are positioned by multidimensional scaling (using a single iteration of the Chalmers algorithm). In this way, the query point is "querying" the grid to find points *points* that are within an acceptable distance value *distance* (a user-specified parameter) with respect to a specific attribute a. Since the grids are independent, different acceptable distances can be assigned for each attribute (grid). This step performs pruning of points that are distant (dissimilar) from the query point, keeping only those points *points* that are within the specifed distance values[2].

When all the points for each attribute are retrieved, a union is performed across the different attribute classes, thus merging the results obtained from all the three grids into a set of candidates *candidates*. To further reduce the number of vertices, the candidate points are pruned by checking whether the real distance of the vertex labels they represent is within the acceptable distance values. As a real distance for name and data attribute we consider the string-edit distance, while for the type attribute we consider the distance between two types in the metamodel tree, normalized with respect to the longest distance in the tree. Finally, for each query vertex we obtain a set of real candidate vertices $realCandidates$ which represent the potential matches.

5.1.4 Match Expansion and Ranking

Each matching candidate is expanded in order to form small localized subgraph structures, denoted as matching patterns, achieved with the help of the project and the neighborhood indexes, as illustrated in Figure 5. The process of match expansion is described in Algorithm 2. At the beginning, for each query vertex v_q, a set of patterns is created, such that each pattern consists of one real candidate of v_q. Then, if query vertex v_q is not the first exam-

[2]Note that we will use the terms vertex and its representing point interchangeably.

Algorithm 1 Search Algorithm

```
Require: V_Q, attribute, distance
  for all v_q ∈ V_Q do
    candidates ← ∅
    for all a ∈ attribute do
      queryPoint ← ChalmersQuery(v_q, a)
      points[a] ← findCandidates(queryPoint,
        grid[a], distance[a])
      candidates ← candidates ∪ points[a]
    end for
    realCandidates[v_q] ← prune(v_q, candidates)
  end for
```

ined vertex, the algorithm checks whether its set of candidates *realCandidates* can extend already existing patterns. Namely, a project vertex from the candidates set *realCandidates* is added to an already created pattern in the set of existing patterns *patternSet*, if all of the following conditions are met:

- The project vertex represents a matching candidate for different query vertices with respect to the already matched query vertices.
- The project vertex belongs to the same project graph as the current matching pattern. This information is retrieved from the project index.
- The project vertex is within the 2-hop neighborhood of any of the project vertices present in the matching pattern. This information is retrieved from the neighborhood index.

As a result, a new set of patterns is created (*newPatternSet*), used to update the set of patterns *patternSet*.

The matching is complete when, for a matching subgraph pattern, all the query vertices have been examined for potential matches. Additional vertices from the project graph are added to the matching pattern if they are in the intersection *intersection* of the neighborhoods of the pattern's vertices (retrieved from the neighborhood index). In this way, all the vertices of a pattern are found. They are used to build a subgraph *subgraph* by finding the edges that connect these vertices from the project graph. Thus, *subgraph* represents a subgraph of the project graph. Once a matching subgraph is built, it is compared to the query graph with respect to the similarity, as explained in Section 5.1.1. Graph-edit distance is used as a similarity metric to rank the modeling patterns with respect to their similarity to the query. In the graph-edit distance computation, the subsituted vertices from a subgraph pattern are those that represent the real candidates, while the inserted vertices are the vertices that were added additionally to the pattern as a result of the intersection with the neighborhood index.

6. RESULTS

We evaluated our approach on a repository of WebML models[3] which consists of 12 real-word WebML projects from different application domains. The projects were divided into segments such that each segment represents a different WebML area in the project (i.e., areas group pages with similar purpose). This way, we obtained 341 segments in total.

The test queries were generated as follows. First, a set of exemplary models were selected by considering different

[3]Provided by the WebRatio company `www.webratio.com`

Algorithm 2 Match Expansion Algorithm

```
Require: V_Q, realCandidates
  patternSet ← ∅
  for all v_q ∈ V_Q do
    patternSet ← patternSet ∪ createPatterns(∅,
      realCandidates[v_q])
    if notFirstVertex(v_q) then
      for all pattern ∈ patternSet do
        if realCandidates[v_q] within two-hop neighborhood of
        pattern and realCandidates[v_q] in the same project as
        pattern then
          newPatternSet ←
            addToExistingPattern(pattern,
            realCandidates[v_q])
          patternSet ←
            updatePatternSet(patternSet, newPatternSet)
        end if
      end for
    end if
  end for
  for all pattern ∈ patternSet do
    for all v_p ∈ pattern do
      intersection ← neighborhood(v_p)∩
        neighborhood(pattern − ∑_{i=1}^{p−1} v_i)
      if intersection ≠ ∅ then
        pattern ← addAdditionalVertices(pattern,
          intersection)
      end if
    end for
    subgraph ← buildGraph(pattern)
  end for
```

WebML modeling patterns, a variety of metamodel concepts and a vocabulary of labels present in the repository. Then, three experienced model developers selected 10 models from the initial set of exemplary models that they believed better represented the typical user needs of a model developer [2]. Subsequently, these models were transformed into graphs (as explained in Section 3.2), which constitute the test queries in our evaluations.

To obtain the ground truth (used in our evaluations), we asked the same model developers to manually evaluate the relevance of each query against each project segment, where a relevance score of (i) 0 implies no relevance, (ii) 1 implies *either* textual or structural similarity, and (iii) 3 implies *both* textual and structural similarity. The final scores were computed by averaging the scores reported by the three domain experts, which was then rounded to the nearest integer [2]. Note that we did not use 2 as a score value to achieve greater diversity in the aggregate scores.

Given a query, MultiModGraph returns a set of modeling patterns (that it believes are relevant to the query). Hence, to assess the quality of the algorithm, we evaluate the relevance of each returned modeling pattern to the given query (based on the ground truth). However, note that a modeling pattern might span multiple project segments. Therefore, to assess a modeling pattern's relevance to a query, we consider all of the project segments that the modeling pattern spans. The final relevance of a modeling pattern is computed as an average of the relevance of the project segments.

In our evaluations, parameters of the Chalmers algorithm such as (i) number of iterations, (ii) max number of points in the random set and (iii) number of dimensions that are used for representing points in space, have been manually tuned to their optimal values [2]. As for the distance thresholds, we performed a preliminary evaluation to discard certain combinations of distance values across the three attribute classes: name, type and data. In Table 1, we show the acceptable distance values we use for each attribute class.

We have observed that using smaller distance values for the name and the type attribute classes does not retrieve sufficient number of candidate points. On the other hand, using greater distance values loosens the distance constraints, i.e., precision drops.

Table 1: Distance values configurations.

Name distance	Type distance	Data distance
0.4	0.4	0.2
0.4	0.4	0.4
0.6	0.4	0.2
0.6	0.4	0.4

As the baseline in our evaluations, we use the A-star algorithm [17] that we adapted for searching WebML models (details are in [2]). First, we compare MultiModGraph against the A-star algorithm with respect to their 11-point interpolated average precision, a metric that combines precision and recall by measuring the highest precision obtained at 11 standard levels of recall (ranging from 0.0 to 1.0) [14]. Namely, for each recall level i, the precision is computed as the maximum precision value for recall levels $j > i$, which is averaged across *all* of the test queries. In the computation of precision and recall values, we consider every modeling pattern with $relevance > 0$ relevant, while every modeling pattern with $relevance = 0$ irrelevant.

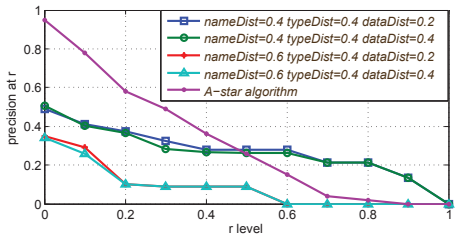


Figure 7: 11-point Interpolated Average Precision for different distance value configurations and the A-star algorithm.

Figure 7 shows the 11-point interpolated average precision for the four different configurations of MultiModGraph and the baseline algorithm. Each algorithm was configured to return the top 150 results. The values denoted as *nameDist*, *typeDist* and *dataDist* represent the acceptable distance values for the name, type and data attribute classes, respectively.

MultiModGraph achieves the best configuration with values of $nameDist = 0.4$ and $typeDist = 0.4$, while increasing *dataDist* from 0.2 to 0.4 does not affect precision/recall. Increasing the *nameDist* value further to 0.6, however, significantly worsens performance. Compared to the A-star (baseline) algorithm, MultiModGraph performs better for recall values greater than 0.5. This is important because it means that the algorithm still continues to retrieve relevant models at high levels of recall.

Figure 8 presents the best-case behaviour of MultiModGraph for configuration values $nameDist = 0.4$, $typeDist = 0.4$ and $dataDist = 0.4$, where queries with the best two 11-point interpolated curves are reported. The algorithm achieves a maximum precision of 1 even at lower levels of recall: up to 0.3 for Query 6, and up to 0.8 for Query 2. For other queries, the algorithm performs worse, which influences the overall algorithm performance when averaged across all of the queries.

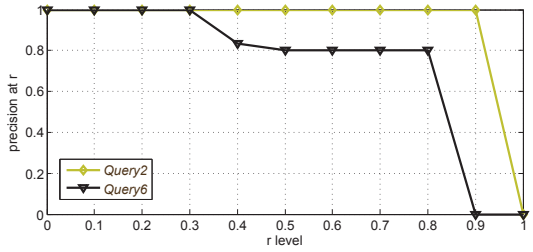
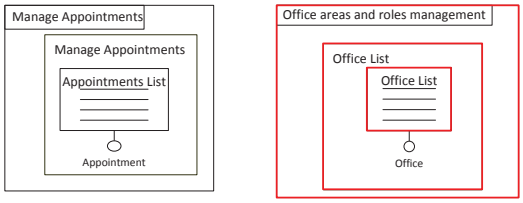


Figure 8: 11-point interpolated average precision for the best performing queries for $nameDist = 0.4$, $typeDist = 0.4$ $dataDist = 0.4$ configuration: Query 2 and Query 6.

MultiModGraph's lower performance in precision in some queries can be attributed to the way the algorithm selects candidate vertices. For a query vertex, a vertex in the project graph is considered a match candidate if at least one of the distances for a specific attribute class are within the specified distance thresholds. This generates patterns similar to the query, where each model element (in the result) is similar to a query model element with respect to a different attribute class. Some of these alternative matching patterns are still "reusable" (indeed, a closer manual inspection of the results further confirms this fact), but not all of them have been considered relevant by the ground truth.



(a)

Figure 9: Example of a query and its corresponding "irrelevant" result as deemed by the ground truth.

For Query 7, its highly ranked result by MultiModGraph is depicted in Figure 9, where each matched element in the result is highlighted by a red rectangle. Note the similarity between the query (on top) and its result (at the bottom). The query is about management of appointments and the result is about management of office areas and roles, but otherwise, they are structurally equivalent.

This highlights two possible future directions. First, ground truth generation can be improved to include relevance assessments at more fine-grained segments (i.e., currently, a project segment corresponds to a WebML area in the project). Second, improvements to the ranking function can be made to capture users' varying notions of relevance across different attribute classes (i.e., name, type and data attributes).

Lastly, we examine the runtime performance of MultiModGraph and compare it against the runtime performance of the baseline. We consider the average execution times over the entire query set. For this experiment, we varied the number of indexed vertices. The experiments were performed on a machine with Intel dual Core Processor 2.4 GHz, 6 GB RAM and Windows 7 (64-bit) operating system.

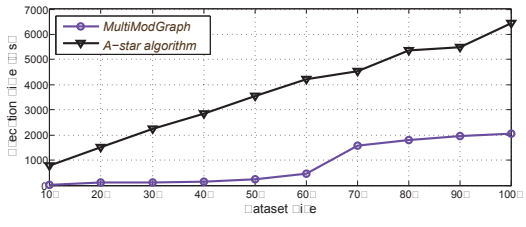


Figure 10: Runtime performance of MultiModGraph and A-star algorithm.

The runtime performance of the MultiModGraph is much better than the runtime performance of the A-star algorithm (on average, MultiModGraph is 12 times faster than the A-star algorithm), as demonstrated in Figure 10. The improvement in runtime performance is due to the use of indexing, however, it comes at a cost of some loss in the quality of the retrieved results, which is due to the approximate nature of the multidimensional scaling process. However, further optimizations are possible to improve both the quality and the runtime performance of MultiModGraph.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a graph-based approach for searching WebML repositories that uses multidimensional scaling. We have evaluated our approach on a real-world WebML repository using 10 test queries. Our preliminary results show that MultiModGraph has a better runtime performance than the baseline algorithm, but this comes at the cost of accuracy. We have argued that some of this inaccuracy could be attributed to the way the ground truth is generated. However, it may also be possible to improve performance by considering alternative objective functions for ranking, which is an integral part of our future work. Some other future work directions include: (i) applying MultiModGraph to different types of models by modifying the graph representation and the grid index according to the metamodel of the modeling language; (ii) testing the scalability of the approach on larger model collections; (iii) automatic tuning of the Chalmers algorithm parameters; (iv) performing efficiency comparison with existing indexing techniques for graph-edit distance (e.g. Closure tree [7]); and (v) tuning the search order in the search algorithm, by matching vertices with less candidates first.

8. REFERENCES

[1] B. Bislimovska. *Textual and content based search in software model repositories*. PhD thesis, Politecnico di Milano, 2014.

[2] B. Bislimovska, A. Bozzon, M. Brambilla, and P. Fraternali. Textual and content-based search in repositories of web application models. *ACM Transactions on the Web (TWEB)*, 8(2):11, 2014.

[3] A. Bongio, P. Fraternali, M. Brambilla, S. Comai, and M. Matera. *Morgan Kaufmann series in data management systems: Designing data-intensive Web applications*. Morgan Kaufmann, 2003.

[4] S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks*, 33(1-6):137–157, 2000.

[5] M. Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *Visualization'96. Proceedings.*, pages 127–131. IEEE, 1996.

[6] D. Grigori, J. C. Corrales, M. Bouzeghoub, and A. Gater. Ranking bpel processes for service discovery. *Services Computing, IEEE Transactions on*, 3(3):178–192, 2010.

[7] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 38–38. IEEE, 2006.

[8] T. Jin, J. Wang, M. La Rosa, A. Ter Hofstede, and L. Wen. Efficient querying of large process model repositories. *Computers in Industry*, 64(1):41–49, 2013.

[9] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. Nema: Fast graph search with label similarity. *Proceedings of the VLDB Endowment*, 6(3):181–192, 2013.

[10] A. G. Kleppe, J. B. Warmer, and W. Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[11] V. Krishna, N. Ranga Suri, and G. Athithan. Mugram: An approach for multi-labelled graph matching. In *International Conference on Recent Advances in Computing and Software Systems (RACSS), 2012*, pages 19–26. IEEE, 2012.

[12] M. La Rosa, H. A. Reijers, W. M. Van Der Aalst, R. M. Dijkman, J. Mendling, M. Dumas, and L. García-Bañuelos. Apromore: An advanced process model repository. *Expert Systems with Applications*, 38(6):7029–7040, 2011.

[13] D. Lucrédio, R. P. d. M. Fortes, and J. Whittle. Moogle: a metamodel-based model search engine. *Software & Systems Modeling*, 11(2):183–208, 2012.

[14] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[15] A. Morrison, G. Ross, and M. Chalmers. Fast multidimensional scaling through sampling, springs and interpolation. *Information Visualization*, 2(1):68–77, 2003.

[16] M. Niemann, M. Siebenhaar, S. Schulte, and R. Steinmetz. Comparison and retrieval of process models using related cluster pairs. *Computers in Industry*, 63(2):168–180, 2012.

[17] A. Sanfeliu and K.-S. Fu. A distance measure between attributed relational graphs for pattern recognition. *Systems, Man and Cybernetics, IEEE Transactions on*, (3):353–362, 1983.

[18] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. In *IEEE 24th International Conference on Data Engineering, 2008. ICDE 2008.*, pages 963–972. IEEE, 2008.

[19] Z. Yan, R. Dijkman, and P. Grefen. Fast business process similarity search. *Distributed and Parallel Databases*, 30(2):105–144, 2012.

[20] J. Yang, S. Zhang, and W. Jin. Delta: indexing and querying multi-labeled graphs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1765–1774. ACM, 2011.

Graph Data Exchange with Target Constraints

Iovka Boneva Angela Bonifati Radu Ciucanu
University of Lille & INRIA, France
{iovka.boneva, angela.bonifati, radu.ciucanu}@inria.fr

ABSTRACT

Data exchange is the problem of translating data structured under a source schema according to a target schema and a set of source-to-target constraints known as schema mappings. In this paper, we investigate the problem of data exchange in a heterogeneous setting, where the source is a relational database, the target is a graph database, and the schema mappings are defined across them. We study the classical problems considered in data exchange, namely the existence of solutions and query answering. We show that both problems are intractable in the presence of target constraints, already under significant restrictions.

1. INTRODUCTION

Data exchange is the problem of translating data structured under a source schema according to a target schema and a set of source-to-target constraints [11]. Such a problem has been studied in settings where both the source and target schemas belong to the same data model, in particular relational and nested relational [15, 11], XML [3], or graph [5]. Settings in which the source and the target schema are of heterogeneous data models have not been considered so far, apart from combinations of relational and nested relational schemas in schema mapping tools [15, 13].

In this paper, we focus on the problem of exchanging data between relational sources and graph-shaped target databases, which might occur in several interoperability scenarios in the Semantic Web, such as ontology-based data access [14] and direct mappings [16]. Motivations to map relational data to graphs abound, due to the far majority of data residing in relational databases and the need of integrating large amounts of linked data.

We express the relationships between the source and the target via *schema mappings* [15, 11, 7] i.e., logical assertions between two conjunctive queries, one on the source and the other on the target. Schema mappings between graph databases have already been introduced in [5] and we adopt their syntax for expressing the consequents of relational-to-graph schema mapping assertions. We point out that the setting without target constraints directly follows from the results in [5] on graph-to-graph data exchange. Furthermore, motivated by the fact that target constraints have been largely investigated within relational data exchange but so far disregarded for graph data exchange [5], we add them to our setting.

In particular, we focus on two fundamental problems of interest: *existence of solutions* (i.e., given a source schema and an instance of it, a target schema, a set of source-to-target constraints, and a set of target constraints, decide whether there exists an instance of the target schema satisfying all given constraints) and *query answering* (i.e., computing the answers that hold for all solutions).

Our *main contributions* are the following:

- We show that in the presence of *target equality-generating dependencies* [6], both existence of solutions and query answering are intractable (NP-hard and coNP-hard, respectively). This holds even under significant restrictions.
- We relax the notion of target constraints by introducing $sameAs$[1] target constraints, inspired by RDF[2]. We show that the existence of solutions becomes tractable while query answering is intractable (coNP-hard) for the same restrictions as for the previous case.
- We show that the notion of graph patterns [4], employed for graph data exchange [5] as *universal representatives* of all solutions, cannot be used as such when adding target constraints.

We point out that our hardness results stand in terms of *query complexity* since in the proofs we have used a fixed source schema and instance, while the target schema and the mappings are part of the input.

We also point out that none of our results is specific to the relational-to-graph setting, and hold in any setting where the target is a graph and the source is an arbitrary data model projecting on relational tuples. The source data can then be either XML, graph-shaped, or any other complex format as long as the left-hand sides of mappings extract relational tuples from it. Indeed, we shall pinpoint that the constraints on the target graph are solely responsible for the intractability results. Nevertheless, in the remainder of the paper, we focus on a relational data source for ease of presentation.

[1] `http://www.w3.org/wiki/WebSchemas/sameAs`
[2] `http://www.w3.org/RDF/`

Organization. In Section 2, we define our problem setting. In Section 3, we illustrate particular cases that can be solved using techniques from relational and graph data exchange. In Section 4, we characterize the complexity of the problems of interest. In Section 5, we present the challenges of defining and querying universal solutions. We discuss conclusions and future work in Section 6.

2. PROBLEM SETTING

Let us assume a countably infinite set of *constants* $\mathcal{V}$ that we use both as *domain* of relational databases and as *node identifiers* (or simply *node ids*) of graph databases.

Source schemas and queries. A *source schema* $\mathcal{R}$ is a finite collection of relational symbols. Each relational symbol has an *arity* that is a positive integer. An *instance* I of $\mathcal{R}$ is a function associating to each relational symbol R from $\mathcal{R}$ a set of tuples over $\mathcal{V}$ having the same arity as R. We abuse notation and use R to denote both relational symbol and its instance. A *source query* is a conjunction of atoms over $\mathcal{R}$ that uses only variables.

Target schemas and queries. A *target schema* Σ is a finite alphabet. An *instance* over Σ is a *directed, edge-labeled graph* $G = (V, E)$, where $V \subseteq \mathcal{V}$ is a finite set of node ids and $E \subseteq V \times \Sigma \times V$ is a finite set of edges. A *nested regular expression* (*NRE*) is an expression of the following grammar:

$$r := \varepsilon \mid a \ (a \in \Sigma) \mid a^- \ (a \in \Sigma) \mid r + r \mid r \cdot r \mid r^* \mid [r],$$

where "+" stands for disjunction, "$\cdot$" for concatenation, "$*$" for Kleene star, "$^-$" for traversing edges backwards, and "[ ]" for nesting. An NRE r defines a binary relation over graph nodes: $[\![r]\!]_G$ is the set of pairs of nodes in G s.t. there exists a path defined by r between the two nodes. We consider the semantics of NREs as in [5]. A *target query* is a *conjunction of nested regular expressions* (*CNRE*) using variables only. We illustrate CNREs in Example 2.2.

Schema mappings. A schema mapping is a set of *source-to-target tuple-generating dependencies* [6] (or simply *s-t tgds*) i.e., a set of formulas of the form

$$\forall \overline{x}.\ (\phi_{\mathcal{R}}(\overline{x}) \rightarrow \exists \overline{y}.\ \psi_{\Sigma}(\overline{x}, \overline{y})),$$

where $\phi_{\mathcal{R}}(\overline{x})$ is query over $\mathcal{R}$ and $\psi_{\Sigma}(\overline{x}, \overline{y})$ is a query over Σ. By $\overline{x}$ and $\overline{y}$ we denote vectors of variables. Moreover, all variables in $\overline{x}$ appear free in $\phi_{\mathcal{R}}(\overline{x})$, all variables in $\overline{y}$ appear free in $\psi_{\Sigma}(\overline{x}, \overline{y})$, and the variables in $\overline{x}$ that appear in $\psi_{\Sigma}(\overline{x}, \overline{y})$ are free.

Target constraints. We consider two well-known types of target constraints:

- *target equality-generating dependencies* [6] (or simply *egds*) i.e., $\forall \overline{x}.\ (\psi_{\Sigma}(\overline{x}) \rightarrow (x_1 = x_2))$,
- *target tuple-generating dependencies* [6] (or simply *target tgds*) i.e., $\forall \overline{x}.\ (\phi_{\Sigma}(\overline{x}) \rightarrow \exists \overline{y}.\ \psi_{\Sigma}(\overline{x}, \overline{y}))$.

In the aforementioned definitions, $\phi_{\Sigma}(\overline{x})$ and $\psi_{\Sigma}(\overline{x})$ are CNREs over Σ, and x_1 and x_2 are among the variables in $\overline{x}$. Moreover, we introduce *sameAs target constraints* that are a special case of target tgds i.e.,

$$\forall \overline{x}.\ (\psi_{\Sigma}(\overline{x}) \rightarrow (x_1, sameAs, x_2)).$$

In the sequel, we omit w.l.o.g. the universal quantifiers in front of a formula.

Definition 2.1 *A* (relational-to-graph) data exchange setting $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$ *consists of a relational source schema* $\mathcal{R}$*, a graph target schema* Σ*, a set* $\mathcal{M}_{st}$ *of s-t tgds, and a set* $\mathcal{M}_t$ *of target constraints.*

Solutions. Given a setting $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$, an instance I of $\mathcal{R}$ and a graph database G over Σ, we say that G is a *solution* for I under Ω if (I, G) satisfies $\mathcal{M}_{st}$ and G satisfies $\mathcal{M}_t$. We denote the set of all solutions by $Sol_{\Omega}(I)$. Usually, in relational data exchange, one aims at finding the universal solutions, from which there exist homomorphisms to all solutions [11]. This notion has been redefined for graph data exchange as universal representatives captured with graph patterns [5] that we discuss in detail in Section 3.2.

Example 2.2 Take a source schema $\mathcal{R}$ consisting of two relations: *Flight* storing information about flights that may have intermediate stops between the source and destination cities, and *Hotel* storing information about the hotels in which the passengers of such flights have stopped. Moreover, take the following instance I:

Flight		
flight_id	*src*	*dest*
01	c_1	c_2
02	c_3	c_2

Hotel	
flight_id	*hotel_id*
01	hx
01	hy
02	hx

Take a target schema consisting of the alphabet $\Sigma = \{f, h\}$. The edges labeled by f indicate a direct flight between two cities while the edges labeled by h indicate that a city has a hotel. Moreover, consider the following s-t tgd that basically requires that for each hotel where the passengers of a flight have stopped, there exists a city where the respective hotel is situated, and there exist flights from *src* to such city and from such city to *dest*:

$$\mathcal{M}_{st}:\ \mathit{Flight}(x_1, x_2, x_3) \wedge \mathit{Hotel}(x_1, x_4) \rightarrow \exists y.\ (x_2, (f \cdot f^*), y) \wedge (y, h, x_4) \wedge (y, (f \cdot f^*), x_3).$$

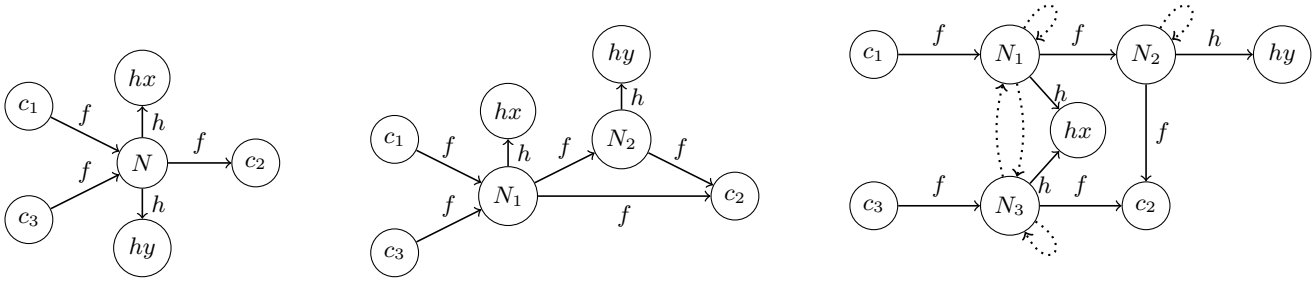
Notice that $\mathcal{M}_{st}$ uses a CNRE on its right hand side. Then, a natural constraint is that a hotel is situated in exactly one city, which can be captured either by the egd $\mathcal{M}_t$ or by the *sameAs* constraint $\mathcal{M}'_t$:

$$\mathcal{M}_t:\ (x_1, h, x_3) \wedge (x_2, h, x_3) \rightarrow (x_1 = x_2),$$
$$\mathcal{M}'_t:\ (x_1, h, x_3) \wedge (x_2, h, x_3) \rightarrow (x_1, sameAs, x_2).$$

The two ways of expressing the aforementioned target constraint yield two different settings $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$ and $\Omega' = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}'_t)$, respectively. We illustrate in Figure 1 solutions for I under these two settings: the graphs G_1 and G_2 are solutions under Ω, while G_3 is a solution under Ω'. ▫

Problems of interest. We are interested in studying the following two problems:

1. *Existence of solutions.* Given a setting $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$ and an instance I of $\mathcal{R}$, decide whether



(a) G_1. (b) G_2. (c) G_3.

Figure 1: Solutions from Example 2.2. The dotted edges are labeled by *sameAs*.

there exists a solution for I under Ω. Additionally, we are interested in finding in our heterogeneous setting a mechanism similar to universal solutions [11] or universal representatives [5].

2. *Query answering.* Given a setting $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$, an instance I of $\mathcal{R}$, and a query Q over Σ, we are interested in the *certain answers of Q w.r.t. I under Ω*, denoted $cert_\Omega(Q, I)$, which are the answers that hold for all solutions i.e., the set $\bigcap\{[\![Q]\!]_G \mid G \in Sol_\Omega(I)\}$ (where by $[\![Q]\!]_G$ we denote the set of tuples of nodes of G selected by the query Q). The query answering problem consists of deciding whether a given tuple of constants belongs to $cert_\Omega(Q, I)$ or not.

Example 2.2 (continued). Take the above instance I of the relations *Flight* and *Hotel*, and the above setting Ω. Then, take the query

$$Q = (x_1, f \cdot f^*[h] \cdot f^- \cdot (f^-)^*, x_2).$$

Intuitively, this query selects the pairs of nodes (x_1, x_2) from which the same hotel can be reached, or in other words, one can fly (possibly with connections) from the city x_1 to another city that has a hotel and an ingoing flight (possibly with connections) whose origin x_2 we want to select. Recall that the graphs G_1 and G_2 are both solutions for I under Ω. On these two graphs, the query Q selects as follows:

$$\begin{aligned}[\![Q]\!]_{G_1} &= \{(c_1, c_1), (c_1, c_3), (c_3, c_1), (c_3, c_3)\},\\ [\![Q]\!]_{G_2} &= \{(c_1, c_1), (c_1, c_3), (c_3, c_1), (c_3, c_3),\\ &\quad (c_1, N_1), (c_3, N_1), (N_1, c_1), (N_1, c_3), (N_1, N_1)\}.\end{aligned}$$

Notice that only four pairs of nodes are common to these answer sets for the two considered graphs. Also notice that these four pairs of nodes are in fact the certain answers of Q w.r.t. I under Ω:

$$cert_\Omega(Q, I) = \{(c_1, c_1), (c_1, c_3), (c_3, c_1), (c_3, c_3)\}.$$

On the other hand, notice that if we want to pose the same query Q under the other aforementioned example of setting (i.e., Ω'), we obtain a different set of certain answers: $cert_{\Omega'}(Q, I) = \{(c_1, c_1), (c_3, c_3)\}$. Intuitively, this happens because the egds from the setting Ω ensure that in all of its possible solutions the nodes having the same hotel have been merged. In the second setting, this natural requirement has been encoded using a *sameAs* constraint, which is not exploited by the query Q, hence some of the certain answers of Q under Ω are no longer certain under Ω'. ▫

3. BACKGROUND

In this section, we show that in two particular cases of our problem setting existing techniques from relational and graph data exchange can be applied (Section 3.1 and Section 3.2, respectively). This does not happen in the general case, as we show in the next section.

3.1 Relational data exchange

If we consider s-t tgds having on the right hand side conjunctions of NREs of the form a (with $a \in \Sigma$), our problem setting reduces to a particular case of relational data exchange and the techniques from relational data exchange can be naturally applied. In particular, we can see the target schema as a set of binary relational symbols (one for each symbol of the target alphabet) and the chase [11] returns a universal solution that can be essentially seen as a graph since it consists of a set of binary relations.

Example 3.1 Take the schemas $\mathcal{R}$ and Σ, the instance I, and the egds $\mathcal{M}_t$ from Example 2.2. Since we consider only NREs of the form a (with $a \in \Sigma$), we cannot express the same $\mathcal{M}_{st}$ as in Example 2.2. However, we can express the following:

$$\begin{aligned}\mathcal{M}'_{st}: \ & \mathit{Flight}(x_1, x_2, x_3) \wedge \mathit{Hotel}(x_1, x_4) \rightarrow\\ & \exists y.\ (x_2, f, y) \wedge (y, h, x_4) \wedge (y, f, x_3).\end{aligned}$$

We illustrate in Figure 2 the chased solution for I under $(\mathcal{R}, \Sigma, \mathcal{M}'_{st}, \mathcal{M}_t)$. Notice that there is no solution that has N_1 and N_2 on the same path from c_1 to c_2. Such a condition is desirable for a flight from c_1 to c_2 whose passengers have stopped in both hotels hy and hx, situated in the cities N_1 and N_2, respectively. We finally point out that we cannot capture solutions satisfying this kind of constraints for flights with an arbitrary number of stops without using the Kleene star (as in Example 2.2). ▫

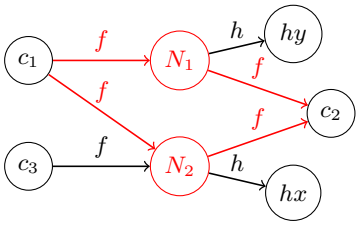


Figure 2: Solution from Example 3.1.

3.2 Graph data exchange

If we consider s-t tgds only, the existence of solutions and query answering can be solved using techniques from graph-to-graph data exchange [5]. In particular, solutions always exist and all solutions are captured by universal representatives defined as graph patterns.

Graph patterns. Let $\mathcal{N}$ be a countably infinite set of labeled null values. A *graph pattern* π over a finite alphabet Σ is a pair (N, D), where $N \subseteq \mathcal{V} \cup \mathcal{N}$ is a finite set of node ids or null values, and $D \subseteq N \times NRE(\Sigma) \times N$, where $NRE(\Sigma)$ denotes the set of all NREs over Σ. The semantics of graph patterns are defined in terms of homomorphisms [4]. Given a graph pattern $\pi = (N, D)$ and a graph database $G = (V, E)$, a *homomorphism* from π into G is a total function $h : N \to V$ s.t.:

1. h is the identity over $N \cap \mathcal{V}$ (i.e., over the node ids from N),
2. for all edges $(u, r, v) \in D$ $(u, v \in N, r \in NRE(\Sigma))$, it holds that $(h(u), h(v)) \in [\![r]\!]_G$.

We write $\pi \to G$ if there exists a homomorphism from π to G. The set of all graphs represented by π over Σ, denoted $Rep_\Sigma(\pi)$ is the set of all graphs G over Σ such that $\pi \to G$.

Universal representatives. Given a setting $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \varnothing)$ and an instance I of $\mathcal{R}$, a graph pattern π is a *universal representative* of I under Ω if $Sol_\Omega(I) = Rep_\Sigma(\pi)$ [5]. In graph data exchange, universal representatives are computed using an adaptation of the standard *chase* procedure from relational data exchange [11]. Moreover, the variant of chase that is tailored for graph data exchange [5] can be easily adapted to construct a universal representative in our relational-to-graph heterogeneous setting. We illustrate a result of this procedure in Example 3.2. Then, query answering reduces to querying the graph pattern [4] chased as universal representative.

Example 3.2 Take the schemas $\mathcal{R}$ and Σ, the instance I, and the s-t tgds $\mathcal{M}_{st}$ from Example 2.2. The graph pattern π in Figure 3 is a universal representative of all solutions for I under $(\mathcal{R}, \Sigma, \mathcal{M}_{st}, \varnothing)$ i.e., all graphs to which there exists a homomorphism from π are solutions. ▫

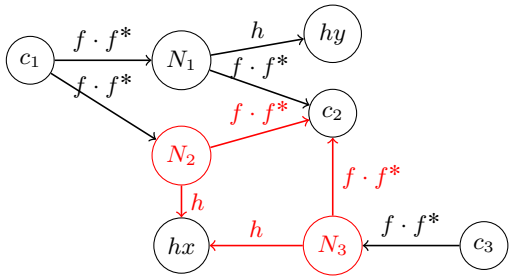


Figure 3: Graph pattern from Example 3.2.

However, notice that the sole s-t tgds might not capture interesting mapping scenarios involving graphs. As an example, the target constraint "a hotel is situated in exactly one city" cannot be expressed in settings such as the one presented in Example 3.2.

4. COMPLEXITY RESULTS

In this section, we present our main contributions. More precisely, we study the complexity of the two problems of interest, namely existence of solutions and query answering, for settings that exhibit target egds (Section 4.1) or target tgds (Section 4.2), respectively.

4.1 Complexity of target egds

First, let us show the intractability of the existence of solutions when we allow egds to our setting.

Theorem 4.1 *Given a setting $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$ where $\mathcal{M}_t$ consists of egds, and an instance I of $\mathcal{R}$, deciding whether there exists a solution for I under Ω is NP-hard.*

Proof. We prove by reduction from 3SAT, known as an NP-complete problem. The reduction works as follows. Given a formula $\rho = C_1 \wedge \ldots \wedge C_k$ in 3CNF over the set of variables $\{x_1, \ldots, x_n\}$, we construct

– The setting $\Omega_\rho = (\mathcal{R}_\rho, \Sigma_\rho, \mathcal{M}_{\rho_{st}}, \mathcal{M}_{\rho_t})$ s.t.

- $\mathcal{R}_\rho = \{R_1, R_2\}$, both unary relations,
- $\Sigma_\rho = \{a, t_1, f_1, \ldots, t_n, f_n\}$.
- $\mathcal{M}_{\rho_{st}}$ contains a unique s-t tgd $R_1(x) \wedge R_2(y) \to (x, a, y) \wedge (x, t_1 + f_1, x) \wedge \ldots \wedge (x, t_n + f_n, x)$.
- $\mathcal{M}_{\rho_t}$ contains two types of egds:

  (*) $(x, (t_j \cdot f_j \cdot a), y) \to (x = y)$, for $1 \leqslant j \leqslant n$,

  (**) $(x, (b_{i_1} \cdot b_{i_2} \cdot b_{i_3} \cdot a), y) \to (x = y)$, for $1 \leqslant i \leqslant k$, for $1 \leqslant i_1, i_2, i_3 \leqslant n$, $x_{i_1}, x_{i_2}, x_{i_3}$ are the variables used in clause C_i, and for $1 \leqslant l \leqslant 3$, b_{i_l} is t_{i_l} if x_{i_l} appears in a negative literal in C_i, and f_{i_l}, otherwise.

– The instance $I_\rho = \{R_1(c_1), R_2(c_2)\}$.

Intuitively, the egds are defined such that a graph collapses if each variable has more than one valuation (*) or the valuation of the variables makes the formula false (**).

We illustrate the construction on the formula $\rho_0 = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$. We have the s-t tgd $R_1(x) \wedge R_2(y) \to (x, a, y) \wedge (x, (t_1 + f_1), x) \wedge \ldots \wedge (x, (t_4 + f_4), x)$, the egds of type (*) $(x, (t_i \cdot f_i \cdot a), y) \to (x = y)$ (with $1 \leqslant i \leqslant 4$), and the egds of type (**) $(x, (f_1 \cdot t_2 \cdot f_3 \cdot a), y) \to (x = y)$ and $(x, (t_1 \cdot f_3 \cdot t_4 \cdot a), y) \to (x = y)$. Then, the graph in Figure 4 is a solution that encodes the valuation v s.t. $v(x_1) = v(x_2)$=true and $v(x_3) = v(x_4)$=false that makes ρ_0 true.

We claim that there exists a solution for I_ρ under Ω_ρ iff $\rho \in$3SAT.

For the *if* part, we show that the existence of a valuation making ρ true implies the existence of a solution. Take a valuation $v : \{x_1, \ldots, x_n\} \to \{\text{true, false}\}$ making ρ true. Then, construct the graph $G = (\{c_1, c_2\}, E)$ s.t. $E = \{(c_1, a, c_2)\} \cup \{(c_1, t_i, c_1) \mid 1 \leqslant i \leqslant n \text{ and } v(x_i) = \text{true}\} \cup \{(c_1, f_i, c_1) \mid 1 \leqslant i \leqslant n \text{ and } v(x_i) = \text{false}\}$. Note that G and I_ρ satisfy the s-t tgd. Since there is exactly one edge labeled $b_i \in \{t_i, f_i\}$ from c_1 to c_2, the egds of type (*) are satisfied. Moreover, since the b_i's correspond to a valuation making ρ true, there is at least one satisfied literal in every clause of ρ, hence the egds of type (**) are also satisfied. Thus, G is a solution.

For the *only if* part, take a solution G. Since G satisfies the s-t tgd, we infer that G encodes at least one valuation

of every variable. Since G satisfies the egds of type (*), we infer that G encodes at most one valuation of every variable. Thus, G encodes exactly one valuation of every variable. Since G satisfies the egds of type (**), we conclude that G encodes a valuation making ρ true. ▫

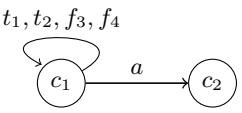


Figure 4: Solution for ρ_0.

We point out that Theorem 4.1 holds even under significantly restricted assumptions that have been used in the proof: (i) *fixed source schema* consisting of two unary relations only, (ii) *fixed source instance*, (iii) s-t tgds using only conjunctions of NREs of the form a or $a + b$ (with $a, b \in \Sigma$) that is a slight relaxation of the restriction from Section 3.1, and (iv) egds that use only NREs of the form $a_1 \cdot \ldots \cdot a_n$, with pairwise distinct $a_1, \ldots, a_n \in \Sigma$ (NREs referred to as "SORE(·)" [2]). Next, we prove that query answering is intractable under the same assumptions and for queries consisting of NREs that use disjunction and concatenation only.

Corollary 4.2 *Given a setting $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$ where $\mathcal{M}_t$ consists of egds, an instance I of $\mathcal{R}$, a NRE r, and a tuple of constants (c_1, c_2), deciding whether $(c_1, c_2) \in cert_\Omega(r, I)$ is coNP-hard.*

Proof. We take the proof of Theorem 4.1, and we additionally consider the NRE $r_\rho = a \cdot a$. We claim that $(c_1, c_2) \in cert_{\Omega_\rho}(r_\rho, I_\rho)$ iff $\rho \notin$3SAT. For the *if* part, notice that $\rho \notin$3SAT implies that there is no solution hence (c_1, c_2) is a certain answer. For the *only if* part, since (c_1, c_2) is a certain answer, we infer that either (i) there is no solution or (ii) there is at least a solution and (c_1, c_2) is an answer for all solutions. But (ii) is false since there exist solutions for which (c_1, c_2) is not an answer for r_ρ (e.g., in Figure 4). Both parts follow directly from the proof of Theorem 4.1. ▫

Finally, we point out that in our reduction the source schema and instance are fixed while the target schema and the mappings are part of the input. Hence, our hardness results stand in terms of *query complexity*. Similar intractability results in the presence of target constraints (particularly in combined complexity) have been shown for relational and XML data exchange [12, 3, 1, 10]. However, our contribution is novel since to the best of our knowledge target constraints on a graph target schema have not been previously considered in the literature, and moreover, we use a fixed source schema and instance in the proof. We also point out that our results are not specific to the relational-to-graph setting and hold in any setting where the target is a graph.

4.2 Complexity of target tgds

In this section, we use *sameAs* constraints instead of egds. First, let us show that the existence of solutions becomes trivial. More precisely, a solution can be computed as follows: (i) chase a graph pattern π using the s-t tgds only, (ii) take a graph G s.t. $\pi \to G$, and (iii) add in G the necessary *sameAs* edges to satisfy the *sameAs* constraints. Recall that the difficulty of deciding the existence of solutions in the case of egds was that we cannot merge two constants. Notice that this difficulty is overcome since we can add *sameAs* edges between any two nodes, even between two constants.

Next, we prove that in the presence of *sameAs* constraints the problem of certain answers is intractable under the same assumptions as in Section 4.1.

Proposition 4.3 *Given a setting $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$ where $\mathcal{M}_t$ consists of sameAs constraints, an instance I of $\mathcal{R}$, a NRE r, and a tuple of constants (c_1, c_2), deciding whether $(c_1, c_2) \in cert_\Omega(r, I)$ is coNP-hard.*

Proof. We take from the proof of Theorem 4.1 the same $\mathcal{R}_\rho$, I_ρ, Σ_ρ, $\mathcal{M}_{\rho_{st}}$, and we replace each $(x = y)$ from $\mathcal{M}_{\rho_t}$ by $(x, sameAs, y)$ to obtain the set of *sameAs* constraints $\mathcal{M}'_{\rho_t}$ and $\Omega'_\rho = (\mathcal{R}_\rho, \Sigma_\rho \cup \{sameAs\}, \mathcal{M}_{\rho_{st}}, \mathcal{M}'_{\rho_t})$. Then, take $r'_\rho = sameAs$. We claim that $(c_1, c_2) \in cert_{\Omega'_\rho}(r'_\rho, I_\rho)$ iff $\rho \notin$3SAT, which follows similarly to Theorem 4.1. ▫

Moreover, we observe that *sameAs* constraints are a particular case of target tgds, and therefore, query answering is intractable in the presence of target tgds.

Corollary 4.4 *Given a setting $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$ where $\mathcal{M}_t$ consists of target tgds, an instance I of $\mathcal{R}$, a NRE r, and a tuple of constants (c_1, c_2), deciding whether $(c_1, c_2) \in cert_\Omega(r, I)$ is coNP-hard.*

5. TOWARDS UNIVERSAL SOLUTIONS

Next, we study a natural adaptation of the standard *chase* procedure [11] to take into account egds. The result of our adapted chase is a graph pattern π. To this purpose, we consider two types of chase steps: (1) for s-t tgds we do similarly to [5] when computing universal representatives in graph data exchange without target constraints, and (2) for egds, for each $\psi_\Sigma(\overline{x}) \to (x_1 = x_2)$, (i) if the images in π of both x_1 and x_2 are constants, then the chase fails, (ii) if one has as image in π a constant and the other a labeled null, then the chase replaces in π the labeled null by the constant, and (iii) if both have labeled nulls as images in π, the chase chooses one of them and replaces it in π with the other.

Example 5.1 For the setting $(\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$ and the instance I from Example 2.2, by applying the aforementioned adapted chase we obtain the graph pattern in Figure 5. ▫

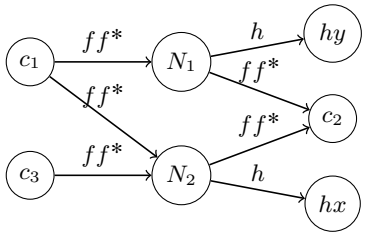


Figure 5: Graph pattern from Example 5.1.

As for relational data exchange, if the chase fails, then there is no solution. As opposed to relational data exchange, we observe that a successful chase does not guarantee the existence of a solution. Intuitively, the difficulty comes from the fact that the chase result is a graph pattern with NREs on the edges (unlike a graph with symbols on the edges). Consequently, there might not exist any graph G s.t. $\pi \to G$ and

G satisfies the target constraints because it may be the case that there is no path satisfying the NREs and the egds at the same time. The following example shows such a situation.

Example 5.2 Take the source schema $\{R, P\}$, an instance $R(c_1)$ and $P(c_2)$, the target schema $\{a, b, c\}$, the s-t tgd $R(x) \wedge P(y) \rightarrow (x, a \cdot (b^* + c^*) \cdot a, y)$, and the egd $(x, a + b + c, y) \rightarrow (x = y)$. The aforementioned adapted chase succeeds and returns the graph pattern π in Figure 6(a). Although the chase has not failed, no solution exists because there is no graph G s.t. $\pi \rightarrow G$ and G satisfies the egds. In particular, the graph G (s.t. $\pi \rightarrow G$) from Figure 6(b) satisfies the s-t tgd but if we try to transform it in a solution we fail because we attempt to merge two constants. ▫

(a) Graph pattern π. (b) Graph G.

Figure 6: Result of a successful chase.

We next show that, even when solutions exist, graph patterns as such cannot be used as universal representatives in the presence of egds.

Proposition 5.3 *Given a setting $\Omega = (\mathcal{R}, \Sigma, \mathcal{M}_{st}, \mathcal{M}_t)$ where $\mathcal{M}_t$ consists of a non-empty set of egds, and an instance I of $\mathcal{R}$, there does not exist a graph pattern π s.t. $Sol_\Omega(I) = Rep_\Sigma(\pi)$.*

Intuitively, let us assume that there exists a graph pattern π s.t. $Sol_\Omega(I) = Rep_\Sigma(\pi)$. Then, if we take a graph $G \in Sol_\Omega$ and a homomorphism $h : \pi \rightarrow G$, we can construct the graph G' by adding nodes and edges to G s.t. some egd is no longer satisfied, thus G' is not a solution for I under Ω, but $h : \pi \rightarrow G'$ is still a homomorphism. The next example clarifies when such a situation can occur.

Example 5.4 The graph in Figure 7 is not a solution for the mappings and instance from Example 2.2 although there exists a homomorphism from the chased graph pattern from Figure 5. ▫

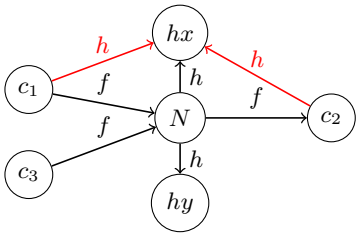


Figure 7: Graph from Example 5.4.

To address the problem of universal representatives in settings involving egds, a natural approach is to define the universal representative as a pair (graph pattern, set of egds). In this case, the solutions are the graphs satisfying the egds and s.t. there exists a homomorphism from the pattern. For example, the universal representatives for Example 2.2 would be the pattern in Figure 5 together with the egd in $\mathcal{M}_t$ from Example 2.2. We also point out that the above discussion can be easily generalized for *sameAs* constraints or arbitrary target tgds.

6. CONCLUSIONS AND FUTURE WORK

We have presented our work on relational-to-graph data exchange. Our main results are the proofs of intractability of the existence of solutions and query answering that hold even under considerable restrictions of the problem setting. As future work, we would like to investigate the complexity upper bounds of these problems and look for tractable fragments to have a complete picture of the difficulty of our setting. A natural question that remains open is how to query universal representatives consisting of a pair (graph pattern, set of target constraints). We would also like to investigate practical scenarios of relational-to-RDF data exchange and other classes of heterogeneous schema mappings. Additionally, it would be interesting to combine existing learning techniques for relational [9] and graph [8] queries in order to propose algorithms that automatically infer relational-to-graph mappings from examples provided by the user.

7. REFERENCES

[1] S. Amano, C. David, L. Libkin, and F. Murlak. XML schema mappings: Data exchange and metadata management. *J. ACM*, 61(2):12, 2014.

[2] T. Antonopoulos, F. Neven, and F. Servais. Definability problems for graph query languages. In *ICDT*, pages 141–152, 2013.

[3] M. Arenas and L. Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2), 2008.

[4] P. Barceló, L. Libkin, and J. L. Reutter. Querying graph patterns. In *PODS*, pages 199–210, 2011.

[5] P. Barceló, J. Pérez, and J. L. Reutter. Schema mappings and data exchange for graph databases. In *ICDT*, pages 189–200, 2013.

[6] C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.

[7] Z. Bellahsene, A. Bonifati, and E. Rahm, editors. *Schema Matching and Mapping*. Springer, 2011.

[8] A. Bonifati, R. Ciucanu, and A. Lemay. Learning path queries on graph databases. In *EDBT*, 2015.

[9] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive inference of join queries. In *EDBT*, pages 451–462, 2014.

[10] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res. (JAIR)*, 48:115–174, 2013.

[11] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[12] P. G. Kolaitis, J. Panttaja, and W. C. Tan. The complexity of data exchange. In *PODS*, pages 30–39, 2006.

[13] P. G. Kolaitis, R. Pichler, E. Sallinger, and V. Savenkov. Nested dependencies: structure and reasoning. In *PODS*, pages 176–187, 2014.

[14] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.

[15] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *VLDB*, pages 598–609, 2002.

[16] J. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to RDF and OWL. In *WWW*, pages 649–658, 2012.

Topic Detection Using a Critical Term Graph on News-Related Tweets

Paraskevas Tsantarliotis
Department of Computer Science & Engineering
University of Ioannina
Ioannina, Greece
ptsantar@cs.uoi.gr

Evaggelia Pitoura
Department of Computer Science & Engineering
University of Ioannina
Ioannina, Greece
pitoura@cs.uoi.gr

ABSTRACT

Social media and online social networks are playing an increasingly important role in our lives, as they attract millions of users around the world. Twitter, one of the most popular micro-blogging services, holds a special position among them, since information shared through this service spreads faster than it would have been possible with traditional sources. There are many interesting works analyzing the information that flows through Twitter. Most of such research focuses on trending topic detection, i.e. what are the people talking about right now. We propose a new method to detect topics using a graph, where nodes correspond to terms and edges correspond to co-occurrence of the two terms in the tweet stream. Dense subgraphs, of this graph, pose special interest, as the nodes that are highly connected share a special relation. Thus, the corresponding terms potentially share a relation too. To explore this fact, we apply a community detection algorithm on the graph. The resulting communities correspond to topics related to various real world events. Experimental evaluation of the results of this technique is also provided on both synthetic and real data.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering; H.4.m [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms

Keywords

topic detection, community detection, term mining

1. INTRODUCTION

In recent years, usage of social media has overcome any expectation. Millions of users from all over the world post content on online social networks, forums or their blogs or subscribe to micro-blogging services. As a result, social media, especially online social networks, have been transformed to a powerful mean of disseminating news (e.g. describing real-world events).

In particular, Twitter [15] is both an online social network and a micro-blogging service, which enables users to send and read short 140-character text messages, called "tweets". The context of tweets varies from chit-chat to political sentiment, creating a very interesting stream of information. Thus, Twitter can be described as an information/data network. Such a network contains important data and poses exploration opportunities and challenges, such as discovering and browsing valuable information.

Most of the efforts focus on trending topic detection. There are many reasons why researchers focus on this particular problem. First of all, it describes what the people are talking about right now. Furthermore, this can be a powerful tool for marketing specialists and opinion tracking companies, since trending topics can describe the opinion and intentions of a large group of people. There are a lot of services and sites dedicated to finding trending topics, such as Trends24[1], TrendsMap[2] and WhatTheTrend[3]. Usually, these services use - one or two - frequent terms or frequent hashtags ("#") to describe a topic. However, using more than a couple of terms to describe a topic would be more informative, e.g. instead of "Edward, Snowden", we would prefer something like: "Edward, Snowden, NSA, surveillance, illegal", which is much more expressive.

In this paper, we propose a new intuitive way to detect topics on data streams where a topic is described by a number of terms. We construct a graph whose nodes correspond to terms appearing in tweets. Two nodes are connected if only they co-occur in the same tweet and the weight of the edge corresponds to the co-occurrence frequency. We call this graph *critical term graph*. Constructing such a graph can be very expensive for large documents, but is suitable for short document, such as tweets. We will discuss how this graph is constructed and its properties in detail later in this paper. Based on this graph, we extract topics from its dense subgraphs. We use a community detection algorithm, which partitions the graph into sets of nodes that are tightly connected with each other and sparsely connected with nodes that belong to different communities. Thus, in this case the communities represent the topics. The output of the algo-

[1] http://trends24.in
[2] http://trendsmap.com
[3] http://whatthetrend.com

rithm is displayed using the visualization tool Gephi[1]. We also provide an evaluation of our system and its variants both on real and synthetic datasets.

In this paper we focus on detecting topics in general and not trending topics. Nevertheless, the technique we propose could possibly work for real time trending topic detection. The main contribution of the paper is the study of the critical term graph and the feasibility of applying community detection algorithms on this graph to identify topics. To this end, we design a new model for generating synthetic tweets by controlling the number of topics and the overlap between them. We also test our algorithms on real news-related tweets, empirically evaluating our model.

In Section 2 we describe works related to ours and in Section 3 we describe in detail the problem and the proposed solution. Evaluation of our results are presented in Section 4. Finally, in Sections 5, we discuss future work and summarize our conclusions.

2. RELATED WORK

Work related to ours includes research both in the broad areas of mining data streams and graph mining, specifically in the areas of topic detection in micro-blogging services and community detection.

There are many papers that focus on identifying trending topics on Twitter like [5, 7, 17, 19]. The authors of [5] describe some interesting methodologies of detecting and identifying trending topics, but they limit their results to unigrams and bigrams. The authors of [17] present TwitterMonitor, which detects bursty keywords and groups them to form clusters. In that short paper, it is made clear that a single pass over the data stream is not enough to detect bursty keywords, but no algorithmic nor experimental details are given. Another interesting approach is described in [7], they focus on detecting trending topics based on metrics that involve both the frequency of the terms and the authority of the users, who wrote the tweets. However, their method requires high computational load. Furthermore, information about the users may not be available in large datasets, because of the restrictions of the Twitter API.

In [19], the authors propose a complete system for detecting trending topics from Twitter posts in near-real time. Their algorithm is similar to the Apriori approach [4] and relies on finding frequent multi-word clusters, that represent a topic, and then calculating the burstiness of each topic. Their approach is based on a hypothesis, similar to ours, that if AB, BC and AC are frequent terms pairs then ABC is frequent. This is not necessarily true, but as we see in our results, it is very likely. Consequently, this hypothesis introduces false positive results. Furthermore, the algorithm used in [19] for topic detection is also interesting, because their work could possibly be extended to be used on the graph-based model we propose.

Graph mining is also related to our work, since we use such techniques to extract useful information. Currently, we focus on community detection algorithms, but other algorithms may be considered in the future. Community detection in graphs is thoroughly investigated in [10]. In general, community detection is used to uncover relation between nodes in complex networks in biology, computer science, sociology, etc. For example, in [9], they use similar technique in the Web to discover communities of Web pages dealing with the same topic. The authors of [8] propose an algorithm for dense subgraph extraction and they test it on a graph that represents the relationships between terms, obtained by a news agency. Their technique manages to group in the same subgraphs, terms that tend to be used together. However, no further analysis is provided. In [13] authors use a similar approach to ours, in order to detect topics related to a particular event, but their main focus is on the evolution of these topics through time. Instead, we use the model to detect topics under real conditions and provide more information about the model.

3. MODEL

In this section, we describe the basic concepts of our approach and the algorithm which we use to detect topics. For the description of the model, we assume one segment of the tweet stream. We discuss later how the method is extended to be applied to to a sequence of segments.

Topic. Similar to other works [17, 19] we define a topic to be a set of terms. Usually, a topic consists of three or more terms, in order to capture its essence. Terms in each set must be frequent, i.e. surpass some threshold, and co-occur with some other words in the topic. Thus, we are interested in looking for frequent term pairs and our goal is to merge these pair to form lager sets, which represent the topics.

Critical Pairs. Similar to [19], in order to detect frequent pairs of terms, we use a support measure:

$$sup(\tau_i, \tau_j) = \frac{|D_{ij}|}{|D|},$$

where τ_i, τ_j are terms, D_{ij} is the set of tweets that the two terms co-occur and D is the total set of tweets. A term pair is frequent if its support is equal or larger than a support threshold. In [19], the authors define a static value for the support. There are some drawbacks when a fixed support is used. As the input size varies, the output of the algorithms may elide some topics, or include too many topics. Instead of a static support threshold, we use a method defining a dynamic support threshold, i.e. the threshold adapts to the size and the content of the input. To achieve this, we sort the co-occurrence frequency of all possible two-word pairs in descending order. The N first pairs are called critical pairs and we use the support of the N-th pair as threshold.

Critical Term Graph. Here, we describe the basic concept of our work. Using the critical pairs, we create a graph, where nodes represent terms that are used in the tweets and edges indicate co-occurrence of the two terms in them. We assign a value to each node, which equals to the term frequency in the tweet corpus. Similarly, we assign to each edge

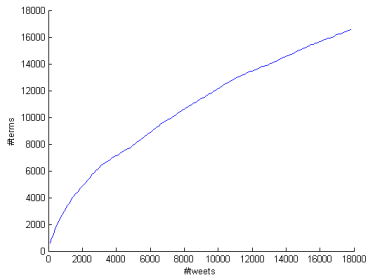


Figure 1: Number of unique terms in tweets.

Figure 2: Results from November 12 2014.

a weight that corresponds to the co-occurrence frequency of its incident terms in the tweets. A normalization of these values, may also be considered.

An important property of this graph is that as the number of tweets increases, the number of newly seen terms decreases and at some point we expect to have more tweets than unique terms, i.e. the tweet corpus obeys to the Heap's Law. In Figure 1, we compare the number of unique terms to the number of tweets in a random stream of all public tweets, acquired by Twitter API[4]. At first, the number of unique terms is larger than the number of tweets, but in the end there are more tweets than unique terms. We expect this diagram to flat out much faster when tweets focus on specific topics, because the vocabulary should be limited. Thus, as the number of tweets increases, it is more likely new edges to be created or the existing edges to become stronger.

Generally, we expect the critical graph to be sparse, not well connected and have community structure (Figure 2). The communities are formed from terms that co-occur in the tweets, thus creating topics. The graph contains a lot of non-connected components. Usually, the small non-connected components (of size < 10) correspond to standalone topics that have no common terms with the other topics. The number of larger non-connected components, on the other hand, is much smaller and these components contain more topics.

3.1 Community Detection Algorithm

Preprocessing. When a new tweet arrives at the system, it gets preprocessed and then stored. Preprocessing involves splitting the tweet in a term list, removing stopwords, stemming the terms and generating all possible term pairs from the term list. We keep a record for the co-occurrence frequency of each term pair, in every time segment.

To detect communities we use a modularity based algorithm [6]. Several algorithms have been proposed to detect communities efficiently. Newman and Girvan [18] introduced modularity as a measure for the quality of the resulting communities. Modularity measures the density of edges inside a community compared to the expected density if the edges were distributed at random. Modularity ranges between -1 and 1. If the modularity of a subgraph tends to 1, it means that the sub-graph is tightly connected and may be a community and vice versa. In weighted networks modularity is defined in [6] as:

$$\frac{1}{2m}\sum_{i,j}[A_{ij} - \frac{k_i k_j}{2m}]\delta(c_i, c_j),$$

where A_{ij} represents the weight of the edge between i and j, $k_i = \Sigma_j A_{ij}$ is the sum of the weights of the edges attached to vertex i, c_i is the community to which node i is assigned, the δ-function $\delta(u,v)$ is 1 if $u = v$ and 0 otherwise and $m = \frac{1}{2}\ \Sigma_{ij} A_{ij}$.

It is easy to see that the community detection problem is equivalent to modularity optimization in a graph. In [6], the authors propose an algorithm consisting of two phases that are repeated iteratively.

Initially, every node is a community. Let i be a node of the graph and j its neighbors. The algorithm evaluates the gain of modularity that would be achieved if node i was removed from its community and placed in the community of j. If the maximum gain is positive, then node i is placed in the community for which the gain is maximum, otherwise the node i stays in its community. This step is repeated for every node in the graph until there is no improvement of modularity. Note that each node can be considered several times, during this phase.

In the second phase, a new graph is built with nodes the communities found in the first phase and edges that have weights equal to the sum of weight of the edges between nodes in the corresponding two communities (edges between nodes of the same community are marked as self-loops). Phase one is applied to the new graph. The two phases are repeated until there are no changes and modularity is maximized.We chose this algorithm because it is fast; its complexity is linear on typical and sparse data [6]. In addition, the algorithm follows an unsupervised approach, so we need no previous knowledge about the number of topics we are looking for.

It is known that modularity optimization algorithms suffer a resolution limit [11], i.e. they fail to resolve communities smaller than a certain scale, even if they are well defined. However, because this algorithm, in intermediate steps, merges communities in order to create new communities, there is a sense of hierarchy. Thus, we are able to use this hierarchical structure to detect smaller communities.

3.2 Snapshots

So far, we have considered topics in a single sequence of tweets. To handle streams of tweets, we divide the stream in small segments of fixed time length. Each segment is marked

[4] `https://dev.twitter.com/streaming/`

with a time id, based on its start and end timestamps. For each segment we store all the possible term pairs and their frequencies, that we have extracted from tweets.Given a time range, to detect topics within this time range, we first identify the appropriate segments that correspond to this specific segment. Then, we group all the term pairs from the appropriate segments, get the critical pairs and finally create the critical term graph. This graph is a snapshot of the graph for the specific time range. Snapshots are useful for tracking topics over time, which is considered as future work (see Section 5).

4. EXPERIMENTAL RESULTS

In this section, we demonstrate the initial results of our approach both on synthetic and real data.

4.1 Synthetic Data

One of the difficulties we faced is the lack of datasets with known ground truth. In order to test our approach under various conditions, we created a model to generate synthetic datasets similar to the one described in [14]. Since real tweets obey to the Heaps' law, the synthetic tweets should be generated by randomly sampling terms based on a Zipf distribution. We present a short description and preliminary results of the model we used to generate synthetic tweets.

We assume that we have k topics. Our goal is to generate tweets for each of the k topics. Initially, we generate the terms, which will be used in the tweets. Similar to [14], we consider *k+1* topic vocabulary bags, i.e. bag B_i=1,...,k contains the terms for the i-th topic and bag B_{k+1} contains general terms. General terms can be used in all topics, they are like stopwords but contain more valuable information for the topic.

Then, we create a $k \times$ *(k+1)* matrix P. Each cell P_{ij} of the matrix corresponds to the probability to pick a term from vocabulary bag B_j while generating tweets for the i-th topic. We call overlap between topics, the case in which when we generate a tweet for a topic i, we include a term from a bag B_j that is different from the bag corresponding to the topic, i.e., $B_i \neq B_j$ Note that vocabulary bag B_{k+1} should have similar probability in every topic. We describe the process that generates tweets for each topic below.

Let t and M be the topic that the tweets refer to and the number of tweets we want to generate for this topic, respectively. At first, we assign occurrence frequency to all terms based on Pareto distribution. We must point out that general terms should have the same frequency in all topics, in order to avoid special relation with any topic. Then, we generate M tweets for the topic. Note that we do not want duplicate tweets. Tweets are considered as a set of terms that make up the tweet. We noticed in our dataset that the number of terms used in the tweets (not including stopwords) follows the Poisson distribution. Thus, we first decide the size of the tweet S and then pick S terms. Term picking can be summarized in two steps:

1. Select a vocabulary bag B_i based on the probability of matrix P_k.

2. Select term from B_i based on occurrence frequency, i.e. terms with high occurrence frequency are more likely to be picked. Note that we don't want duplicate terms to avoid spam.

We also need to check if the tweet is unique. If so, we decrease the frequency of each term used in the tweet by 1. Otherwise, we create a new tweet. This process is repeated until we create tweets for all k topics. Therefore, the output the of the model is $k{\cdot}M$ synthetic tweets.

In order to evaluate the results of our model, we consider community detection as a problem of assigning all similar nodes to the same communities [3]. Thus, based on pair counting, we can predict the following cases:

- True Positive (TP): Term pairs that belong to the same topic are assigned to the same communities.
- True Negative (TN): Term pairs that belong to different topics are assigned to different communities.
- False Negative (FN): Term pairs that belong to the same topic are assigned to different communities.
- False Positive (FP): Term pairs that belong to different topics are assigned to the same community.

Using the above, we can calculate the following evaluation measures, which are defined in [3, 12]:

- Precision: $P = \frac{TP}{TP + FP}$
- Recall: $R = \frac{TP}{TP + FN}$
- Jaccard Coefficient: $J = \frac{TP}{TP + FP + FN}$
- Rand Statistic: $RS = \frac{TP + TN}{TP + TN + FP + FN}$

Figures 3 and 4 show the results of our approach. In Figure 3, we present three different plots comparing the number of topics that our model estimated to the real number of topics. We test our model using different amounts of critical pairs, in three different percentages of overlap between topics, 5%, 15% and 25%. The overlap between topics is distributed randomly. The synthetic datasets are generated using 3% general terms in each topic and they contain 3 to 10 frequent topics and 2 to 10 less frequent topics, for each experiment. Note that the experiments have been repeated multiple times and we present average values.

We notice that our model in most cases predicts the correct number of topics. The maximum discord, between the estimated and the real number of topics occurs when using 1000 critical pairs to detect topics in larger datasets. In these cases, the model predicts less topics than the actual. But as we can see in Figure 4 the detected topics are the frequent ones, because all the evaluation metrics are close to 1. We must also point out that using more than 1000 critical pairs in the datasets with 5 to 10 topics does not perform well. Even though the model predicts almost the correct number of topics, the critical pairs contain noise and this leads to estimating false topics. As we can see in Figure 4 as the noise increases the corresponding metrics are decreasing.

The number N of critical pairs affects the number of topics detected. A default value of 2000 seems to work in most cases. By reducing N, we may end up loosing some topics, however, the ones detected are the most important (i.e., the most frequent) ones. By increasing N, we may get false results when the number of actual topics is small. This is one

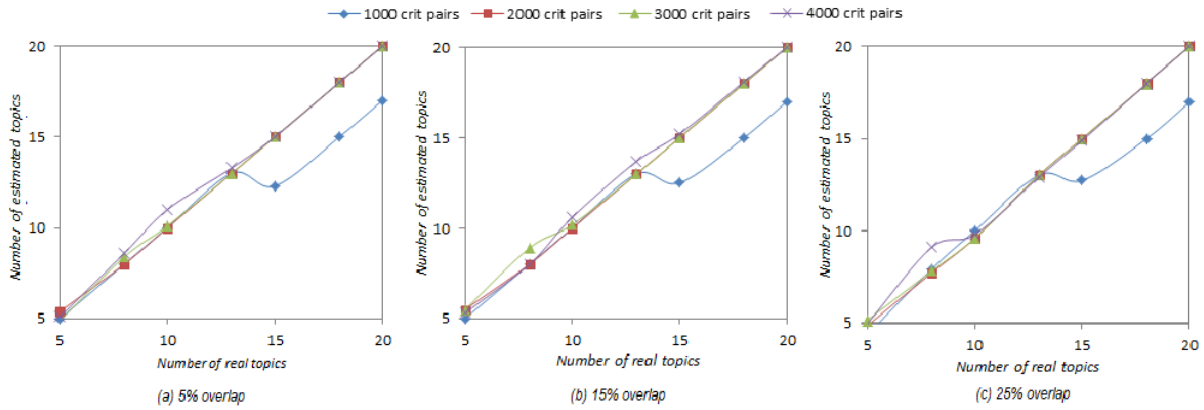


Figure 3: Comparison of the number of estimated topics to the number of real topics, using different amounts of critical pairs and increasingly overlapping topics ((a) 5%, (b) 15% and (c) 25%).

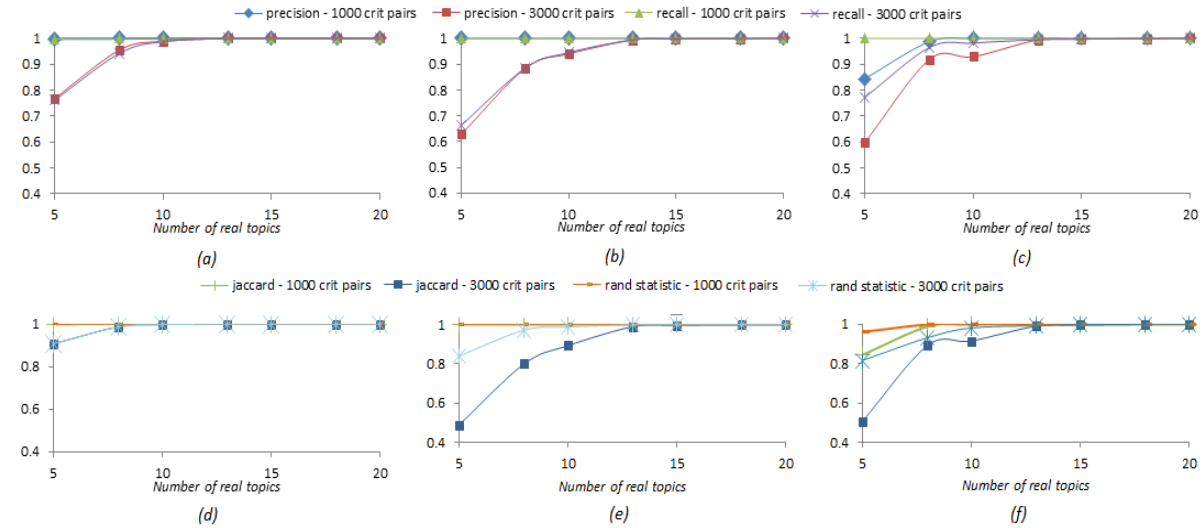


Figure 4: Evaluation metrics on synthetic datasets using 1000 and 3000 critical pairs and increasingly overlapping topics ((a) & (d) 5%, (b) & (e) 15% and (c) & (f) 25%).

of the issues we plan to address in future work. An initial idea is to dynamically adjust N, by looking at the actual frequency of the N-th pair in adjacent segments.

4.2 Real Data

We provide a set of experimental results based on real data and discuss their quality. The datasets used in these experiments are closely related to news, sports and lifestyle. This can help us to get an empirical evaluation of the results. We have implemented a crawler, which follows public accounts of popular news agencies, magazines, politicians, celebrities, etc and people that are related to them. For example, some of the accounts are Barak Obama, The Guardian, various journalists, N.A.S.A. and F.I.F.A.com. Most of these accounts are verified by Twitter, to avoid spam, and their tweets are written in English.

Thus, the context of the dataset varies from political to sport related events. We have to point out that the tweets are related to real world events, the popularity of these events directly affects our results. An example it is shown in Figure 2. We got 10 topics that happened at November 12 of 2014. The most obvious of the them refers to first ever successful land of a spacecraft (Philae) on a comet (Comet 67P). We can also see information about the U.S.-China emissions deal, the Forex scandal, the Ebola outbreak and a rather bizarre news about a loose tiger near Paris.

Similarly, Figure 5 displays topic from (Monday) December 1 2014. We see topics that refer to World AIDS Day, new Star Wars movie trailer, cyber Monday and black Friday sales. Another interesting example in the same figure is that the algorithm manages to detect different topics for two different protests in two different places, Hong Kong and Ferguson U.S. respectively, even though they have some common words. Still, there are some terms that could belong to both topics, but since the algorithm produces non-overlapping communities, this is not possible.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a model to detect topics in

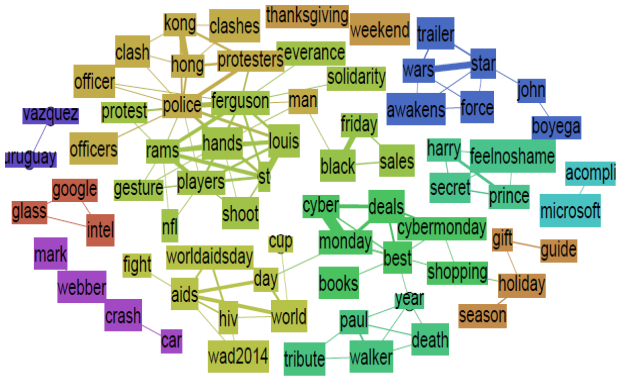


Figure 5: Results from December 1 2014.

short text documents, such as tweets, using a graph-based model, the critical term graph. This graph consists of nodes, which represent the terms found in tweets, and edges, which represent the co-occurrence frequency of the corresponding terms. Detecting communities in such graph can be an efficient tool for topic detection. We empirically proved the effectiveness of our model with experiments on real news-related tweets. In addition, we proposed a method to create synthetic tweets, and tested our model under various conditions.

There are many possible improvements and extensions. At first, we can improve the critical term graph, using Natural Language Processing techniques on the tweets. Synonyms could be considered duplicate information in the critical term graph, thus these terms could be grouped. Furthermore, verbs could also be eliminated, because in most cases they do not add much valuable information. These improvements would reduce the graph size and potentially improve the performance of the model. Both synonyms and verbs can be detected using simple dictionary based methods, such as WordNet[2]. Alternatively, synonyms can be detected using more advanced methods like [16].

Furthermore, an evaluation process of the results based on real data can be considered by using information from external sources, such as Google Trends or news agencies.

We are currently focusing on different ways on assigning weights on the edges and how the algorithm responds to these changes. The quality of results when using other overlapping and non-overlapping community detection (or graph partitioning) algorithms is, also, another interesting topic.

Another extension could be the use of this model in trending topic detection. This can be achieved by using the concept of burstiness. In [19], the authors define the trending topics as topics that exhibit high frequency in a specific time segment compared to its previous segments. In our model, this could be achieved by using additional constraints on selecting the critical pairs, e.g. the occurrence frequency of the corresponding terms should exhibit bursty behavior.

Our ultimate goal is to use the critical term graph for tracking the evolution of the topics through time. We are considering two approaches on this problem. The first approach involves building different critical term graphs for each snapshot and then trying to map communities, and thus the corresponding topics, between these snapshots. Alternatively, we can maintain a single critical term graph and incrementally update the graph by adding, modifying or removing edges and nodes.

6. ACKNOWLEDGMENTS

This work was co-financed by the European Union (ERDF) and Greek national funds through the Operational Program Thessaly-Mainland Greece and Epirus 2007-2013 of the NSRF 2007-2013. We also thank A. Likas for the insightful discussions and suggestions.

7. REFERENCES

[1] Gephi, `https://gephi.org/`.
[2] Wordnet, `http://wordnet.princeton.edu/`.
[3] *Social Media Mining*, chapter 6. Cambridge University Press, 2014.
[4] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. VLDB*, volume 1215, pages 487–499, 1994.
[5] J. Benhardus and J. Kalita. Streaming trend detection in twitter. *International Journal of Web Based Communities*, 9(1):122–139, 2013.
[6] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of community hierarchies in large networks. *CoRR*, abs/0803.0476, 2008.
[7] M. Cataldi, L. Di Caro, and C. Schifanella. Emerging topic detection on twitter based on temporal and social terms evaluation. In *Proceedings of the Tenth International Workshop on Multimedia Data Mining*, page 4. ACM, 2010.
[8] J. Chen and Y. Saad. Dense subgraph extraction with application to community detection. *Knowledge and Data Engineering, IEEE Transactions on*, 24(7):1216–1230, 2012.
[9] G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee. Self-organization and identification of web communities. *Computer*, 35(3):66–70, 2002.
[10] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.
[11] S. Fortunato and M. Barthelemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.
[12] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Cluster validity methods: Part I. *SIGMOD Record*, 31(2):40–45, 2002.
[13] S. Higuchi, T. Hashimoto, T. Kuboyama, and K. Hirata. Exploring social context from buzz marketing site-community mapping based on tree edit distance. In *PERCOM Workshops, 2013 IEEE International Conference*, pages 187–192. IEEE, 2013.
[14] A. Kalogeratos and A. Likas. Document clustering using synthetic cluster prototypes. *Data Knowl. Eng.*, 70(3):284–306, 2011.
[15] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on WWW*, pages 591–600. ACM, 2010.
[16] Y. Li, B.-J. P. Hsu, C. Zhai, and K. Wang. Mining entity attribute synonyms via compact clustering. In *Proceedings of the 22nd ACM CIKM*, pages 867–872. ACM, 2013.
[17] M. Mathioudakis and N. Koudas. Twittermonitor: Trend detection over the twitter stream. In *Proceedings of the 2010 ACM SIGMOD*, pages 1155–1158. ACM, 2010.
[18] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.
[19] N. Pervin, F. Fang, A. Datta, K. Dutta, and D. Vandermeer. Fast, scalable, and context-sensitive detection of trending topics in microblog post streams. *ACM TMIS*, 3(4):19, 2013.

Graph Databases and Railway Operations Research Requirements

Alexander Kuckelberg
VIA Consulting & Development GmbH
Römerstr. 48-50
D-52064 Aachen
+49 241 463 662-16
a.kuckelberg@via-con.de

ABSTRACT

In this work-in-progress paper we describe requirements, scenarios and mandatory functionalities of graph databases within the application field of railway operations research (ROR).

The underlying railway infrastructure data of all ROR tasks can naturally be described by graph structures and can therefore be managed by graph databases; railway operations research functionalities might consequently be described as database functions on its graphs.

While the infrastructure data should remain persistent, a graph database might be a good choice to match the persistence needs quite close or even identical to the data structures to be managed. Moreover, the functionalities might be transformed into database functionality.

In current, productive systems, relational databases respectively models are still the most widely-used models, on which current infrastructure persistence is realized.

The work-in-progress focuses on the question, if graph databases with database supported functionalities might be a good alternative compared to current solutions on top of relational models.

This paper tries to outline a generic graph model as it can be used in ROR, to define requirements and framework conditions. It tries to summarize generic demands and to describe the query and functionality requirements that have to be satisfied by such databases. This paper presents basic ideas and the origin point of intended and starting database research projects and cooperation with universities in the next month and years.

Keywords

Railway infrastructure data, persistence of graph topology, railways operations research functionality, infrastructure database.

1. INTRODUCTION

The application field of railway operations research is a quite special field which is usually associated to other topics than graph databases or graph structured data and their management.

Usually railway operations research deals with topics like delay propagation, robustness of timetables, capacity of infrastructure, capacity allocation or evaluation of infrastructure modification effects. Such topics can be analyzed and answered on behalf of analytical, constructive or simulative approaches [5].

While the main focus of ROR activities usually lies on sufficient algorithms, formulas or modelling approaches the mostly unmentioned basic of all functionality is an infrastructure graph, which acts as the basis for running time computations, blocking or minimum headway time determination and the ability to select alternative routes, additional stops or to perform rescheduling operation.

Therefore, an essential but usually unconsidered component of all railway operations research activities is an infrastructure network graph, on which all functionalities are based.

The railway infrastructure network consists of rails, switches, crossings, buffer stops etc. and can mathematically be described as a (directed) graph, which is the most static part of a railway operations research project[1].

Based on this graph – the infrastructure graph – functionalities are defined and tasks are performed. The most elementary functionality is the determination of running times and the determination of infrastructure occupation but also the search for matching routes, alternative stop policies or the evaluation of infrastructure capacities based on timetables or queueing theory.

The following chapters try to introduce approaches to infrastructure graph modelling, existing exchange formats and perspectives onto such graphs.

A generic graph definition as a consensus of different views and approaches is derived and typical functionalities performed on such graphs are outlined.

The last chapter finally describes our work in progress and summarizes currently ongoing database research activities, primarily targeting a performant prototype and accompanying prove of concept of the approach described in this paper and their suitability with respect to manageability on behalf of graph databases.

[1] Usually several timetables and their robustness of delay behavior are evaluated for a given network infrastructure, therefore the infrastructure is considered as “most static” within a project.

2. DATA MODEL AND FUNCTIONALITY

To describe the requirements and functionalities a graph database that is tightly fitting the application field of railway operations research should satisfy, it is worth to take short looks into existing models, data exchange formats and systems, focusing on infrastructure models.

To introduce more aspects from railway operations research – not only the basic infrastructure network and graph-like topology – some typical tasks and questions are shortly described to allow a better understanding of required functionalities.

From that starting point it hopefully becomes clearer, what is expected from graph databases for this quite specific task, in functionality, data model and performance requirements.

2.1 Proprietary models

Proprietary data models for railway operations research were introduced decades ago. While old systems for timetabling support in the 80s used quite specific track and infrastructure models like sequences of elements for single or double track lines, specific configuration records to describe the characteristic (and track existences) of stations etc. the graph topology approach became popular in the 90s with the increased availability of personal computers and their performance.

It became clear, that graph models are the most flexible structures and best fitting representations of the real network, but computation power and the acceptance of computer based systems still had to grow.

Typical systems of this time use either tool specific, binary and size dense data and file formats or standard database models like the relational one to store and manage the network data persistently. While relational databases were considered to be performant, widely available and standardized such databases only support railway operations functionality in a quite limited manner. Databases are primarily used as persistence stores to guarantee ACID characteristics when working with network infrastructure data.

The functionality is usually implemented on top of the standard database system. The computation of e.g. possible routes cannot be implemented directly within the relational database domain[2]. So currently ROR functionality is performed on behalf of data loaded into main memory with corresponding performance and accessibility benefits but with the drawback, that database functionalities like transactional control is not available. Finally the evaluation of graph database based approaches for data persistency has to be compared against this "traditional" scenario: load from database and restore the network graph, perform functionalities within main memory, and probably store manipulated data back into the database in contrast to the directly performed functionalities within graph databases.

2.2 Graph models and UIC RailTopoModel

Even if the graph model is currently considered to be a sufficient and suitable approach to model network infrastructure, the content of these models differs, especially when considering different levels of granularity as Figure 1 illustrates.

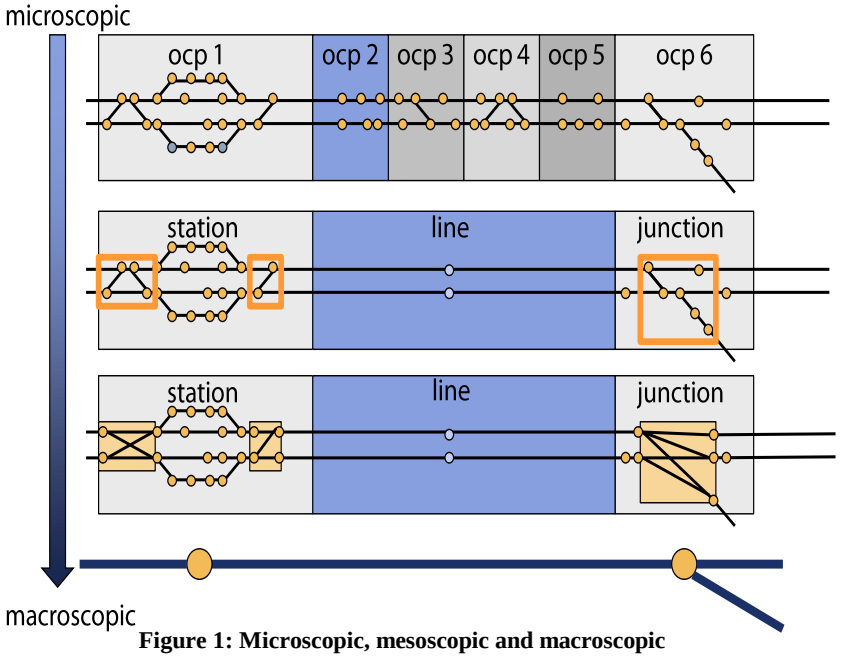


Figure 1: Microscopic, mesoscopic and macroscopic network infrastructure graphs and node aggregation into operational control points, stations, junctions and lines.

There are several more or less widely used approaches, philosophies and granularities used by different systems. Mostly, these models were implied by legacy systems, research prototypes or available data sources.

There are node weighted or edge weighted and attributed approaches, which both have advantages and disadvantages with respect to redundancies, performance or expressional strength.

There are microscopic and macroscopic models considering the network topology and network elements in varying detail depth. Microscopic models consider not only the track related topology but also signals, liberation equipment, curve radius, track gradients, tunnels, switches or stopping positions, balises, speed profiles etc., usually in precise of meters. Beside this, specific tools used for infrastructure planning might moreover contain much more elements and positioning precise.

Moreover, the application field a tool is designed for as well as the local technical requirements and circumstances determine the content of the infrastructure network graph[3].

[2] The relational model and SQL was extended by specific functionality, e.g. closure operators, but nevertheless these functionalities are not really used within current systems for different reasons like performance, portability or even availability within a specific RDBMS.

[3] In Germany there is e.g. a train protection system called LZB (lineare Zugbeeinflussung) which requires to model LZB areas, marker boards for area characteristics and which is not available in most other countries. The same is true for several other systems, which usually are country centered developments.

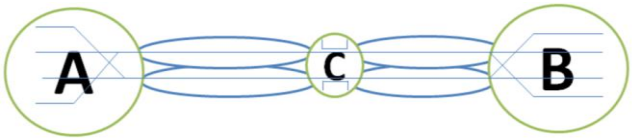


Figure 2: Micro-/macroscopic infrastructure modelling (UIC RailTopoModel).

Several country specific, national models and modelling approaches exists, e.g. in Germany, the DB Netz AG uses a node weighted graph model – the Spurplan – within their timetabling systems RUT-K [8] that defines a wide range of allowed elements and element instances to set up a network. The same is true for a Belgium specific approach to an infrastructure graph model, the INT graph [6].

To generalize and abstract the overall problem of infrastructure network graph modelling and its management within database systems, a scientific, generic model has to be used or to be derived.

One quite interesting project targeting such a generic infrastructure model is the RailTopoModel initiated by the UIC [1]. One of the ideas behind is to model network topologies for macroscopic as well as for microscopic approaches and to define mapping and transformation functionalities between different levels of granularity (Figure 2).

Consequently the UIC RailTopoModell is a (at least currently) promising approach to set up a generic network graph model that might cover a bright range of generic requirements. It therefore is one source of the overall graph model which should be implemented and supported by a graph database targeted by our ongoing work.

One crucial aspect which constantly causes problems with respect to generalization and universal usability and acceptance of such models are nationally affected (non-functional) requirements, e.g. a regulatory for clustering the network into operation control points (OCP), the aggregation of tracks within lines, separation of grids and intergrids and much more. Such classification criterion and requirements are often the background for a specific modelling and might be generally be described as graph clustering, coverage and overlapping problems (chapter 3.2).

2.3 Exchange Formats

Similar to graph models the exchange formats evolved. Besides "owning" a graph model it makes sense to define a sufficient exchange format for data fitting to this model.

While for the proprietary formats mentioned within section 2.1 usually binary file formats were used, nowadays data exchange formats are XML based, e.g. defined by XSD schemata.

The German Spurplan used by DB Netz AG implied the (company internal) XML-ISS standard for railway research operation tools. For more operation and planning centered systems other standards exists and are currently under development, e.g. within the PlanPro project [7].

The RailML project [2] is another example, where an international partnership tries to define an exchange format (not only for infrastructure) in a more or less generic and universally valid manner. Unfortunately, this approach again focusses on a quite specific model – a track oriented view – which contradicts the initially expressed universality. Moreover in practice, missing semantic specifications reduces the universal validity of exchange formats like RailML to a pairwise agreement and convention, which again strongly contradicts any standardization intention of this project.

3. GRAPH DATABASES

This paper wants to gain insight into ongoing work. This work focuses on graph databases and how such (new[4]) database approaches might be used in a beneficial manner to support, replace or extend the nowadays systems, their functionalities and performances.

The ongoing work focuses on research and evaluation activities and join-projects with universities and the determination of solutions which matches the application field requirements in a quite optimal manner.

In the following subsections requirements and demands are outlined, that have to be considered when designing and evaluating graph databases and their functionalities to be enabled to compare such rather new and alternative approaches to existing ones.

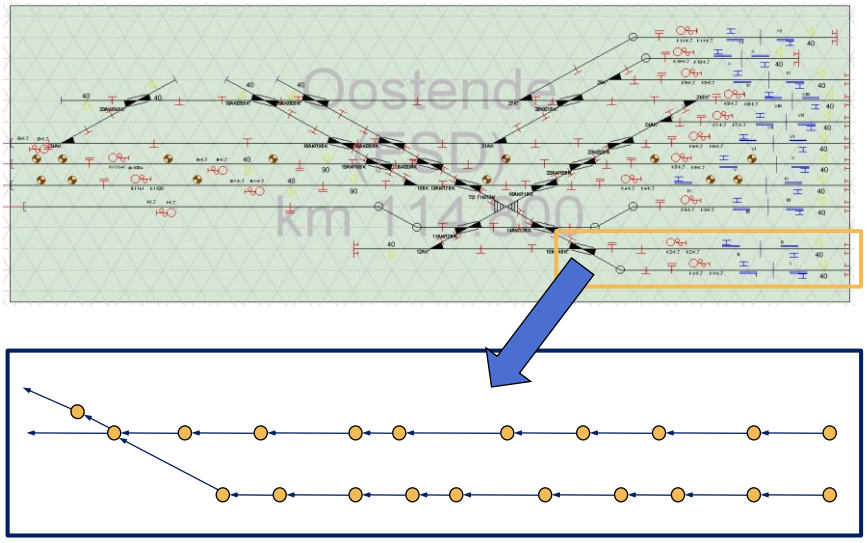


Figure 3: Rail network and network graphs.

3.1 Base Topology and requirements

As mentioned before, a generic graph model fits the network graph modelling requirements in a best way, similar to the mentioned UIC RailTopoModel (Figure 3). Additionally to this simple model, an infrastructure graph database must consider several more aspects outlined in the following subsections.

From our point of view, the core and elementary rail network topology should be modelled by a directed graph similar to the one proposed by the UIC RailTopoModel:

[4] At least within the application field considered.

- A network graph is a graph G=(N, E) where N is a set of nodes and E is a set of (directed) edges with $E \subseteq N \times N$.
- The directed cardinality $|n|=(s_e, s_l)$ of a node $n \in N$ is defined by the number s_e of edges entering n and the number s_l of edges leaving n.
- A node $n \in N$ is called an inner node if $|n|=(1,1)$ and edge node otherwise.
- The graph is considered to be a node weighted graph, where characteristic values, e.g. speeds allowed, changing gradients or signaling functionality is assigned to nodes.
- Track sections are paths throughout the graph starting and terminating at edge nodes with only inner nodes within the path. For a path $P=(n_1, .., n_m)$ of nodes, n_i is an inner node for $i=2 \ldots m-1$ and $(n_i, n_{i+1}) \in E$ for $i=1 \ldots m-1$. A track section contains at least two nodes (m>1).
- All inner nodes are attributed by direction validity, e.g. a node is valid for train running within the direction of the associated edge of in opposite direction (or both). This validity has to be considered by all functionalities like running or occupation time computation as well as routing and route evaluation.
- For all nodes of a track section $P=(n_1,\ldots,n_m)$ a layout position within a defined (from many possible) positioning system is given. This might be a GIS coordinate in case of GIS systems or the layout coordinates of a linearized or user friendly display of the network graph.
- For all nodes of a track section $P=(n_1,\ldots,n_m)$ a (relative) positioning information pos(n) is assigned with $pos(n_1)=0$, $pos(n_m)=1$ and $pos(n_i) \leq pos(n_{i+1})$ for $i=1 \ldots m-1$.
- The mileage of nodes respectively section elements is derived from location information (GIS/meters/etc.) assigned to the section start and end due to the positioning information.

Nodes might additionally be distinguished due to their semantic, for which area or length they are valid. Most nodes respectively corresponding infrastructure element are usually point elements, whose semantic is related to a specific point, e.g. a speed change, a stopping position (the "H"-board) or the location of a signal.

But nevertheless semantic might be extended to area and length validity, e.g. speed restriction zones, level crossings etc. The validity semantic is expressed by node attributes.

3.2 Topology Coverage and Clustering

Some of the most problematic issues towards a unified topology model are national rules and regulations as mentioned before. E.g. in Germany infrastructure elements are logically organized with operation control points as the top-most classification criteria. In other countries grids or inter-grids are the primary structuring criteria; sometimes a track line is the major criteria.

The topology graph model we consider for the intended graph database implementation tries to generalize all these approaches on behalf of graph coverages and graph node clustering:

- The cluster C of a network graph G=(N, E) is a graph $G_C=(N_C, E_C) \subseteq G$ such that $N_C \subseteq N$ and for all $n_1, n_2 \in N_C$ with $(n_1,n_2) \in E$, also $(n_1,n_2) \in E_C$ holds.
- A coverage $CV=\{C_1, \ldots C_m\}$ of a network graph G=(N, E) is a set of clusters of G.
- A total coverage $TC=\{C_1, \ldots C_m\}$ of a network graph G=(N, E) is a set of clusters of G such that $C_i=(N_i, E_i)$ for $i=1,\ldots,m$ and N_i, N_j $(i,j=1,\ldots,m, i \neq j)$ are disjunctive node sets whose union is N.

With this clustering, it is possible to define the varying logical orderings and classifications as mentioned before:

- The logical separation of a network graph G into operation control points is a coverage of G.
- A network graph G can be separated into grids and inter-grids. A grid-inter-grid-approach is a total coverage $TC=\{C_1, \ldots C_m\}$ (m>0) of G such that a cluster C_i is a grid, whenever there is a node $n \in C_i$ with $|n|=(x_{in}, x_{out})$ and $x_{in}>1$ or $x_{out}>1$ (switch or crossing), and an inter-grid otherwise. All edges from G not contained in TC always connect nodes from grids to nodes from inter-girds or wise versa.
- Lines $L=\{C_1, \ldots C_m\}$ (m>0) of a network graph G are a (not total) coverage of G where all nodes of each cluster C_i $(i=1 \ldots m)$ are part of at least one path within C_i.
- Power supply areas of a network graph G are areas within the corresponding network, where (electrical) power is supplied by one or more transformer substations. Therefore the power supply areas of G can be modelled as a (not total) coverage of G.

All examples stated before are examples of different logical clustering of the overall network graph and should illustrate the functionality which has mandatorily to be supported by database, especially the support of clustering and additional cluster constraints.

3.3 Interlocking Routes

Several existing infrastructure data models for railway networks concentrate on a quite limited view on the rail (and graph) topology as a primary (and only) modeling aspect, as e.g. RailML does until nowadays.

For railway operations research tools this view is not sufficient. Track related systems like railways basically rely on interlocking techniques and therefore this aspect has to be supported by models and consequently by databases as well.

A route of an infrastructure graph G is a path within G corresponding to the technical circumstances given by the

concrete settings of an interlocking station and its ability to control signals, switches and track accessibility.

So one additional requirement a graph database for infrastructure graph management has to fulfill is to support coverages representing routes and paths within the graph.

In contrast to "usual" routing and path finding functionality (which nevertheless is required but considered later on within the paper) specific route data has to be stored, because such routing information has to be enriched by application field specific attributes. Therefore it could be said that the graph database has to be able to manage attributed routing information. Such attributes might be information about the usability (electrification, axle weights, stopping positions offered etc.) that are available in addition to the pure infrastructure information, the classification of certain routes or the relevance for different train types respectively train classes[5]. With such route information several railway operations research functionalities are supported like computer based routing or rescheduling.

Routes typically start at one graph node and describe a path to another graph node. Such nodes can be signals, track ends/boundaries or even specific reference nodes[6].

3.4 Temporal Validity

One aspect typically not considered by infrastructure models is the spatio-temporal validity of the infrastructure network. Railway operations research functionality typically concentrates on a specific network graph, but this consideration is not necessarily true in any case.

Within timetabling periods there are more or less important changes somewhere in the network. Switches are added or removed, interlocking stations are extended or modified, tracks or even complete areas are closed for maintenance work etc.

Therefore it must be ensured, that a universally usable graph network considers temporal validities and retrieves network graphs and topologies depending on requested times respectively time periods.

So one essential question for this work in progress is how graph databases can be used to access different topologies changing over time and with which performance.

3.5 Routing and path finding

Last but not least another elementary functionality for the considered application field is the routing functionality as known from several similar application fields like route guidance and navigation systems.

The graph database has to offer this functionality on behalf of interlocking routes as described in section 3.3. In practice, queries for routes between two graph nodes have to consider the train characteristic and priority of the routes as well as the attributed routing information.

While this "plain" routing functionality is used e.g. for timetabling, the routing for railway operation simulation or analytical evaluation has to perform this search slightly different. Usually along the train run overtaking sections for the specific train have to be determined. Overtaking sections are areas of a network graph, where in practice no change of train order can be performed. The two ends of an overtaking section are characterized by the ability to change this train order, concretely to allow one train to overtake or to be overtaken by another train.

This is again determined by alternative routing selecting sufficient routes at the section ends which e.g. offer a sufficient stopping position and electrification.

3.6 Summary

In this chapter, requirements against a graph database to handle network graphs for railway operations research purpose were mentioned and introduced. Roughly speaking, the most important are:

- Support of (node weighted) graphs with different positioning and layout systems.
- Temporal validities and the ability to retrieve time specific graph topologies.
- Clustering and coverage of network graphs to ensure generality.
- Management of interlocking routes and routing functionality on top of these routes.

With this functional "specification" the evaluation of graph databases as a sufficient persistent storage system can be started.

4. WORK IN PROGRESS/NEXT STEPS

The handling of infrastructure network data and ensuring its persistency is an elementary component of nowadays railway operations research tools.

The current legacy system landscape usually uses "traditional, relational" approaches to store and manage such data. There is an obvious mismatch between the relation and set oriented paradigms of these databases and the topology, semantic and structure of graphs which are a "natural" model for railway network infrastructure.

If functionalities like simulation, timetabling, capacity evaluation or other tasks from the application field of railway operations research should be performed, they are currently performed on in-memory data structures which had been created from the relational data sets while loading them.

The existence of graph databases obviously closes the gap between the database model and the one of the specific application domain. A central question for commercial tools is if it is worth to shift to new, less evaluated approaches like graph databases.

For this reason we work on an evaluation of the performance and functional capabilities of graph databases in comparison to "traditional, relational" approaches.

At the Workshop on Querying Graph Structured Data 2015 (GraphQ 2015) we expect to be able to present and show first results, provide an insight into current settings of this ongoing

[5] Often there are tracks and lines dedicated e.g. to freight or passenger trains, even if both types are physically comparable (same gauge, same locomotive etc.) but routes are more relevant for on type than for the other. Therefore a route might have a high priority for freight trains and a very low one for passenger trains.

[6] This corresponds to the interlocking paradigms, where exactly one origin and one target have to be defined before the interlocking process – e.g. setting up switches and signals – is accepted and started.

evaluation or at least discuss aspects of this problem field at the workshop itself.

The evaluation is intended to start soon as a joint-project between VIA Consulting & Development GmbH as initiator of this work, different students and universities specialized on graph database techniques and the railway infrastructure manager DB Netz AG in Germany.

It is expected to provide graph and route data from existing systems with expected graph sizes up to several hundred-thousand nodes and thousands of interlocking routes for the whole German railway network. In this way it will be ensured, that the research and evaluation work is related to practical conditions and requirements.

The next steps from the current stage of the ongoing work are the definition and selection of different evaluation and comparison scenarios and modelling approaches with respect to specific databases. As a basis of comparison, a relational database as it is currently used in practice is considered.

5. REFERENCES/LITERATURE

[1] UIC – International Union of Railways, UIC RailTopoModel: Railway Network Description – A conceptual model to describe a railway network, http://documents.railml.org/science/280714_uic_railtopomodel_rc2.pdf

[2] railML.org Initiative, http://www.railml.org

[3] Hansen, I. A.; Pachl, J. (eds.): Railway Timetabling & Operations. Analysis - Modelling - Optimisation - Simulation - Performance Evaluation. Eurailpress 2014, ISBN 978-3777104621

[4] Kuckelberg, A.; Seybold, B.: "Adaptive Rule-Based Infrastructure Modelling" – In: Proc. of the 5th International Seminar on Railway Operations Modelling and Analysis, Copenhagen, 13.-15.05.2013.

[5] Janecek, D.; Kuckelberg, A.; Nießen, N.: "Kapazitätsermittlung von Eisenbahnknoten und Strecken" – In: Eisenbahntechnische Rundschau (ETR) 61 (2012) 10, pp. 30-36.

[6] INT – Graph Model Design, Infrabel internal working paper, INT RFT.04

[7] PlanPro, Durchgängige Datenhaltung der Leit- und Sicherungstechnik (LST) von der Planung bis zum Bestand (in German), DB Netze, http://fahrweg.dbnetze.com/fahrweg-de/start/technik/innovationen/planpro

[8] K. Wölfle, RUT-K – Computer-Aided Train-Path Management, Paris, 13.10.05, Talk at UIC

Linked Web Data Management (LWDM)

Devis Bianchini (Università di Brescia),
Valeria De Antonellis (Università di Brescia),
Roberto De Virgilio (Università Roma Tre)

An Extensible Framework for Query Optimization on TripleT-Based RDF Stores

Bart G. J. Wolff
Eindhoven University of Technology
b.g.j.wolff@alumnus.tue.nl

George H. L. Fletcher
Eindhoven University of Technology
g.h.l.fletcher@tue.nl

James J. Lu
Emory University
jlu@emory.edu

ABSTRACT

The RDF data model is a key technology in the Linked Data vision. Given its graph structure, even relatively simple RDF queries often involve a large number of joins. Join evaluation poses a significant performance challenge on all state-of-the-art RDF engines. TripleT is a novel RDF index data structure, demonstrated to be competitive with the current state-of-the-art for join processing. Query optimization on TripleT, however, has not been systematically studied up to this point. In this paper we investigate how the use of *(i)* heuristics and *(ii)* data statistics can contribute towards a more intelligent way of generating query plans over TripleT-based RDF stores. We propose a generic framework for query optimization, and show through an extensive empirical study that our framework consistently produces efficient query evaluation plans.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Query processing

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

RDF, SPARQL, TripleT, indexing, query processing

1. INTRODUCTION

Motivation. The goal of the Linked Data vision is to create a global "web of data": an infrastructure for machine-readable semantics for data on the web [9]. This vision aims to make data from a wide variety of sources available under the standardized RDF data model, allowing for this data to be shared across different domains using web standards.

As adoption of the linked data vision grows, data stores have to be able to deal with increasingly large datasets. This poses a scalability problem, both for storage and indexing, as well as for query evaluation. By its triple-centric graph-like nature, even the most basic RDF queries involve a large number of (self-)joins, which pose a significant performance challenge on state-of-the-art RDF database engines. At present, real-world RDF datasets can involve hundreds of millions or even billions of triples, making it challenging to offer interactive query response time.

When compared to relational database technology, RDF stores are a relatively new concept. A number of RDF stores exist, one of them being the Three-way Triple Tree data structure (TripleT) [6], which features a value-based, role-free indexing scheme, unique among the current state-of-the-art. Research has shown this approach to be competitive with, and often at an advantage to, alternative indexing schemes, in terms of both storage and query evaluation costs [6]. However, query optimization on TripleT-based RDF stores has not been systematically studied up to this point.

Our contributions. In this paper, we present our experiences and results of a comprehensive investigation of query optimization on TripleT [18]. In particular, we study how the use of *(i)* heuristics and *(ii)* dataset statistics can contribute towards a more effective generation of query plans, minimizing query execution time over the TripleT RDF store. Our aim here is to understand the effectiveness of various parts of the heuristics that influence query plan generation. These query plans are tailored to (and evaluated on) our implementation of the TripleT store, which we also describe in this paper.

The novelties of our work include an extensible generic rule-based framework for query optimization over TripleT, and an extensive empirical study into the effectiveness of proposed rules in generating optimized query plans. Furthermore, the complete experimental framework, including both disk-based storage and the query processing pipeline, is available as open-source code for further study.[1]

Our proposed optimization framework, together with a few key heuristics rules, is able to consistently produce efficient query plans for a wide variety of query types and datasets. In comparing heuristics-based and statistics-based rules, our aim was to understand the benefit offered by the use of statistics. Our study shows that not only do rules using statistics in general offer little performance improvements compared to heuristics-only rules, but also that a purely heuristics-based approach may exhibit an order of magnitude reduction in evaluation costs in certain situa-

[1] https://github.com/b-w/TripleT

tions. These observations support those of Tsialiamanis et al. in their study of heuristics-based optimization of RDF queries [17].

2. BACKGROUND

Definitions. We present the basics of data and queries. Further details can be found in [1, 18]. Let $\mathbb{U}$ be a set of URIs and $\mathbb{L}$ be a set of literals, such that $\mathbb{U} \cap \mathbb{L} = \emptyset$. Then we define an *RDF triple* as an element $(s, p, o) \in \mathbb{U} \times \mathbb{U} \times (\mathbb{U} \cup \mathbb{L})$. We define an *RDF dataset* (or, alternatively, an *RDF graph*), denoted $\mathbb{T}$, as a set of $n \geq 0$ RDF triples: $\mathbb{T} = \{t_1, t_2, \cdots, t_n\}$.

At the core of many RDF query languages such as SPARQL lies the concept of Basic Graph Patterns (BGPs) [1]. A BGP is a conjunction of Simple Access Patterns (SAPs), where each SAP is a triple consisting of some combination of fixed values (atoms) and unfixed values (variables). Formally, let $\mathbb{A} = \mathbb{U} \cup \mathbb{L}$ be a set of atoms, and let $\mathbb{V}$ be a set of variables, such that $\mathbb{A} \cap \mathbb{V} = \emptyset$. Then we define an *SAP* as a triple $S = (s, p, o) \in (\mathbb{U} \cup \mathbb{V}) \times (\mathbb{U} \cup \mathbb{V}) \times (\mathbb{A} \cup \mathbb{V})$. We then define a *BGP* as a conjunction of SAPs: $P = S_1 \wedge S_2 \wedge \cdots \wedge S_n$, for some $n \geq 0$. Equivalently, P may be regarded as the set $\{S_1, S_2, \ldots, S_n\}$.

A *binding* for BGP P is a function B from the variables occurring in P to the set of atoms $\mathbb{A}$. We define the *application* of binding B to P, denoted $B(P)$, as the set of triples resulting from replacing every occurrence of every variable v in P with $B(v)$. Finally, the result of querying graph $\mathbb{T}$ with P, denoted $P(\mathbb{T})$, is the set of all bindings $\mathbb{B}$ such that for each $B \in \mathbb{B}$ it holds that $B(P) \subseteq \mathbb{T}$.

We indicate variables with the prefix '?.' As a small example, the BGP $P = (jan, knows, ?p) \wedge (?p, fanOf, mozart)$ on graph

$$\mathbb{T} = \{(jan, knows, sue), (jan, knows, tim), (sue, fanOf, mozart)\}$$

evaluates to $P(\mathbb{T}) = \{\langle ?p : sue \rangle\}$.

Related work. Numerous RDF stores and indexes have been developed in recent years. Notable examples include Virtuoso [5], RDF-3X [15], and Sesame.[2] We refer the reader to Luo et al. [12] for a thorough survey of storage and indexing solutions for massive RDF datasets.

The study of query optimization is as old as the study of database systems. On the topic of RDF, Neumann and Weikum [14, 15] address some of the scalability problems that arise when processing join queries on very large RDF graphs. Optimizations for BGPs using statistics for selectivity estimation are discussed by Stocker et al. [16], while Tsialiamanis et al. present a number of heuristics for BGP static analysis and optimization [17]. Various studies have been made on techniques for selectivity and cardinality estimation using precomputed information over RDF datasets [7, 10, 13, 15].

TripleT was originally proposed by Fletcher and Beck [6]. Value-based indexing for join processing was also shown to be effective in the context of relational and complex-object databases (e.g., [2, 3, 4]). Some prior work exists featuring TripleT. The performance of different join algorithms on the TripleT index was investigated by Li [11]. An extension of TripleT was used by Haffmans and Fletcher [8] as physical representation of data used for a proposed RDFS entailment algorithm, where it was shown to be good candidate for RDFS data storage.

[2] `http://www.openrdf.org`

3. A THREE-WAY TRIPLE TREE

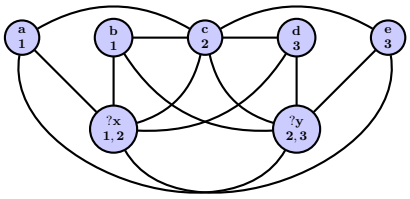
The primary novelty of the TripleT index is that it is built over the individual atoms in a dataset, rather than over complete triple patterns. TripleT uses a number of *buckets* that store the actual triples in the dataset. Each bucket stores all the (s, p, o) triples in the dataset, ordered on some permutation of $\{s, p, o\}$. For instance, an SOP-bucket would (conceptually) store the triples sorted first on *subject*, then on *object*, and lastly on *predicate*. The possible bucket orderings are thus SPO, SOP, PSO, POS, OSP, and OPS, though in our implementation we limit ourselves to using SOP-, PSO-, and OSP-buckets only. The remaining permutations, SPO, POS, and OPS, are symmetrical and are not considered in our investigation. Of important note is that each bucket does not contain the triple part (s, p, or o) that corresponds to its primary sort order. This information is implied by the index and does not need to be repeated.

There is one entry in the index for each unique atom in the dataset. This entry contains a number of pointers to triple ranges in each of the bucket files. For instance, the index entry for an atom a might contain a pointer to range $[x \cdots y]$ in the SOP-bucket, meaning that in this bucket, which is sorted on *subject*, triples from position x to position y contain the value a in their *subject* position. Similarly, the same entry might contain pointers to triple ranges in the PSO- and OSP-buckets that contain a in the *predicate*- and *object* positions, respectively.

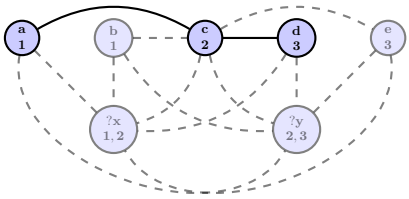
The index supports retrieval of bindings matching a single SAP. The sort ordering of a bucket determines how suitable it is for retrieving triples matching a particular SAP. An SOP-bucket, for example, would be well suited for retrieving triples matching $(a, ?x, ?y)$, but would be inefficient at retrieving triples matching $(?x, a, ?y)$. For the latter case the bucket ordering implies the entire bucket needs to be read in order to find all possible matches, while for the former case the index entry for a directly points to the range in the bucket where any matches must be contained.

Implementation details. Our work is based on our own open-source implementation of the TripleT RDF store [18]; we briefly highlight salient features here and refer the reader to the full report for further details and design rationale. A single TripleT database is stored on disk across eight different physical files. We use a dictionary to translate between "friendly" representations and internal representations of atoms. This dictionary is stored in two BerkeleyDB[3] hash databases. The TripleT index is stored in a single BerkeleyDB hash database. There is one entry for each unique atom in the dataset. Each bucket belonging to a TripleT database is stored in its own separate file. There are three buckets for each database. The bucket files themselves are flat binary files containing sequences of triples. Each bucket contains all triples belonging to the dataset, although the files themselves contain only the parts of each triple that are not already present in the index. The statistics of a TripleT database are stored in a BerkeleyDB hash database. The

[3] `http://www.oracle.com/technetwork/database/database-technologies/berkeleydb`



(a) The atom collapse C_{P_1}



(b) A join graph J_{P_1}

Figure 1: Example graphs for the BGP $P_1 = (a, b, ?x)_1 \wedge (?x, c, ?y)_2 \wedge (?y, d, e)_3$

statistical database contains information for estimating output sizes for single SAPs or joins between two SAPs, as well as some summarizing statistics [18].

4. QUERY OPTIMIZATION

Our framework for generating optimized query plans for BGPs over the TripleT index consists of a generic algorithm in which a number of decision points can be manipulated by a given set of rules. In this section, P is defined as a BGP, consisting of k SAPs $(s_1, p_1, o_1), \cdots, (s_k, p_k, o_k)$, denoted $S_1, \cdots, S_k$, resp.

4.1 Atom collapses

We define the *atom collapse* C_P of P as the undirected edge-labeled graph with the atoms and variables of P as nodes, and edges to indicate there is a shared variable between the SAPs associated with nodes.

Formally, the set of nodes consists of *atom*-nodes and *variable*-nodes. For each SAP $S_i \in P$ we have an *atom*-node $(a, \{S_i\})$ for each unique atom $a \in S_i$. We have a *variable*-node (v, P_v) for each unique variable $v \in P$ with $P_v \subseteq P$ being the set of SAPs which contain v. For the special case of SAPs that do not contain any atoms, we have a special atom node $(a_0, \{S_i\})$, where a_0 is a nil-atom.

Let (x, X) and (y, Y) be nodes in the collapse graph. In the set of edges we have an undirected edge $(x, X) - (y, Y)$ with label L if and only if there exists some variable v such that $v \in S_X, S_X \in X$ and $v \in S_Y, S_Y \in Y$ and $S_X \neq S_Y$. Label L consists of a set of (S_i, S_j, v, p_i, p_j) tuples, where there are tuples for every variable v such that $v \in S_i, S_i \in X$ and $v \in S_j, S_j \in Y$ with $S_i \neq S_j$ and with p_i and p_j denoting the positions (s, p, or o) that variable v has in S_i and S_j respectively. Note that one variable can occur in multiple tuples in one label, as long as each tuple as a whole is unique within L.

As an example, Figure 1(a) shows the atom collapse for the BGP $P_1 = (a, b, ?x)_1 \wedge (?x, c, ?y)_2 \wedge (?y, d, e)_3$. For ease of reference, we have numbered the SAPs. Here we have an edge between $(e, \{S_3\})$ and $(?x, \{S_1, S_2\})$, due to the shared variable $?y$ of S_3 and S_2, but no edge between $(e, \{S_3\})$ and $(a, \{S_1\})$, since S_3 and S_1 do not share a variable.

4.2 Join graphs

We define a *join graph* J_P of P as a subgraph of atom collapse C_P, with the nodes from J_P being a subset of the atom-nodes from C_P such that for each SAP $S_i \in P$ there is exactly one node $(a, \{S_i\})$ in J_P, and the edges from J_P being the same as those from C_P. The nodes from the join graph are known as *seed nodes* as they represent the physical access path for each of the SAPs, which is the TripleT bucket used for retrieving them.

Formally, the set of nodes in J_P is defined as

$$NODES(J_P) \subseteq \{(x, X) \in NODES(C_P) \mid x \text{ is an atom}\}$$

such that $\forall S_i \in P.(\exists!(x, X) \in NODES(J_P).(X = \{S_i\}))$. The set of edges is defined as

$$EDGES(J_P) = \{((x, X) - (y, Y) : L) \in EDGES(C_P) \mid (x, X), (y, Y) \in NODES(J_P)\}.$$

Note that P can have multiple distinct, valid join graphs. Figure 1(b) shows a possible join graph for P_1.

4.3 Decision points

The goal of the optimizer is to generate a query plan over the TripleT engine, where a *query plan* is a tree consisting of physical operators as internal nodes and index bucket scans as leaves, for evaluating P. Our optimization framework takes the BGP P as input, first computes its atom collapse C_P, then a join graph J_P, and lastly produces a query plan Q_P for J_P.

The computation features four distinct *decision points*, and we follow a rule-based approach for dealing with them. For computing the join graph there is one such point: (1) deciding which seed node to select from the collapse graph C_P of P. The computation of J_P proceeds by selecting seeds until all SAPs of P are accounted for. For computing the query plan Q_P from J_P there are three: (2) deciding which join edge to select from J_P; (3) deciding which join type to apply for the selected join edge; and, (4) deciding what scan to select for a given SAP. All four decision points are resolved by a number of configurable rules that are separate from the rest of the algorithm.

The rules. Decision points 1 and 4 are identical (both involve selecting a seed for an SAP) and can be resolved by two possible rules. Rule **seed-1** (S1) selects one preferred seed for each distinct SAP in the input set based on the positions of the atoms in the SAP, following the ordering $s \succ o \succ p$. The intuition here is that subjects are more selective than objects, which in turn are more selective than predicates. Rule **seed-2** (S2) does the same but prioritizes the atoms in the SAP according to their selectivity as indicated by dataset statistics.

Decision point 2 has the greatest influence on query response time, as it determines the order of joins in the query plan. It is resolved by five rules. Rule **join-1** (J1) aims to prioritize those joins for which it is possible to do a merge join, which is intuitively cheaper to perform given

the TripleT index organization. Rule **join-2** (J2) prioritizes joins involving the most selective SAPs, where selectivity is determined through the ordering $(s,p,o) \succ (s,?,o) \succ (s,p,?) \succ (?,p,o) \succ (s,?,?) \succ (?,?,o) \succ (?,p,?) \succ (?,?,?)$. Here, s,p,o denote arbitrary atoms and ? denotes an arbitrary variable; and, $S \succ T$ indicates pattern S is more selective than pattern T. Rule **join-3** (J3) aims to prioritize joins between two SAPs that have the most selective positioning of join variables, following the ordering $s \bowtie p \succ o \bowtie p \succ s \bowtie o \succ s \bowtie s \succ o \bowtie o \succ p \bowtie p$. Rule **join-4** (J4) prioritizes joins between SAPs which feature a literal value (e.g. "Sue") in one of its positions, over those featuring only URIs (e.g. "http://example.org/Sue"). The intuition behind rules J2-J4 generalizes our intuition behind S1. Rule **join-5** (J5) prioritizes joins between SAPs which, according to the statistics database over the dataset, produce the smallest intermediate result sets.

Decision point 3 is resolved by a fixed heuristic: whenever it is possible to do a merge join (i.e. the left- and right input sets involved in the join are both sorted on their shared join variables), we do so; if not, we perform a hash join instead.

The rules for resolving decision points 1, 2, and 4 can be used in any configuration (i.e. which rules are and are not used, and in which order are they applied). Hence at each decision point there is a variable, ordered list of rules R which act as filters and which are applied in sequence on the set of options in order to arrive at a final choice. Each rule $r \in R$ reduces the set of options I to a set of options $I' \subseteq I$ through filter step $I \xrightarrow{r} I'$. Any items in I' are then said to be equivalent under r. Similarly, an ordered list of rules R performs filtering step $I \xrightarrow{R} I'$, with any items remaining in I' being called equivalent under R.

5. EXPERIMENTAL STUDY

The goal of our experiments is to gather evidence relevant to answering the following questions:

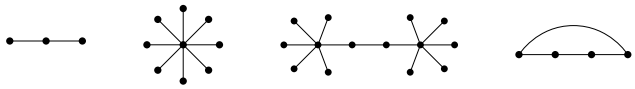
1. How effective is each individual rule for generating optimized query plans?
2. How effective are combinations of rules for generating optimized query plans?
3. Does the order in which rules are applied matter?
4. What is the impact of using statistics?
5. How do our optimization techniques perform under different types of queries?
6. How do our optimization techniques perform under different kinds of datasets?

These questions can be divided into four sections: *(a)* the value of rules, *(b)* the value of statistics, *(c)* the difference between queries, and *(d)* the difference between datasets.

The value of rules. As discussed in Section 4.3, our optimization techniques make use of a number of different rules, which are applied in a certain sequence when we arrive at a decision point where a choice needs to be made. Most of them work based on some heuristic. One would not expect each rule to be just as effective as the next; in fact, such would be a highly surprising outcome. Instead, one would expect there to be noticeable differences in the effectiveness of individual rules. One would also expect that certain combinations of rules will prove to be highly effective, more so than what the sum of the parts might suggest. The order in which rules are applied at a decision point would be expected to matter to a certain degree but be less important than which rules are and are not used.

The value of statistics. Although we have described only two rules in Section 4.3 which make use of statistics, their purpose is the same as that of all of our heuristics-based rules: to minimize intermediate result sizes produced during query plan execution. Of course, the use of these statistics-based rules comes at a cost: a full statistics database needs to be computed and maintained over the dataset.

The difference between query types. There are several different types of queries we use in our experiments. As our datasets are essentially graphs and our queries are graph patterns, it's easy to visualize them as such. In Figure 2 the four common query shapes that our queries are based around are shown, where a query's SAPs are represented by nodes which are connected if they share a variable.



(a) Chain query (b) Star query (c) Star-chain query (d) Loop query

Figure 2: Different query shapes we study

Aside from their **shape**, other variables we study are query **size** (in number of SAPs), and query **selectivity**. The influence of query size on execution time is difficult to predict. On one hand, more SAPs means more joins; on the other, more SAPs can also mean higher selectivity which can be exploited by the plan generator. As for query selectivity, a query which features more atoms in more selective positions in its SAPs generally produces a smaller result set. Again, selective SAPs in a query can be favorably exploited by the plan generator.

The collection of concrete queries used in our experiments – covering the full range of combinations of shape, size, and selectivity – is detailed in Wolff [18].

The difference between datasets. The test data we have used comes from three different sources, covering both real and synthetic data: DBpedia,[4] SP^{2}Bench,[5] and UniProt.[6] From each source we have obtained three different datasets: one 100.000 (100K) triples dataset, one 1.000.000 (1M) triples dataset, and one 10.000.000 (10M) triples dataset. For all datasets, the 100K set is a strict subset of the 1M set, which in turn is a strict subset of the 10M set.

Plan of study. In our experiments we primarily compare different rule sets to each other. Hence, the composition of the rule set is the main variable in each experimental run. We define a *run* in our experiment as testing one particular rule set on every dataset using every available query. Here, *testing* some rule set x on dataset y using query z is com-

[4] http://dbpedia.org/

[5] http://dbis.informatik.uni-freiburg.de/forschung/projekte/SP2B/

[6] http://www.uniprot.org/

Table 1: Results overview on 1M datasets: execution time (ms)

	SP²Bench 1M					UniProt 1M				
	Chain	Star	Star-chain	Selective	Non-selective	Chain	Star	Star-chain	Selective	Non-selective
Run A-3	9975,5	6806,2	5070,5	2376,5	12191,6	1151,9	24328,1	25884,0	17512,9	16729,7
Run A-4	18121,6	6293,4	198520,0	55936,2	92687,1	1194,7	39200,0	2173,3	1472,3	26906,4
Run D-2	12173,2	6876,5	5195,6	3018,5	13145,0	1158,0	28490,1	3620,1	4010,5	18168,2

prised of: opening the TripleT database for dataset y; telling the query plan generator to use rule set x; feeding query z to the database; enumerating and immediately discarding the query results; closing the database. Each run is tested five times, each time "cold", i.e. without preserving any caches between tests, with average costs reported.

The runs we have performed are detailed in Table 2, where a number indicates that a particular rule was used in that run, the number itself indicating the order (a lower number denoting a higher priority). Each of the runs has a particular purpose: the **A-runs** are designed to test the heuristics rules against the statistics rules; the **B-runs** aim to get a sense of the value of the individual heuristics rules; the **C-runs** focus on the ordering of rules; the **D-runs** are used to test different subset combinations of rules.

Table 2: Runs and their rule sets

	Rules						
Runs	S1	S2	J1	J2	J3	J4	J5
A-1	1		1	2	3	4	
A-2		1					1
A-3	1	2	1	2	3	4	5
A-4	2	1	2	3	4	5	1
B-1	1			1	2	3	
B-2	1		1		2	3	
B-3	1		1	2		3	
B-4	1		1	2	3		
C-1	1		4	3	2	1	
C-2	1		3	2	1	4	
C-3	1		2	1	4	3	
D-1		1	1	3			2
D-2	1		2	1			
D-3		1		1	2		3

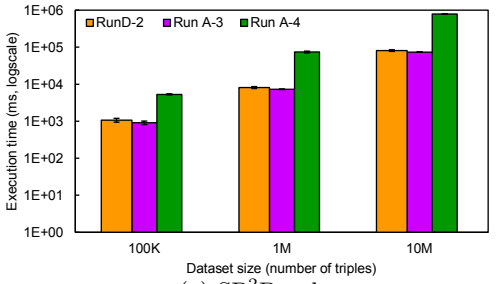
5.1 Empirical results

In the interest of space, we highlight only a few of our main observations here and refer the reader to Wolff [18] for a detailed presentation and analysis of all results.

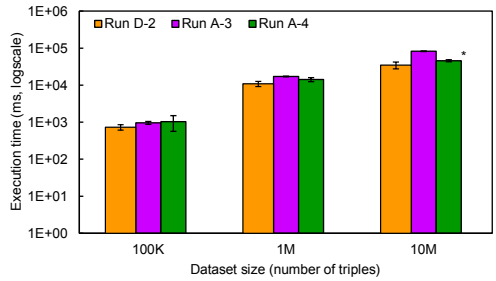
The A-runs. These runs were designed to test the performance of the heuristics-based rules (runs A-1 and A-3) against the statistics-based rules (runs A-2 and A-4).

We focus our discussion on A-3 and A-4 as illustrations of these two groups. Table 1 presents results for the SP²Bench and UniProt 1M datasets. We visualize results over all dataset sizes in Figure 3, where we plot the average query execution time over all queries, along with the average standard deviation. These results confirm that the heuristics-based A-3 rule set offers better performance overall.

For the SP²Bench datasets, the heuristics-based A-3 is most effective, with execution time of A-4 reaching up to an order of magnitude higher. The DBPedia datasets showed performance similar to that of SP²Bench. On UniProt the differences are less pronounced, though we note that A-4 failed to scale up to the 10M set for the star-chain query.



(a) SP²Bench



(b) UniProt

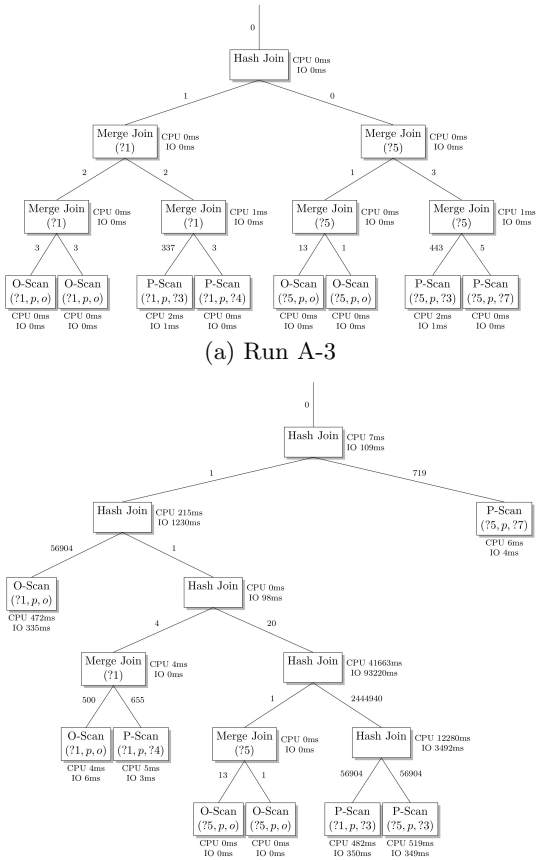
Figure 3: Results overview of runs A-3, A-4, and D-2

Here, a combination of primarily heuristics rules with statistics as back-up worked best, as seen with A-3.

On closer inspection, the general cause for A-4's performance is that its plans feature more hash joins than those produced by A-3. Performing a hash join can result in a significant amount of intermediate result materialization, whereas this is not the case with the merge join because it can take advantage of the fact that the input streams are guaranteed to be sorted. This is illustrated by the actual plans generated by A-3 and A-4 as presented in Figure 4, for the following small selective star-chain SP²Bench query:

(?1, http://www.w3.org/1999/02/22-rdf-syntax-ns#type, http://localhost/vocabulary/bench/Article),
(?1, http://purl.org/dc/elements/1.1/creator, http://localhost/persons/Paul_Erdoes),
(?1, http://swrc.ontoware.org/ontology#journal, ?3)
(?1, http://purl.org/dc/terms/references, ?4),
(?5, http://www.w3.org/1999/02/22-rdf-syntax-ns#type, http://localhost/vocabulary/bench/Article),
(?5, http://purl.org/dc/elements/1.1/creator, Dell_Kosel),
(?5, http://swrc.ontoware.org/ontology#journal, ?3),
(?5, http://purl.org/dc/terms/references, ?7)

The B- and C-runs. These runs were designed to get a sense of the value for each individual heuristics rule and to measure the importance of rule ordering, respectively. In



(a) Run A-3

(b) Run A-4

Figure 4: Plans generated for a small selective star-chain query on the SP^{2}Bench 1M dataset

short, we observed in these experiments that the J1 rule is the single most important heuristics rule: configurations that gave it a lower priority often failed to scale up to the 10M datasets. Aside from that there appeared to be no clear winner in individual rules or ordering of the rules.

The D-runs. In these runs we look at configurations which use a limited subset of rules. These experiments showed that minimal rule sets are quite stable and effective in performance. As a point of comparison with A-3 and A-4, we present the results for D-2 in Table 1 and Figure 3. Here we see that even this very limited rule set is always competitive with both A-3 and A-4. The D-2 run appears to provide the best, consistent performance over all three datasets, and preferring the J2 rule (selectivity) over the J1 rule (merge-joins) is one of the few configurations to perform well on the UniProt star- and star-chain queries, which proved to be some of the most difficult queries we have tested. This performance by the D-2 run is somewhat surprising, as this configuration consists of only three heuristics rules (one seed rule, two join rules), and does not use statistics at all.

5.2 Discussion

We have seen through an extensive empirical evaluation how the value of individual rules used by the plan generator can vary greatly. Especially the merge join prioritization rule, which is given preference in A-3, appears to be invaluable for the generation of efficient query plans. As a heuristic, there are of course practical scenarios which violate this good behavior of merge join prioritization. In our experiments, we experienced this only in the case of UniProt star-chain queries. Indeed, here the performance generally improved when this rule was given a lower priority, as in D-2, where the selectivity prioritization rule proved to have the largest positive impact. The benefits of the three remaining heuristics rules are similar. In particular, in their absence, the impact on query response time is roughly the same.

In general, we have observed that the impact of statistics and the statistical prioritization join rule is measurable but limited. When used alone or as the primary join rule, the statistics rule produces query plans significantly worse than those produced by the heuristics rules, as evidenced by run A-4. This suggests that the value of statistics rules is found mostly in a supporting role.

In summary, we recommend the D-2 rule configuration for general use, as it is a purely heuristic and minimal approach which delivers excellent results across the board. Overall, our findings corroborate results obtained by Tsialiamanis et al. [17], where a heuristics-based planner for SPARQL queries is shown to be competitive with the cost-based approach taken in the state of the art RDF-3X store [15].

6. CONCLUDING REMARKS

In this paper we have presented results of a study of query optimization on TripleT-based RDF stores. We have proposed a query optimization framework that takes the shape of a generic, rule-based algorithm. We also proposed a number of heuristic and statistical rules for use by this algorithm.

We have evaluated this framework in an extensive series of experiments. These experiments have shown that a small number of relatively simple heuristics can consistently produce efficient evaluation plans for a wide variety of queries and datasets. We have also seen that while statistics do add value, the value is minimal, and not within reasonable proportion to the costs involved in constructing and maintaining statistical data structures over massive graphs.

A number of interesting avenues for future work remain open. A study of runtime optimization strategies in our framework, such as sideways information passing [14], and further sophisticated join-ordering [7] strategies are both naturally rich areas for exploration. We have also encountered various challenges with using statistics for query optimization. Additional work in this area would be interesting, and may yet help our optimization framework produce even more efficient query plans.

Acknowledgments. We thank Antonio Badia, Paul De Bra, and Herman Haverkort for their helpful comments.

References

[1] M. Arenas, C. Gutierrez, and J. Pérez. Foundations of RDF databases. In *Reasoning Web*, pages 158–204, Brixen-Bressanone, 2009.

[2] B. C. Desai. Performance of a composite attribute and join index. *IEEE TSE*, 15(2):142–152, 1989.

[3] A. Deshpande and D. Van Gucht. A storage structure for nested relational databases. In *Nested Relations and Complex Objects, LNCS 361*, pages 69–83. Springer, 1987.

[4] A. Deshpande and D. Van Gucht. An implementation for nested relational databases. In *VLDB*, pages 76–87, Los Angeles, 1988.

[5] O. Erling and I. Mikhailov. RDF support in the Virtuoso DBMS. In T. Pellegrini et al, editor, *Networked Knowledge - Networked Media*, pages 7–24. Springer, 2009.
[6] G. H. L. Fletcher and P. W. Beck. Scalable indexing of RDF graphs for efficient join processing. In *CIKM*, pages 1513–1516, Hong Kong, 2009.
[7] A. Gubichev and T. Neumann. Exploiting the query structure for efficient join ordering in SPARQL queries. In *EDBT*, pages 439–450, Athens, Greece, 2014.
[8] W. J. Haffmans and G. H. L. Fletcher. Efficient RDFS entailment in external memory. In *SWWS*, pages 464–473, 2011.
[9] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.
[10] H. Huang and C. Liu. Estimating selectivity for joined RDF triple patterns. In *CIKM*, pages 1435–1444, Glasgow, 2011.
[11] K. Li. Cost analysis of joins in RDF query processing using the TripleT index. Master's thesis, Emory University, 2009.
[12] Y. Luo, F. Picalausa, G. H. L. Fletcher, J. Hidders, and S. Vansummeren. Storing and indexing massive RDF datasets. In R. De Virgilio et al, editor, *Semantic Search over the Web*, pages 31–60. Springer, 2012.
[13] T. Neumann and G. Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *ICDE*, pages 984–994, Hannover, Germany, 2011.
[14] T. Neumann and G. Weikum. Scalable join processing on very large RDF graphs. In *SIGMOD*, pages 627–640, Providence, Rhode Island, USA, 2009.
[15] T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113, 2010.
[16] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds. SPARQL basic graph pattern optimization using selectivity estimation. In *WWW*, pages 595–604, Beijing, 2008.
[17] P. Tsialiamanis, L. Sidirourgos, I. Fundulaki, V. Christophides, and P. Boncz. Heuristics-based query optimisation for SPARQL. In *EDBT*, pages 324–335, Berlin, 2012.
[18] B. G. J. Wolff. A framework for query optimization on value-based RDF indexes. Master's thesis, Eindhoven University of Technology, 2013. http://alexandria.tue.nl/extra1/afstversl/wsk-i/wolff2013.pdf.

Towards an RDF validation language based on Regular Expression derivatives

Jose Emilio Labra Gayo*
University of Oviedo
Spain

Eric Prud'hommeaux
W3c
Stata Center, MIT

Sławek Staworko
LINKS, INRIA & CNRS
University of Lille, France

Harold Solbrig
Mayo Clinic, College of Medicine
Rochester, MN, USA

ABSTRACT

There is a growing interest in the validation of RDF based solutions where one can express the topology of an RDF graph using some schema language that can check if RDF documents comply with it.

Shape Expressions have been proposed as a simple, intuitive language that can be used to describe expected graph patterns and to validate RDF graphs against those patterns. The syntax and semantics of Shape Expressions are designed to be familiar to users of regular expressions.

In this paper, we propose an implementation of Shape Expressions inspired by the regular expression derivatives but adapted to RDF graphs.

1. INTRODUCTION

The industry need to describe and validate conformance of RDF instance data with some *schema* has motivated a W3C Workshop [24] and the chartering of W3C RDF Data Shapes Working Group.[1] Here, a *schema* defines an RDF graph structure where a node has expected properties with defined cardinalities, connecting to literal values or other described nodes.

As currently defined, RDF Schema [2] and OWL [22] are widely recognized as being insufficient to fulfil this task, leading to proposals like the RDF vocabulary *Resource Shapes*[2] and the *Shape Expressions*[3] language.

The operational semantics of Shape Expressions has been presented at [23] and the complexity and expressiveness of the language has been studied at [1]. A Shape Expression is a labelled pattern that describes RDF nodes using a syntax inspired by regular expressions.

Example 1. The following shape expression describes `Person` shapes as nodes that have one property `foaf:age` with values of type `xsd:int`, one or more properties `foaf:name` with values of type `xsd:string` and zero or more properties `foaf:knows` with values of shape `Person`.

```
<Person> {
  foaf:age   xsd:integer
, foaf:name  xsd:string+
, foaf:knows @<Person>*
}
```

It is possible to automatically check which nodes comply with the declared shapes in an RDF Graph.

Example 2. The nodes `:john` and `:bob` in the following graph have shape `Person` while the node `:mary` does not have that shape.

```
:john foaf:age 23;
      foaf:name "John";
      foaf:knows :bob .

:bob  foaf:age 34;
      foaf:name "Bob", "Robert" .

:mary foaf:age 50, 65 .
```

Shape expressions can be used to describe and validate the contents of linked data portals [16] and there are several implementations and online validation tools like ShEx Workbench[4] and RDFShape[5].

Regular expressions are a well-known formalism to describe the shape of sequences of characters. They have also been employed to describe the shape of XML trees and form the theoretical basis of RelaxNG. In 1964, Janusz Brzozowski proposed a method for directly implementing a regular expression recognizer based on regular expression derivatives [3]. In this paper, we adapt the derivatives approach to RDF Graph based recognizers. We define regular

*Corresponding author
[1] http://www.w3.org/2014/data-shapes/
[2] http://www.w3.org/Submission/shapes/
[3] http://www.w3.org/Submission/shex-defn/
[4] http://www.w3.org/2013/ShEx/FancyShExDemo
[5] http://rdfshape.weso.es

shape expressions, which form the basis of the Shape Expressions language, and present the algorithm that can be used to check if an RDF node has a given Shape. The algorithm has been implemented and the performance results are better than a backtracking implementation.

2. PRELIMINARIES

Given a set S, we denote S^* as the powerset of S, $\{\}$ denotes the empty set and $\{a_1, \ldots, a_n\}$ denotes a set with elements $a_1, \ldots, a_n$. The singleton set $\{a\}$ will be simplified as a.

Let V_s = vocabulary of subjects, V_p = vocabulary of predicates and V_o = vocabulary of objects. In RDF, if we define $\mathcal{I}$ as the set of IRIs, $\mathcal{B}$ as the set of blank nodes and $\mathcal{L}$ as the set of literals, we have $V_s = \mathcal{I} \cup \mathcal{B}$, $V_p = \mathcal{I}$ and $V_o = \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$.

A graph Σ is defined as a set of triples $\langle s, p, o\rangle$ such that $s \in V_s$, $p \in V_p$ and $o \in V_o$. Σ* denotes all possible graphs. The expression $t \rtimes t_s$ represents the addition of triple t to a graph t_s. Given two graphs g_1 and g_2, $g_1 \oplus g_2$ denotes the union of g_1 and g_2. Notice that we are using union of RDF graphs instead of merging. Union of two RDF graphs preserves the identity of blank nodes shared between graphs while merging does not [11].

The decomposition of a graph g is defined as the set $\{(g_1, g_2) | g_1 \oplus g_2 = g\}$. The decomposition of a graph with n triples is an exponential operation that generates a graph with 2^n pairs of graphs that can be obtained by calculating the powerset of g and pairing each element with its complement.

Example 3. Let $g = \{\langle n, a, 1\rangle, \langle n, b, 1\rangle, \langle n, b, 2\rangle\}$, the decomposition of g is:

$$
\begin{array}{ll}
\{ & (\{\},\{\langle n,a,1\rangle,\langle n,b,1\rangle,\langle n,b,2\rangle\}),\\
 & (\{\langle n,a,1\rangle\},\{\langle n,b,1\rangle,\langle n,b,2\rangle\}),\\
 & (\{\langle n,b,1\rangle\},\{\langle n,a,1\rangle,\langle n,b,2\rangle\}),\\
 & (\{\langle n,b,2\rangle\},\{\langle n,a,1\rangle,\langle n,b,1\rangle\}),\\
 & (\{\langle n,a,1\rangle,\langle n,b,1\rangle\},\{\langle n,b,2\rangle\}),\\
 & (\{\langle n,a,1\rangle,\langle n,b,2\rangle\},\{\langle n,b,1\rangle\}),\\
 & (\{\langle n,b,1\rangle,\langle n,b,2\rangle\},\{\langle n,a,1\rangle\}),\\
 & (\{\langle n,a,1\rangle,\langle n,b,1\rangle,\langle n,b,2\rangle\},\{\}),\\
\}
\end{array}
$$

We define the shape of a node n in a graph g, Σ_n^g as the set of triples related to n in graph g. It is formed by all the triples of the form $\langle n, p, o\rangle \in g$. We define Σ^* as all possible shapes that a node n can have.

3. WHY NOT SPARQL?

Shape Expressions have been proposed as a high level, intuitive language to validate RDF. This problem can also be partially solved using SPARQL queries [14] which leverage on the whole expressiveness of the SPARQL query language.

The main issue of SPARQL queries is that they can become unwieldy and difficult to generate, manage and debug by hand.

Example 4. A SPARQL query that can express part of example 1 is:

```
ASK {
{ SELECT ?Person {
  ?Person foaf:age ?o .
} GROUP BY ?Person HAVING (COUNT(*)=1)}
{ SELECT ?Person {
  ?Person foaf:age ?o .
FILTER ( isLiteral(?o) &&
         datatype(?o) = xsd:integer )
} GROUP BY ?Person HAVING (COUNT(*)=1)}
{ SELECT ?Person (COUNT(*) AS ?Person_c0) {
  ?Person foaf:name ?o .
} GROUP BY ?Person HAVING (COUNT(*)>=1)}
{ SELECT ?Person (COUNT(*) AS ?Person_c1) {
  ?Person foaf:name ?o .
  FILTER (isLiteral(?o) &&
  datatype(?o) = xsd:string)
} GROUP BY ?Person HAVING (COUNT(*)>=1)}
  FILTER (?Person_c0 = ?Person_c1)
{ {
{ SELECT ?Person (COUNT(*) AS ?Person_c2){
  ?Person foaf:knows ?o .
} GROUP BY ?Person}
{ SELECT ?Person (COUNT(*) AS ?Person_c3){
  ?Person foaf:knows ?o .
  FILTER ((isIRI(?o) || isBlank(?o)))
  }
 GROUP BY ?Person HAVING (COUNT(*) >= 1)}
 FILTER (?Person_c2 = ?Person_c3)
} UNION  { SELECT ?Person {
   OPTIONAL { ?Person foaf:knows ?o }
   FILTER (!bound(?o))
}}}}
```

Representing RDF validation constraints as SPARQL queries is not practical for large data portals and there is a need for a higher level, declarative language with a more intuitive semantics.

Apart from that, the previous example is not completely right as it has omitted the recursive definition where it should validate that the values of `foaf:knows` all have the shape of `Person`. Trying to represent recursive definitions in SPARQL is not possible in general.[6] From our point of view SPARQL can be used as a lower level language for constraint validation in the sense that Shape Expressions can be mapped to SPARQL queries. In fact, one of our implementation of Shape Expressions is already able to generate those SPARQL queries from Shape Expressions.

4. INTRODUCING REGULAR SHAPE EXPRESSIONS

In this section we define Regular Shape Expressions as a simplified language based on the whole Shape Expressions language. This language will be used as the basis for our implementations. A regular shape expression E defines the triples related with a given node in a graph. Although the concept presented in this paper is focused on RDF graphs, we consider that these definitions can be applied to describe the topology of other graph structures.

Given three non-empty sets V_s, V_p, V_o and $v_s \subseteq V_s$, $v_p \subseteq V_p$ and $v_o \subseteq V_o$, the abstract syntax of regular shape expressions (E) over V_s, V_p, V_o is:

E, F	::=	$\emptyset$	empty, no shape
	|	ε	empty set of triples
	|	$\_ \xrightarrow{v_p} v_o$	arc with predicate $p \in v_p$ and object $o \in v_o$
	|	$E*$	Kleene closure (0 or more E)
	|	$E \parallel F$	And (unordered concatenation)
	|	$E \mid F$	Alternative

We do not provide the concatenation operator from string based regular expressions because the arcs in a graph are not ordered. The

[6]This particular query could be represented using zero-length paths as proposed by Joshua Taylor in StackOverflow `http://goo.gl/uMoXBQ`

And operator ($\|$) for unordered concatenation appears in [1] and is similar to interleave or shuffle [6, 10] although in the case of graphs and regular shape expressions there is no ordered concatenation operator.

The operators $E+$ (one or more) and $E?$ (optional) can be defined as:

$$\begin{aligned} E+ &= E \parallel E* \\ E? &= E \mid \varepsilon \end{aligned}$$

The Shape Expressions language also contains a range operator $E\{m,n\}$ which represents between m and n repetitions of E. It can be defined as:

$$E\{m,n\} = \begin{cases} E\{m,n-1\}|E & \text{if } m < n \\ E\{m-1,n-1\} \parallel E & \text{if } m = n > 0 \\ \varepsilon & \text{if } m = n = 0 \end{cases}$$

Example 5. The regular shape expression

$$\llcorner \xrightarrow{a} 1 \parallel \llcorner \xrightarrow{b} \{1,2\}*$$

declares a shape that contains one arc with predicate a and value 1, and one or more arcs with predicate b and values 1 or 2.

Example 6. We can consider `xsd:int` and `xsd:string` as subsets of $\mathcal{L}$ (the set of Literals) in RDF, so we can define the shape:

$$\llcorner \xrightarrow{\texttt{foaf:age}} \texttt{xsd:integer} \parallel (\llcorner \xrightarrow{\texttt{foaf:name}} \texttt{xsd:string})+$$

that declares nodes that must have an arc with predicate `foaf:age` and value in `xsd:int` and one or more arcs with predicate `foaf:name` and value in `xsd:string`. In ShEx notation it can be represented as:

```
<Example> {
  foaf:age  xsd:integer
, foaf:name xsd:string+
}
```

Given a node n, the shape of a regular shape expression e with respect to n, denoted as $\mathcal{S}_n[\![e]\!]$ is the set of graphs $\mathcal{S}_n[\![e]\!] \subseteq \Sigma^*$ generated by the following rules:

$$\begin{aligned} \mathcal{S}_n[\![\emptyset]\!] &= \emptyset \\ \mathcal{S}_n[\![\varepsilon]\!] &= \{\} \\ \mathcal{S}_n[\![\llcorner \xrightarrow{v_p} v_o]\!] &= \{\langle n,p,o\rangle | p \in v_p \text{ and } o \in v_o\} \\ \mathcal{S}_n[\![e*]\!] &= \{\} \cup \mathcal{S}_n[\![e \parallel e*]\!] \\ \mathcal{S}_n[\![e_1 \parallel e_2]\!] &= \{t_1 \cup t_2 |\, t_1 \in \mathcal{S}_n[\![e_1]\!] \text{ and } t_2 \in \mathcal{S}_n[\![e_2]\!]\} \\ \mathcal{S}_n[\![e_1 \mid e_2]\!] &= \mathcal{S}_n[\![e_1]\!] \cup \mathcal{S}_n[\![e_2]\!] \end{aligned}$$

Example 7. Let $e = \llcorner \xrightarrow{a} 1 \parallel \llcorner \xrightarrow{b} \{1,2\}*$, then

$$\begin{aligned} \mathcal{S}_n[\![e]\!] = \{&\{\langle n,a,1\rangle\}, \\ &\{\langle n,a,1\rangle, \langle n,b,1\rangle\}, \\ &\{\langle n,a,1\rangle, \langle n,b,2\rangle\}, \\ &\{\langle n,a,1\rangle, \langle n,b,1\rangle, \langle n,b,2\rangle\}\} \end{aligned}$$

For any expression x, the operators $\|$, $|$, ε and $\emptyset$ obey the following simplification rules:

$$\begin{aligned} \emptyset \mid x &= x \\ x \mid \emptyset &= x \\ \emptyset \parallel x &= \emptyset \\ x \parallel \emptyset &= \emptyset \\ \varepsilon \parallel x &= x \\ x \parallel \varepsilon &= x \end{aligned}$$

5. MATCHING REGULAR SHAPE EXPRESSIONS

Given a regular shape expression e and a node n in a graph g, we want to determine if Σ_n^g (the subgraph formed by the triples related with n) matches the regular shape expression $\mathcal{S}_n[\![e]\!]$, i.e. we want to determine if $\Sigma_n^g \in \mathcal{S}_n[\![e]\!]$.

The semantics of Regular Shape Expressions is defined by a relation $e \simeq \Sigma_n^g$ (e matches Σ_n^g) which can be expressed using axioms and inference rules [23]. Figure 1 presents the operational semantics of Regular Shape Expressions. Those rules can be directly implemented using backtracking.

Example 8. Let $e = \llcorner \xrightarrow{a} 1 \parallel \llcorner \xrightarrow{b} \{1,2\}*$ and a graph g where $\Sigma_n^g = \{\langle n,a,1\rangle, \langle n,b,1\rangle, \langle n,b,2\rangle\}$, a trace of the matching algorithm is represented in figure 2. Notice that we have to decompose the matching graph g in all the pairs of graphs g_1 and g_2 whose union give g. In this case, the decomposition returns all the pairs depicted in example 3.

As can be seen, a naïve implementation of Regular Shape expression matching using backtracking leads to exponential growth and has poor performance.

6. REGULAR SHAPE EXPRESSION DERIVATIVES

The derivative of a shape $\mathcal{S}_n(E) \subseteq \Sigma^*$ with respect to a triple $t \in \Sigma$ is a shape that includes only the remaining triples that when appended to t will become $\mathcal{S}_n(E)$.

Definition 1. The derivative of a Shape $\mathcal{S}_n(E) \subseteq \Sigma^*$ with respect to a triple $t \in \Sigma$ is defined as $\partial_t(\mathcal{S}_n(E)) = \{t_s | t \rtimes t_s \in \mathcal{S}_n(E)\}$

We need a helper function $\nu : E \to Bool$ (also called nullable) that checks if a regular shape expression can match the empty graph.

$$\nu(E) = \begin{cases} \texttt{true} & \text{if } E \text{ matches the empty graph} \\ \texttt{false} & \text{otherwise} \end{cases}$$

$$\begin{aligned} \nu(\emptyset) &= \texttt{false} \\ \nu(\varepsilon) &= \texttt{true} \\ \nu(\llcorner \xrightarrow{v_p} v_o) &= \texttt{false} \\ \nu(e*) &= \texttt{true} \\ \nu(e_1 \parallel e_2) &= \nu(e_1) \wedge \nu(e_2) \\ \nu(e_1 \mid e_2) &= \nu(e_1) \vee \nu(e_2) \end{aligned}$$

The following rules, inspired from Brzozowski [3], compute the derivative of a regular shape expression with respect to a triple t.

$$Or_1 \frac{r_1 \simeq g}{r_1|r_2 \simeq g} \qquad Or_2 \frac{r_2 \simeq g}{r_1|r_2 \simeq g}$$

$$And \frac{r_1 \simeq g_1 \quad r_2 \simeq g_2}{r_1 \parallel r_2 \simeq g_1 \oplus g_2}$$

$$Empty \frac{}{\varepsilon \simeq \{\}}$$

$$Star_1 \frac{}{r* \simeq \{\}} \qquad Star_2 \frac{r \simeq g_1 \quad r* \simeq g_2}{r* \simeq g_1 \oplus g_2}$$

$$Arc \frac{p \in v_p \quad o \in v_o}{\text{␣} \xrightarrow{v_p} v_o \simeq \langle s, p, o \rangle}$$

Figure 1: Inference rules for Shape expression rules

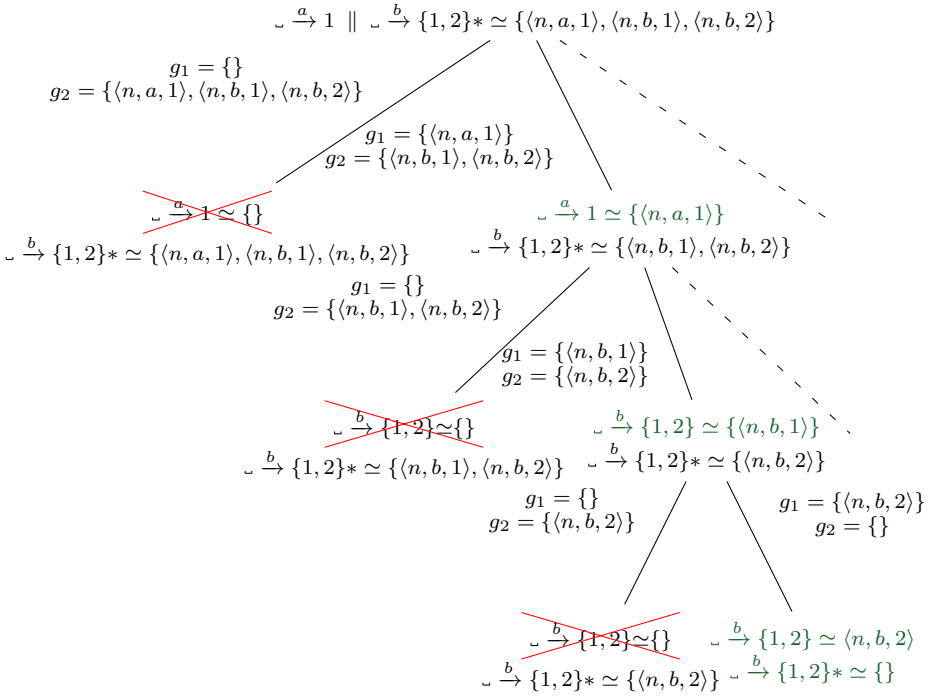


Figure 2: Regular Shape Expression matching using backtracking

$$\partial_t(\emptyset) = \emptyset$$
$$\partial_t(\varepsilon) = \emptyset$$
$$\partial_{\langle s,p,o\rangle}(\_ \xrightarrow{v_p} v_o) = \begin{cases} \varepsilon & \text{if } p \in v_p \text{ and } o \in v_o \\ \emptyset & \text{otherwise} \end{cases}$$
$$\partial_t(e*) = \partial_t(e) \parallel e*$$
$$\partial_t(e_1 \parallel e_2) = \partial_t(e_1) \parallel e_2 \mid \partial_t(e_2) \parallel e_1$$
$$\partial_t(e_1 \mid e_2) = \partial_t(e_1) \mid \partial_t(e_2)$$

Example 9. Let $e = \_ \xrightarrow{a} 1 \parallel \_ \xrightarrow{b} \{1,2\}*$, the derivative of e with respect to $\langle n, a, 1\rangle$ is $\_ \xrightarrow{b} \{1,2\}*$. A trace of the derivatives calculation can be:

$$\begin{aligned}
& \partial_{\langle n,a,1\rangle}(_ \xrightarrow{a} 1 \parallel _ \xrightarrow{b} \{1,2\}*) \\
= \; & \partial_{\langle n,a,1\rangle}(_ \xrightarrow{a} 1) \parallel _ \xrightarrow{b} \{1,2\}* \\
\mid \; & \partial_{\langle n,a,1\rangle}(_ \xrightarrow{b} \{1,2\}*) \parallel _ \xrightarrow{a} 1 \\
= \; & \varepsilon \parallel _ \xrightarrow{b} \{1,2\}* \\
\mid \; & \partial_{\langle n,a,1\rangle}(_ \xrightarrow{b} \{1,2\}) \parallel _ \xrightarrow{b} \{1,2\}* \parallel _ \xrightarrow{a} 1 \\
= \; & _ \xrightarrow{b} \{1,2\}* \\
\mid \; & \emptyset \parallel _ \xrightarrow{b} \{1,2\}* \parallel _ \xrightarrow{a} 1 \\
= \; & _ \xrightarrow{b} \{1,2\}* \\
\mid \; & \emptyset \parallel _ \xrightarrow{a} 1 \\
= \; & _ \xrightarrow{b} \{1,2\}* \mid \emptyset \\
= \; & _ \xrightarrow{b} \{1,2\}*
\end{aligned}$$

Notice that the derivative of a Regular Shape Expression can grow in its size.

Example 10. The regular shape expression $e = (\_ \xrightarrow{a} \{1,2\} | \_ \xrightarrow{b} \{1,2\})*$,checks that there are the number of arcs with predicate a and values in $\{1,2\}$ and arcs with predicate b and values in $\{1,2\}$ is the same. The derivative of e with respect to $\langle n, a, 1\rangle$ is $\_ \xrightarrow{b} \{1,2\} \parallel (\_ \xrightarrow{a} \{1,2\} | \_ \xrightarrow{b} \{1,2\})*$. Notice that it grows because once it finds an arc with predicate a, it needs to find another arc with predicate b and continue with the rest of the graph.

The rules can be extended to graphs (sets of triples) as follows:

$$\begin{aligned}
\partial_{\{\}}(e) &= e \\
\partial_{t \rtimes t_s}(e) &= \partial_{t_s}(\partial_t(e))
\end{aligned}$$

7. MATCHING USING DERIVATIVES

For any graph g, we have that $\Sigma_n^g \in \mathcal{S}_n[\![e]\!]$ if, and only if, $\varepsilon \in \mathcal{S}_n[\![\partial_{\Sigma_n^g}(e)]\!]$ which is true when $\nu(\partial_{\Sigma_n^g}(e)) = \texttt{true}$. We can express the algorithm in terms of the relation $e \simeq \Sigma_n^g$ defined as the smallest relation satisfying:

$$\begin{aligned}
e \simeq \{\} &\Leftrightarrow \nu(e) \\
e \simeq t \rtimes t_s &\Leftrightarrow \partial_t(e) \simeq t_s
\end{aligned}$$

It is straightforward to show that $e \simeq \Sigma_n^g$ if, and only if, $\Sigma_n^g \in \mathcal{S}_n[\![e]\!]$.

Notice that when a regular shape expression matches a set of triples, we compute the derivative for each of the triples in the set.

Example 11. Let $e = \_ \xrightarrow{a} 1 \parallel \_ \xrightarrow{b} \{1,2\}*$ and $\Sigma_n^g = \{\langle n,a,1\rangle, \langle n,b,1\rangle, \langle n,b,2\rangle\}$, the matching algorithm proceeds as:

$$\begin{aligned}
& _ \xrightarrow{a} 1 \parallel _ \xrightarrow{b} \{1,2\}* \simeq \{\langle n,a,1\rangle, \langle n,b,1\rangle, \langle n,b,2\rangle\} \\
\Leftrightarrow \; & \partial_{\langle n,a,1\rangle}(_ \xrightarrow{a} 1 \parallel _ \xrightarrow{b} \{1,2\}*) \simeq \{\langle n,b,1\rangle, \langle n,b,2\rangle\} \\
\Leftrightarrow \; & _ \xrightarrow{b} \{1,2\}* \simeq \{\langle n,b,1\rangle, \langle n,b,2\rangle\} \\
\Leftrightarrow \; & \partial_{\langle n,b,1\rangle}(_ \xrightarrow{b} \{1,2\}*) \simeq \{\langle n,b,2\rangle\} \\
\Leftrightarrow \; & _ \xrightarrow{b} \{1,2\}* \simeq \{\langle n,b,2\rangle\} \\
\Leftrightarrow \; & \partial_{\langle n,b,2\rangle}(_ \xrightarrow{b} \{1,2\}*) \simeq \{\} \\
\Leftrightarrow \; & _ \xrightarrow{b} \{1,2\}* \simeq \{\} \\
\Leftrightarrow \; & \nu(_ \xrightarrow{b} \{1,2\}*) \\
\Leftrightarrow \; & \texttt{true}
\end{aligned}$$

As can be seen the derivatives algorithm takes a linear approach where it is consuming a triple in each step and calculating the corresponding derivative of the regular shape expression. The algorithm does not need to decompose the graph or to do backtracking. The main complexity of the algorithm comes from the process of calculating and representing derivatives of shape expressions.

Example 12. Let $e = \_ \xrightarrow{a} 1 \parallel \_ \xrightarrow{b} \{1,2\}*$ and $\Sigma_n^g = \{\langle n,a,1\rangle, \langle n,a,2\rangle, \langle n,b,1\rangle\}$, the matching algorithm proceeds as:

$$\begin{aligned}
& _ \xrightarrow{a} 1 \parallel _ \xrightarrow{b} \{1,2\}* \simeq \{\langle n,a,1\rangle, \langle n,a,2\rangle, \langle n,b,1\rangle\} \\
\Leftrightarrow \; & \partial_{\langle n,a,1\rangle}(_ \xrightarrow{a} 1 \parallel _ \xrightarrow{b} \{1,2\}*) \simeq \{\langle n,a,2\rangle, \langle n,b,1\rangle\} \\
\Leftrightarrow \; & _ \xrightarrow{b} \{1,2\}* \simeq \{\langle n,a,2\rangle, \langle n,b,1\rangle\} \\
\Leftrightarrow \; & \partial_{\langle n,a,2\rangle}(_ \xrightarrow{b} \{1,2\}*) \simeq \{\langle n,b,1\rangle\} \\
\Leftrightarrow \; & \emptyset \simeq \{\langle n,b,1\rangle\} \\
\Leftrightarrow \; & \texttt{false}
\end{aligned}$$

8. SHAPE EXPRESSION SCHEMAS

In this section, we extend the regular shape expressions language to include labels for shape expressions. We assume a finite set of labels Λ.

A Shape Expression Schema is a tuple (Λ, δ) where δ is a shape definition function that maps labels to regular shape expressions over $V_s \cup \Lambda, V_p \cup \Lambda, V_o \cup \Lambda$. Typically, we present a schema as a collection of rules of the form $\lambda \longmapsto e$ where $\lambda \in \Lambda$ and $e \in E$

Example 13. Let Λ={p}, we can define the following Shape Expression Schema: The regular shape expression

$$\begin{aligned}
p \; \longmapsto \; & _ \xrightarrow{a} 1 \\
\parallel \; & _ \xrightarrow{b} \{1,2\} + \\
\parallel \; & _ \xrightarrow{c} p*
\end{aligned}$$

declares a schema where nodes of shape p contain an arc with predicate a and value 1, one or more arcs with predicate b and values 1 or 2, and zero or more arcs with predicate c and values of shape p. Notice that shape expression schemas can contain recursive references.

Example 14. Let Λ={person}, we can define the following

Shape Expression Schema which corresponds to example 1

$$\texttt{person} \longmapsto \_ \xrightarrow{\texttt{foaf:age}} \texttt{xsd:int}$$
$$\| \_ \xrightarrow{\texttt{foaf:name}} \texttt{xsd:string+}$$
$$\| \_ \xrightarrow{\texttt{foaf:knows}} \texttt{person*}$$

A shape typing is a mapping from nodes in a graph to labels. Given a graph and a regular shape schema, we define a type inference algorithm which assigns a shape typing to the nodes in the graph. The expression $\Gamma \vdash n \simeq_s s$ represents the shape typings generated when matching a node n with a shape s in the context Γ.

The context contains the current typing which can be accessed through $\Gamma.typing$. The expression $\Gamma\{n \to t\}$ means the addition of type t to n in context Γ. The semantic definition of $\simeq_s$ is depicted in Figure 3.

We define the following definitions on shape typings:

$\odot$	=	Empty typing
$n \to s : \tau$	=	Add shape type s to node n in typing t
$\tau_1 \uplus \tau_2$	=	Combine typings τ_1 and τ_2

The operational semantics presented in figure 1 can be extended to handle shape typings. The definitions are presented in figure 4. As can be seen the definitions are straightforward. The main novelty is the semantics of arcs which have been divided in two cases. Arc_{type} handles the case where the shape expression contains a value set, while Arc_{ref} handles the case where the shape expression contains a reference to a label. In that case, the object is matched against the shape expression associated with that label.

In order to adapt the inference rules to employ the derivatives algorithm, we modify the derivative function $\partial_t(e, \Gamma)$ to take a new parameter Γ that represents the typing context and to return a pair (e', τ) where $e \in E$ represents the derivative and τ represents the resulting typing. The new definition is:

$$\partial_t(\emptyset, \Gamma) = (\emptyset, \odot)$$
$$\partial_t(\varepsilon, \Gamma) = (\emptyset, \odot)$$
$$\partial_{\langle s,p,o\rangle}(\_ \xrightarrow{v_p} v_o, \Gamma) = \begin{cases} (\varepsilon, \Gamma.\tau) & \text{if } p \in v_p \text{ and } o \in v_o \\ (\emptyset, \odot) & \text{otherwise} \end{cases}$$
$$\partial_{\langle s,p,o\rangle}(\_ \xrightarrow{v_p} l, \Gamma) = \begin{cases} (\varepsilon, \tau) & \text{if } \Gamma\{o \to l\} \vdash \delta(l) \simeq_s \Sigma_o^g \rightsquigarrow \tau \\ (\emptyset, \odot) & \text{otherwise} \end{cases}$$
$$\partial_t(e*) = \textbf{let } (e', \tau) = \partial_t(e, \Gamma) \textbf{ in } e' \ \| \ e*$$
$$\partial_t(e_1 \ \| \ e_2) = \textbf{let } (e_1', \tau_1) = \partial_t(e_1, \Gamma), (e_2', \tau_2) = \partial_t(e_2, \Gamma) \textbf{ in } (e_1' \ \| \ e_2 \ | \ e_2' \ \| \ e_1, \tau_1 \uplus \tau_2)$$
$$\partial_t(e_1 \mid e_2) = \textbf{let } (e_1', \tau_1) = \partial_t(e_1, \Gamma), (e_2', \tau_2) = \partial_t(e_2, \Gamma) \textbf{ in } (e_1' \mid e_2', \tau_1 \uplus \tau_2)$$

The algorithm presented in this paper has been implemented in Scala[7] and Haskell[8]. The Scala implementation contains several extensions like reverse arcs, relations, negations, etc. that have been omitted in this paper for brevity while the Haskell prototype follows the simplified definitions presented here. Comparing the performance between the backtracking and the derivatives approach, we noticed that the latter obtains better results than the former.

[7] `http://labra.github.io/ShExcala/`
[8] `http://labra.github.io/Haws/`

Although the theoretical complexity of Shape Expression validation, which has been characterized in [1], remains the same, the derivatives algorithm behaves much better than the backtracking one. Further work needs to be done to check if we can identify a subset of the language with better complexity results while being expressive enough. In particular, the *Single Occurrence Regular Bag Expressions* subset defined in that paper offers a tractable language which could be expressive enough. In the future we are planning to adapt our implementation to that subset and study its performance behaviour in practice.

9. RELATED WORK

Regular expression derivatives where introduced by Brzozowski in 1964[3] and were used for string based recognizers of regular expressions. In 1999, Joe English proposed the use of derivatives for XML validation [9]. That idea was taken by James Clark to implement RelaxNG [4]. An updated presentation of regular expression derivatives is presented in [21] where the authors describe how to handle large character sets (Unicode). Our presentation follows the notations used in that paper adapted to regular shape expressions. With regards to the implementation, we took some inspiration by the Haskell implementation of a W3C XML Schema regular expression matcher maintained by Uwe Schmidt [26] which contains a definition for the interleave operator. There has also been some recent work applying regular expression derivatives to submatching [28] and parsing [17] and comparing it with the more traditional approach to regular expression matching based on NFA [7].

The main inspiration for Shape Expressions has been RelaxNG [30], a Schema language for XML that offers a good trade-off between expressiveness and validation efficiency. The semantics of RelaxNG has also been expressed using inference rules in the specification document [20] and is based on tree grammars [19]. Inspired by that specification, we presented the semantics of Shape Expressions using type inference rules in [23]. Our first prototype implementation of Shape Expressions in Haskell[9] employed a direct translation of the inference rules using a backtracking monad transformer. We consider that the equational reasoning presentation of the algorithm can be used to proof its correctness using an inductive representation of RDF graphs [15].

Besides Shape Expressions, there are several approaches that have been proposed to validate RDF Graphs which can be roughly classified as: inference based, SPARQL-based and grammar-based approaches.

OWL based approaches try to adapt RDF Schema or OWL to express validation semantics. However, using Open World and Non-unique name assumption limits validation possibilities. [5, 29, 18] propose the use of OWL expressions with a Closed World Assumption to express integrity constraints.

SPARQL-based approaches use the SPARQL Query Langugage to express the validation constraints. SPARQL has much more expressiveness than Shape Expressions and can even be used to validate numerical and statistical computations [14]. SPARQL Inferencing Notation (SPIN)[12] constraints associate RDF types or nodes with validation rules. These rules are expressed as SPARQL queries. There have been other proposals using SPARQL combined with other technologies, Simister and Brickley[27] propose a combination between SPARQL queries and property paths which is used in Google and Kontokostas et al [13] proposed *RDFUnit* a Test-driven framework which employs SPARQL query templates that are instantiated into concrete quality test queries.

Grammar based approaches define a domain specific language to

[9] Available at `https://github.com/labra/haws`

$$MatchShape\frac{\Gamma\{n \to l\} \vdash \delta(l) \simeq \Sigma_n^g \rightsquigarrow \tau}{\Gamma \vdash l \simeq_s n \rightsquigarrow \tau}$$

Figure 3: Inference rule to match shapes

$$Or_1\frac{\Gamma \vdash r_1 \simeq g \rightsquigarrow \tau}{\Gamma \vdash r_1|r_2 \simeq g \rightsquigarrow \tau} \qquad Or_2\frac{\Gamma \vdash r_2 \simeq g \rightsquigarrow \tau}{\Gamma \vdash r_1|r_2 \simeq g \rightsquigarrow \tau}$$

$$And\frac{\Gamma \vdash r_1 \simeq g_1 \rightsquigarrow \tau_1 \quad \Gamma \vdash r_2 \simeq g_2 \rightsquigarrow \tau_2}{\Gamma \vdash r_1 \parallel r_2 \simeq g_1 \oplus g_2 \rightsquigarrow \tau_1 \uplus \tau_2}$$

$$Empty\frac{}{\Gamma \vdash \varepsilon \simeq \{\} \rightsquigarrow \odot}$$

$$Star_1\frac{}{\Gamma \vdash r* \simeq \{\} \rightsquigarrow \odot} \qquad Star_2\frac{\Gamma \vdash r \simeq g_1 \rightsquigarrow \tau_1 \quad \Gamma \vdash r* \simeq g_2 \rightsquigarrow \tau_2}{\Gamma \vdash r* \simeq g_1 \oplus g_2 \rightsquigarrow \tau_1 \uplus \tau_2}$$

$$Arc_{type}\frac{p \in v_p \quad o \in v_o}{\Gamma \vdash \_ \xrightarrow{v_p} v_o \simeq \langle s,p,o\rangle \rightsquigarrow \odot} \qquad Arc_{ref}\frac{\Gamma \vdash l \simeq_s o \rightsquigarrow \tau}{\Gamma \vdash \_ \xrightarrow{v_p} l \simeq \langle s,p,o\rangle \rightsquigarrow \tau}$$

Figure 4: Inference rules for Shape expression schemas

declare the validation rules. OSLC Resource Shapes [25] have been proposed as a high level and declarative description of the expected contents of an RDF graph expressing constraints on RDF terms. Shape Expressions have been inspired by OSLC although they offer more expressive power. Dublin Core Application Profiles [8] also define a set of validation constraints using Description Templates with less expressiveness than Shape Expressions.

10. CONCLUSIONS AND FUTURE WORK

The industrial adoption of an RDF schema language will depend on rigorous analysis of efficiency of Shape Expressions and other approaches to schema.

In this paper, we propose an implementation of Shape Expressions inspired by derivatives of regular expressions.

There are two main lines of future work: On one hand, we are planning to develop a set of benchmarks that will enable us to assess the performance of the different shape expression implementations. On the other hand, we are currently working on the implementation of new features for the Shape Expression language. The evolution of the recently chartered W3c Data Shapes Working group will affect the adoption of those features. In this paper we offered a minimal set of language features which we consider representative. However, there are several extension proposals like inverse arcs, negations, predicates, etc. that could also be implemented using the proposed approach.

11. ACKNOWLEDGEMENTS

We thank Joshua Taylor for his careful review of this paper and his help in the definition of SPARQL queries. This work has been partially funded by the Spanish project MICINN-12-TIN2011-27871 ROCAS (Reasoning On the Cloud by Applying Semantics).

12. REFERENCES

[1] I. Boneva, J. E. Labra Gayo, S. Hym, E. G. Prud'hommeaux, H. Solbrig, and S. Staworko. Complexity and expressiveness of ShEx for RDF. In *International Conference on Database Theory (ICDT)*, 2015.

[2] D. Brickley and R. V. Guha. RDF Schema 1.1. `http://www.w3.org/TR/rdf-schema/`, 2014.

[3] J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[4] J. Clark. An algorithm for RELAX NG validation. `http://www.thaiopensource.com/relaxng/derivative.html`, 2002.

[5] K. Clark and E. Sirin. On RDF validation, stardog ICV, and assorted remarks. In *RDF Validation Workshop. Practical Assurances for Quality RDF Data*, Cambridge, Ma, Boston, September 2013. W3c, `http://www.w3.org/2012/12/rdf-val`.

[6] D. Colazzo, G. Ghelli, and C. Sartiani. Efficient inclusion for a class of xml types with interleaving and counting. *Inf. Syst.*, 34(7):643–656, Nov. 2009.

[7] R. Cox. Regular expression matching in the wild. `http://swtch.com/~rsc/regexp/regexp3.html`, March 2010.

[8] K. Coyle and T. Baker. Dublin core application profiles. separating validation from semantics. In *RDF Validation Workshop. Practical Assurances for Quality RDF Data*, Cambridge, Ma, Boston, September 2013. W3c, `http://www.w3.org/2012/12/rdf-val`.

[9] J. English. How to validate XML. `http://www.flightlab.com/~joe/sgml/validate.html`.

[10] W. Gelade. Succinctness of regular expressions with interleaving, intersection and counting. *Theoretical Computer Science*, 411(31-33):2987 – 2998, 2010.

[11] P. J. Hayes and P. F. Patel-Schneider. RDF 1.1 Semantics. `http://www.w3.org/TR/rdf11-mt/`, 2014.

[12] H. Knublauch. SPIN - Modeling Vocabulary. `http://www.w3.org/Submission/spin-modeling/`, 2011.

[13] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, and A. Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 747–758, Republic and Canton of Geneva, Switzerland, 2014. International World Wide Web Conferences Steering Committee.

[14] J. E. Labra Gayo and J. M. Álvarez Rodríguez. Validating statistical index data represented in RDF using SPARQL queries. In *RDF Validation Workshop. Practical Assurances for Quality RDF Data*, Cambridge, Ma, Boston, September 2013. W3c, http://www.w3.org/2012/12/rdf-val.

[15] J. E. Labra Gayo, J. Jeuring, and J. M. Álvarez Rodríguez. Inductive representations of RDF graphs. *Science of Computer Programming*, 95, Part 1(0):135 – 146, 2014. Special Issue on Systems Development by Means of Semantic Technologies.

[16] J. E. Labra Gayo, E. Prud'hommeaux, H. Solbrig, and J. M. Alvarez Rodríguez. Validating and describing linked data portals using RDF Shape Expressions. In *1st Workshop on Linked Data Quality*, Sept. 2014.

[17] M. Might, D. Darais, and D. Spiewak. Parsing with derivatives: A functional pearl. *SIGPLAN Not.*, 46(9):189–195, Sept. 2011.

[18] B. Motik, I. Horrocks, and U. Sattler. Adding Integrity Constraints to OWL. In C. Golbreich, A. Kalyanpur, and B. Parsia, editors, *OWL: Experiences and Directions 2007 (OWLED 2007)*, Innsbruck, Austria, June 6–7 2007.

[19] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of xml schema languages using formal language theory. *ACM Trans. Internet Technol.*, 5(4):660–704, Nov. 2005.

[20] OASIS Committee Specification. RELAX NG Specification:. http://relaxng.org/spec-20011203.html, 2001.

[21] S. Owens, J. Reppy, and A. Turon. Regular-expression derivatives re-examined. *Journal of Functional Programming*, 19(2):173–190, 2009.

[22] W. OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 2012. Available at http://www.w3.org/TR/owl2-overview/.

[23] E. Prud'hommeaux, J. E. Labra, and H. Solbrig. Shape expressions: An RDF validation and transformation language. In *10th International Conference on Semantic Systems*, Sept. 2014.

[24] RDF Working Group W3c. W3c validation workshop. practical assurances for quality rdf data, September 2013.

[25] A. G. Ryman, A. L. Hors, and S. Speicher. OSLC resource shape: A language for defining constraints on linked data. In C. Bizer, T. Heath, T. Berners-Lee, M. Hausenblas, and S. Auer, editors, *Linked data on the Web*, volume 996 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[26] U. Schmidt. Regular expressions for XML Schema. http://www.haskell.org/haskellwiki/Regular_expressions_for_XML_Schema, 2010.

[27] S. Simister and D. Brickley. Simple application-specific constraints for rdf models. In *RDF Validation Workshop. Practical Assurances for Quality RDF Data*, Cambridge, Ma, Boston, September 2013. W3c, http://www.w3.org/2012/12/rdf-val.

[28] M. Sulzmann and K. Z. M. Lu. POSIX regular expression parsing with derivatives. In M. Codish and E. Sumii, editors, *Functional and Logic Programming - 12th International Symposium, FLOPS 2014, Kanazawa, Japan, June 4-6, 2014. Proceedings*, volume 8475 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2014.

[29] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity constraints in OWL. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*. AAAI, 2010.

[30] E. van der Vlist. *Relax NG: A Simpler Schema Language for XML*. O'Reilly, Beijing, 2004.

RDF Constraint Checking

Peter M. Fischer, Georg Lausen, Alexander Schätzle
Univ. of Freiburg, Faculty of Engineering, 79110 Freiburg, Germany
{peter.fischer,lausen,schaetzle}@informatik.uni-freiburg.de

Michael Schmidt
metaphacts GmbH, Industriestraße 39c, 69190 Walldorf, Germany
ms@metaphacts.com

ABSTRACT

Linked Open Data (LOD) sources on the Web are increasingly becoming a mainstream method to publish and consume data. For real-life applications, mechanisms to describe the structure of the data and to provide guarantees are needed, as recently emphasized by the W3C in its Data Shape Working Group. Using such mechanisms, data providers will be able to validate their data, assuring that it is structured in a way expected by data consumers. In turn, data consumers can design and optimize their applications to match the data format to be processed.

In this paper, we present several crucial aspects of RDD, our language for expressing RDF constraints. We introduce the formal semantics and describe how RDD constraints can be translated into SPARQL for constraint checking. Based on our fully working validator, we evaluate the feasibility and efficiency of this checking process using two popular, state-of-the-art RDF triple stores. The results indicate that even a naive implementation of RDD based on SPARQL 1.0 will incur only a moderate overhead on the RDF loading process, yet some constraint types contribute an outsize share and scale poorly. Incorporating several preliminary optimizations, some of them based on SPARQL 1.1, we provide insights on how to overcome these limitations.

1. INTRODUCTION

Linked Open Data (LOD) sources on the Web using RDF are increasingly becoming popular. As a consequence mechanisms are needed that can be used to validate RDF datasets. Using such means data providers can validate their data to assure that they are providing information structured in a way as expected by data consumers, and other way round, data consumers can validate their interfaces against the data to be processed. As RDF is a graph based data model, validation not only should refer to single triples, but also graph patterns must be considered. The RDF Validation Workshop [16] states a gap between the current standards offering and the industry needs for validation of RDF data. More recently, in continuation of the workshop, a W3C working group is in the process of being established [4]. As major issues, this working group will address the definition and publication of topology and value constraints of RDF graphs, validation of such constraints and optimization of SPARQL queries based on it. To tackle these issues, in continuation of our previous work [9], we have developed a constraint language RDD (RDF Data Descriptions) [13, 14] that captures a broad range of constraints including keys, cardinalities, subclass, path and subproperty restrictions, making it easy to implement RDD checkers and clearing the way for semantic query optimization.

The intention of an RDD is similar to SPIN [5], IBM's Resource Shapes [11], and Stardog ICV [6], where among these systems Stardog ICV seems to be the one an RDD is mostly related to. In Stardog ICV [6], constraints are stated using OWL and considered relative to a certain inference machinery whose type may range from no inferencing, RDFS- to OWL-inferencing. In contrast, RDD is a language using a compact special-purpose syntax designed for only expressing constraints independent of a specific inference machinery. This makes RDD in particular applicable for RDF under ground semantics, which is a common scenario in the Linked Data context.

Just recently two other interesting validation methods have been proposed. Shape Expressions [10] semantically act as a type inference system that can derive types for given nodes in an RDF graph. Its functionality resembles schema languages for XML, in particular RelaxNG. A test-driven approach for validation is suggested in [8]. Test cases may be manually derived or automatically from existing RDFS/OWL specifications. These approaches are similar to RDDs in the sense that the final validation can be performed using SPARQL query expressions. However, while Shape expressions are based on regular expressions, RDDs incorporate relational constraints to RDF and therefore support the mapping of relational databases to RDF using R2RML [2], for example. While [8] is based on templates which are instantiated and afterwards executed to determine the degree to which corresponding constraints are fulfilled, RDD constraints are checked for fulfillment and in case they are violated, for efficiency reasons, only a small number of counterexamples is listed. As a distinctive feature, different to both discussed approaches, RDDs are based on a human readable language in a similar vein to relational databases. Finally, the topic of our current paper is measuring the cost of various constraint patterns in particular for data sets of

varying size to get more information about scalability and starting points for optimization. Neither [10] nor [8] elaborate on these aspects. However, we emphasize that the constraint types considered in these works are similiar to a large degree to the ones imposed by RDDs, such that many of the results on efficient constraint checking via SPARQL proposed in this work carry over to these approaches as well.

In the current paper we elaborate on checking constraints described using RDDs. We first describe an RDD constraint checker which maps a given RDD into a set of SPARQL 1.0 queries. The main contribution of the current paper is a comprehensive experimental evaluation of the checking process. We analyze the overhead induced by the various constraint types and demonstrate the effectiveness of different kinds of optimization, where some optimizations are based on SPARQL 1.1 language features. We show that constraints proposed in RDD can be validated with moderate cost compared to the initial loading of a respective RDF graph.

The paper is organized as follows. In Section 2 we present RDD, the *RDF Data Description* language [13], which we use to define constraints over RDF graphs. Section 3, to have a formal basis, presents a first-order logic (FOL) semantics of RDDs. Section 4 describes our *RDD Checker* implementation of RDD based on a mapping from FOL to SPARQL and Section 5 presents the findings of our *RDD Checker* evaluation. In Section 6 we then discuss several ideas on how to optimize the RDD to SPARQL mapping and illustrate their potential impact on the efficiency of the checking process in Section 7. Section 8 concludes the paper and gives an outlook on future work.

2. RDF DATA DESCRIPTION (RDD)

The RDF data description language (RDD)[13, 14] allows to express the following kinds of constraints:

- A RangeTypeConstraint indicates that the property *prop* points to either a URI, BlankNode, Resource, or a (possibly typed) Literal.
- A Min/MaxConstraint indicates that the property *prop* occurs at least or at most a number of times, respectively.
- A Domain/RangeConstraint indicates a guaranteed *domain* or *range* for subject and objects associated with property *prop*, respectively.
- A PathConstraint indicates that the value of property *prop* can as well be reached by following a given *path* of properties.
- A SubPropertyConstraint indicates that for every triple using property *subProp*, there is also an identical triple using property *prop*.
- A Partiality/TotalityConstraint expresses that property *prop* occurs at most or exactly one time, respectively.

All these constraints may occur in *unqualified* form, i.e. hold for a property independently of its context, or in *qualified* form, i.e. hold for a property only when the property is used in combination with a subject of a given (fixed) class. While the above mentioned constraints all refer to properties, the following class-specific constraints exist:

- A SingletonConstraint indicates that a class has exactly one instance.
- A KeyConstraint indicates the properties uniquely identifying the entities of a class in all possible class instances.
- A SubclassConstraint allows for the inheritance of constraints along class hierarchies.

```
PREFIX ex: <http://www.example.com#>
...
CWA CLASSES {
 OWA CLASS foaf:Person SUBCLASS ex:Student {
   KEY rdfs:label : LITERAL
   MAX(2) foaf:mbox : LITERAL
   TOTAL foaf:age : LITERAL(xsd:integer)
   RANGE(foaf:Person) foaf:knows : IRI }

 OWA CLASS ex:Student {
   KEY ex:matricNr : LITERAL(xsd:integer)
   MIN(1), RANGE(ex:Course) ex:course : RESOURCE
   PATH(ex:course/ex:givenBy),
     RANGE(foaf:Person) ex:taughtBy : IRI }

 OWA CLASS ex:Course { ... }
}

OWA PROPERTIES {
  PARTIAL foaf:nick : LITERAL
  foaf:knows SUBPROPERTY ex:taughtBy
}
```

Figure 1: Example RDF Data Description

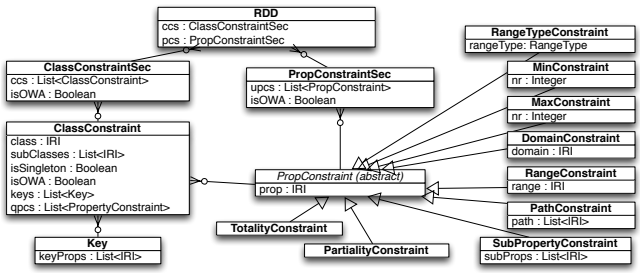


Figure 2: Structural Overview of the RDD Language

Figure 1 provides an example RDD demonstrating the usage of the various constraint types. At top-level, an RDDs consist of two main sections:

(i) A class constraint section (keyword CLASSES) defining *qualified property constraints* and *class-specific* constraints.

(ii) A global property constraint section (keyword PROPERTIES) defining *unqualified property constraints*.

The class constraint section contains a list of `CLASS` definitions, where each may contain a set of (qualified) property constraints. The RDD in Fig. 1 defines a class `foaf:Person`, where property `rdfs:label` as `KEY` uniquely identifies a person, which has at most two mailboxes (`foaf:mbox`) associated, exactly one age (`foaf:age`) and `foaf:knows` always

points to objects of type `foaf:Person`. It also defines a class `ex:Student` as a subclass of `foaf:Person`. With RDDs focusing on instance-level constraints, this is not a subclass relation in the sense of RDFS, but guarantees that every instance of class `ex:Student` satisfies the same constraints as defined for `foaf:Person`. Additionally, every student must be uniquely identified by a property `ex:matricNr` as `KEY` and is enrolled in at least one course (`ex:course` always pointing to objects of type `ex:Course`). Moreover, for every property `ex:taughtBy` there is a path along the properties (edges) `ex:course` followed by `ex:givenBy` pointing to the same entity of type `foaf:Person`. The OWA specifications coming with the class definitions for `foaf:Person`, `ex:Student`, and `ex:Course` say that these classes are interpreted under open world assumption, i.e. an instance may carry properties other than those listed in the body. Differing in its semantics, the CWA constraint associated with the top-level CLASSES section implies that there are no classes other than those specified in its body (i.e., `foaf:Person`, `ex:Student`, and `ex:Course`); keyword OWA associated with the PROPERTIES section indicates that no such constraint is imposed at property level. This example illustrates that RDDs allow to specify a mix of open and close world semantics at different levels. Finally, the property constraint section defines that every person or student may have one nickname (`foaf:nick`) and if it has a `ex:taughtBy` property pointing to x, it also has property `foaf:knows` pointing to x.

Figure 2 visualizes the syntactical structure and concepts of RDD in a UML-style notation. Boxes denote concepts, arrowed lines sub-concepts relationships and the remaining line type a uses-relationship. A more detailed description of the RDD syntax can be found in [14].

3. RDD SEMANTICS

We first like to introduce the basic RDF notation [3]. Let U be a set of URI references, B a set of blank nodes and L a set of literals. A triple $t := (s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an *RDF triple*; s is called subject, p property and o object. A finite set of triples is called an *RDF graph*.

Let an RDF graph G and an RDD r be given. Following [14] we shall now demonstrate how a corresponding set cs of FOL constraints can be derived. We stick to the following two notational conventions. Variables are distinguished from other terms by using \$ as a prefix. Moreover, formulas of the kind $\forall \$x_1, \ldots, \$x_n \phi$ are abbreviated to ϕ thereby assuming that all free variables in ϕ are globally $\forall$-quantified. We shall use four unary relations IRI, $BNode$, $Resource$, and $Literal$ containing all IRIs, blank nodes, resources (i.e. IRIs and blank nodes), and literals that appear in any position of any triple in G, respectively. An RDF graph G is modeled as a ternary relation $G(s, p, o)$ representing the triples of the respective graph in the obvious way. To improve readability, we define the following two shortcuts:

$$allDist(\$x_1, \ldots, \$x_n) := \bigwedge_{1 \le i < j \le n} \$x_i \neq \$x_j, \text{ and}$$
$$someEq(\$x_1, \ldots, \$x_n) := \bigvee_{1 \le i < j \le n} \$x_i = \$x_j,$$

enforcing that the n variables passed as parameters are all pairwise distinct (*allDist*) or some of them are equal (*someEq*). In the following, we present the constraint types that are imposed through the constructs in RDDs.

Unqualified CWA$_P$.

Whenever CWA PROPERTIES is specified in r, the unqualified property constraint cwa_P is used to restrict the usage of properties to only those which are mentioned in the RDD's property section. Let $p_1, \ldots, p_n$ be the properties mentioned in the unqualified property constraint section. Then cwa_P is defined as follows:

$$cwa_P : \; G(\$s, \$p, \$o) \rightarrow \$p = p_1 \vee \cdots \vee \$p = p_n$$

Unqualified Property Constraints.

The *unqualified range type* restriction enforces the range type of a property, according to one of the keywords **IRI**, **BNODE**, **RES(OURCE)**, **LIT(ERAL)** or some given type R specified in the RDD specification. This gives rise to the following kinds of constraints:

$$range(p, \mathbf{IRI}) : \; G(\$s, p, \$o) \rightarrow IRI(\$o)$$
$$range(p, \mathbf{BNODE}) : \; G(\$s, p, \$o) \rightarrow BNode(\$o)$$
$$range(p, \mathbf{RES}) : \; G(\$s, p, \$o) \rightarrow Resource(\$o)$$
$$range(p, \mathbf{LIT}) : \; G(\$s, p, \$o) \rightarrow Literal(\$o)$$
$$range(p, R) : \; G(\$s, p, \$o) \rightarrow G(\$o, \mathbf{rdf:type}, R)$$

The remaining unqualified property constraints are *domain, min, max, total, subprop, part* and defined as follows:

$$domain(p, D) : \; G(\$s, p, \$o) \rightarrow G(\$s, \mathbf{rdf:type}, D)$$
$$min(p, n), n \geq 1 : \; Resource(\$s) \rightarrow \exists \$o_1, \ldots \$o_n$$
$$(G(\$s, p, \$o_1) \wedge \cdots \wedge G(\$s, p, \$o_n) \wedge allDist(\$o_1, \ldots, \$o_n))$$
$$max(p, n), n \geq 1 : \; G(\$s, p, \$o_1) \wedge \cdots \wedge G(\$s, p, \$o_{n+1})$$
$$\rightarrow someEq(\$o_1, \ldots, \$o_{n+1})$$
$$total(p) : \; min(p, 1) \wedge max(p, 1)$$
$$part(p) : \; max(p, 1)$$
$$subprop(p, p_s) : \; G(\$s, p_s, \$o) \rightarrow G(\$s, p, \$o)$$
$$path(p, q_1, \ldots, q_n), n \geq 1 : G(\$s, p, \$o) \rightarrow \exists \$o_1, \ldots, \$o_{n-1}$$
$$(G(\$s, q_1, \$o_1) \wedge \cdots \wedge G(\$o_{n-1}, q_n, \$o))$$

Note that *total* and *part* both define functional restriction of a property p; using $total(p)$ the property p must be defined for all subjects, whereas for $part(p)$ there may exist subjects in G where p is not defined.

Qualified CWA$_P$ and Qualified Property Constraints.

The qualified versions of constraints are different from the unqualified only in that their application is restricted to a corresponding class C, i.e. conjugating $G(\$s, \mathbf{rdf:type}, C)$ to the prerequisites of the corresponding constraint. For example, a range restriction qualified by class C is of the following form:

$$range(p, C, R) :$$
$$G(\$s, \mathbf{rdf:type}, C) \wedge G(\$s, p, \$o) \rightarrow G(\$o, \mathbf{rdf:type}, R)$$

Class Constraints.

Finally, as part of the constraint section of a class C, class constraints *key* and *singleton* can be specified[1]:

$$key(C, p_1, \ldots, p_n, R_1, \ldots, R_n) :$$
$$range(p_1, C, R_1) \wedge \ldots \wedge range(p_n, C, R_n) \wedge$$
$$total(p_1, C) \wedge \ldots \wedge total(p_n, C) \wedge$$
$$(\; G(\$s_1, \mathbf{rdf:type}, C) \wedge G(\$s_2, \mathbf{rdf:type}, C) \; \wedge$$

[1] *subclass* constraints are handled as described in [14] and need not be considered in the context of the current paper.

$G(\$s_1, p_1, \$o_1) \wedge \cdots \wedge G(\$s_1, p_n, \$o_n) \wedge$
$G(\$s_2, p_1, \$o_1) \wedge \cdots \wedge G(\$s_2, p_n, \$o_n) \rightarrow \$s_1 = \$_2$)
singleton(C) :
$\exists \$s(G(\$s, \mathbf{rdf:type}, C)) \wedge$
($G(\$s_1, \mathbf{rdf:type}, C) \wedge G(\$s_2, \mathbf{rdf:type}, C) \rightarrow \$s_1 = \$s_2$)

Moreover, whenever CWA CLASSES is specified in r, the constraint CWA_C can be used to restrict the usage of classes to only those which are mentioned in the RDD. Let $c_1, \ldots, c_n$ be the classes mentioned in the class section. Then cwa_C is the constraint:

$$cwa_C : \; G(\$s, \mathbf{rdf:type}, \$c) \rightarrow \$c = c_1 \vee \cdots \vee \$c = c_n$$

Finally, we define the consistency of an RDF graph w.r.t. a given RDD as follows:

Definition *Let G be an RDF graph and let cs be the set of first-order logic constraints defined by a corresponding RDD r. RDF graph G is consistent with respect to cs if and only if for all constraints $c \in cs$ it holds that c is valid in G, i.e. $G \models c$, respectively, $G \models cs$.*

It is well-known that for a fixed set of FOL constraints consistency of a given arbitrary RDF data set can be decided in polynomial time. In particular, in the following sections we will discuss an appropriate mapping into SPARQL.

4. RDD CHECKER

We have implemented the aforementioned decomposition of an RDD into the corresponding set of FOL constraints which can be used for further investigation, e.g. it may serve as input to an FOL reasoner. Furthermore, we have also implemented a mapping from the generated FOL constraints to corresponding SPARQL 1.0 queries which can be used to check the consistency of an RDF dataset w.r.t. a given RDD. As the resulting queries are compliant to the SPARQL 1.0 spec, they can be executed with any SPARQL 1.0 query engine. Our implementation comes with built-in bindings for *Sesame* [1], such that a given RDD can be verified against any RDF dataset out of the box. In addition, the checker can be pointed at arbitrary SPARQL endpoints. The binaries and source code of our *RDD Checker* (implemented in Java) are available for public download[2].

In the following we give an exemplary depiction of how we can use SPARQL to check whether an RDD constraint in FOL holds on a given RDF document. For more details on the connection between SPARQL and FOL, the interested reader may be referred to, e.g., [9, 15].

Consider the totality constraint (`TOTAL foaf:age`) on class `foaf:Person` taken from Figure 1. This constraint assures that every entity in an RDF dataset of type `foaf:Person` needs to have exactly one `foaf:age` property defined. As defined in Section 3, a $total(p)$ constraint is a combination of $min(p, 1)$ and $max(p, 1)$ requiring that every person has at least and at most one age, respectively. This gives rise to the following two (qualified) FOL rules:

$min(\mathbf{foaf:age}, \mathbf{foaf:Person}, 1)$:
$G(\$s, \mathbf{rdf:type}, \mathbf{foaf:Person}) \rightarrow \exists \$o_1(G(\$s, \mathbf{foaf:age}, \$o_1)$
$max(\mathbf{foaf:age}, \mathbf{foaf:Person}, 1)$:
$G(\$s, \mathbf{rdf:type}, \mathbf{foaf:Person}) \wedge$
$G(\$s, \mathbf{foaf:age}, \$o_1) \wedge G(\$s, \mathbf{foaf:age}, \$o_2) \rightarrow \$o_1 = \$o_2$

[2] `http://dbis.informatik.uni-freiburg.de/forschung/projekte/rdd/`

We do not use a pattern-based translation approach where there is a query pattern for each constraint type but instead define it along the structure of the corresponding FOL rules. The concept of our mapping from an RDD constraint c to SPARQL is to define a query for every FOL rule imposed by c that retrieves those entities from an RDF graph G violating the rule. If no such entities exist for every rule of c, then $G \models c$. The generic idea is to define a *graph pattern* matching the body of the rule, and use a *filter* expression to select only those entities matching the graph pattern that do not fulfill the head of the rule. As a single witness already leads to violation, we can limit the number of results such that a query engine does not have to compute all results, if supported. Our *RDD Checker* implementation uses a customizable default value of three. The corresponding SPARQL queries for the given *min* and *max* rules from above are listed in Figure 3.

MIN(1):
```
SELECT ?s {
  ?s rdf:type foaf:Person
  OPTIONAL { ?s foaf:age ?o1 }
  FILTER (!BOUND(?o1))
} LIMIT 3
```

MAX(1):
```
SELECT ?s {
  ?s rdf:type foaf:Person .
  ?s foaf:age ?o1 . ?s foaf:age ?o2
  FILTER (!(?o1=?o2))
} LIMIT 3
```

Figure 3: SPARQL queries for constraint *total*(foaf:age) on class foaf:Person

The atoms of a rule can be represented by *triple patterns* (i.e. triples with variables) in the SPARQL query, e.g. $G(\$s, \mathbf{foaf:age}, \$o_1)$ can be mapped to `?s foaf:age ?o1`. The concatenation of atoms in the body of a rule can then be equivalently represented by a set of `AND`(.) connected triple patterns forming a so-called *basic graph pattern*. The following filter expression then defines the negation of the rule's head. An equality-generating head can be simply represented by a `FILTER` where we negate (denoted by `!`) the conditions of the head (see *max* constraint). In the case of an existentially quantified head we use a combination of `OPTIONAL` and `!BOUND` as SPARQL 1.0 does not have a natural support for negation (see *min* constraint). This way, the `FILTER` only accepts those bindings for variable `?s` where `OPTIONAL` did not find any binding for `?o1`, hence those entities (persons) that do not have an a `foaf:age` property. This construct is equivalent to the explicit `FILTER NOT EXISTS` functionality added in SPARQL 1.1.

This is a rather direct mapping and we can apply this strategy to all RDD constraints to generate queries that adhere to the SPARQL 1.0 spec. Our *RDD Checker* implementation currently uses this mapping such that every available SPARQL 1.0 query engine can be used to check the validity of RDD constraints on an arbitrary RDF dataset.

Though this one-to-one mapping gives us a correct and complete realization of RDD, it is not an optimal solution in terms of efficiency. As the example already illustrates, an RDD constraint can consist of more than one FOL rule in general and hence lead to more than one SPARQL query for verification. As many queries have to iterate over the

whole dataset or the same parts of it, this naturally raises the issue of efficiency of the checking process and possible optimizations to reduce the number of iterations over the whole dataset. In the following section, we first present the findings of our *RDD Checker* evaluation based on the mapping to SPARQL as described in this section. In Section 6 we then discuss several ideas on how to optimize the representation of RDD constraints in SPARQL and illustrate their potential impact on the efficiency of the checking process in Section 7.

5. RDD CHECKER EVALUATION

The goal of our evaluation is to determine how validation compares with common database operations and how individual constraints contribute to it. From these findings, we can then derive potential opportunities for optimizations.

Setup.

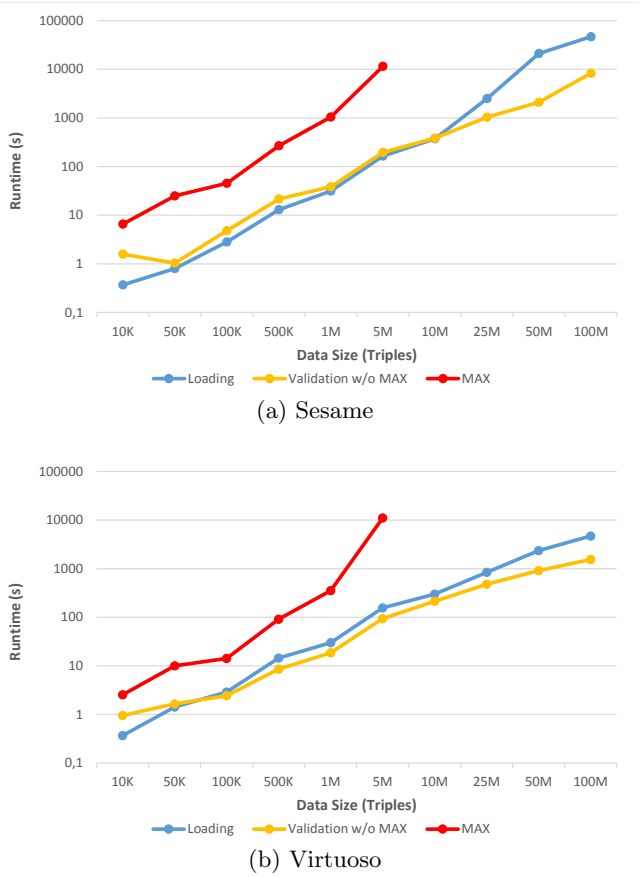
We perform our evaluation on top of Sesame [1] 2.7.12 and Virtuoso Open Source 7.1.0, which are commonly used, highly compliant and feature-complete SPARQL implementations. Sesame supports a range of storage backends, out of which we picked the Native Java Disk Storage without Schema Reasoning, as it supports almost arbitrary data sizes and does not impose any additional cost. The experiments were performed on a system with a single Xeon X5667 with 4 physical cores (8 hyperthreaded) at 3.06 GHz, 32 GB RAM and 12 TB of disk storage (LSI MegaRaid with 4x4TB LGST 7200 rpm SATA Deskstar disks in a RAID 5 configuration), running Ubuntu Linux 12.04 LTS. The Java for Sesame heap size was set to 28 GB, sufficient to keep even the largest dataset we tested in memory. Virtuoso is a native program, so no such tuning was needed.

Data and Constraints.

We studied the modeling of constraints and the cost of validating them on the SP2 Benchmark [12]. SP2Bench contains a well-defined and rather structured RDF dataset with documented constraints and data distributions, which models a publication database similar to DBLP. In contrast to most RDF datasets, it includes a generator that can generate a wide of range of scalings while maintaining distributions and correlations. For the scope of this paper, we evaluated datasets ranging from 10K to 100M triples, corresponding to 1.1 MB to 11 GB when stored as N3 files. This range covers a majority of typical, real-life RDF datasets.

Our RDD file takes a class-centric approach (similar to relational or object-oriented modelling), describing 12 classes with mixed OWA/CWA settings, *key*, *partial*, *total* and *range* definitions. In total it contains 215 constraints that were mined from the SP2Bench dataset, and hold over all scalings tested. The RDD is available for download from our project website[3]. All RDDs were translated and directly validated using our RDD checker implementation, yielding 251 SPARQL queries, since the translation of e.g., *total* or *key* constraints needs several SPARQL queries for a single constraint (c.f. Section 4). These queries are set to use the SELECT form of SPARQL with a LIMIT of 3 as to produce a small number of witnesses of the violations. We study lifting this limit in Section 7.

[3]`http://dbis.informatik.uni-freiburg.de/forschung/projekte/rdd/`



(a) Sesame

(b) Virtuoso

Figure 4: Cost of RDD Validation compared to loading

Validation vs. Loading.

In our first experiment, we compare the cost of validating our RDD file with the cost of loading data. We consider such bulk validation a common application in order to publish stable or slowly changing data. Figure 4 shows this comparison over the entire range of data sizes we analyzed for both triple stores. Sesame and Virtuoso show the same behaviour, with just overall higher performance for Virtuoso. The results show that the cost and scaling validation needs to broken down in two, very distinct sets: Without the *max* constraints, validation scales well, even better than loading. At lower scales, validation is actually held back by the effort of invoking 251 individual queries, which in turn mostly need to access the full dataset each, highlighting significant optimization potential. The *max* constraints have much higher cost and scale much worse. Beyond a scale of 5m triples, many individual *max* queries take longer than 90 minutes, which we had set as a timeout (corresponding to the load time of the largest data set). This high cost and bad scaling is caused by the need to express violations of a $max(n)$ constraint by a $n+1$-way join (see Section 6.1), pointing out a massive inefficiency.

Individual Constraints.

In our second experiment we further investigated the impact of individual constraint classes, giving us insights into possible optimizations and design guidelines. The second most expensive clause is CWA for classes, since it needs to visit all triples belonging to a specific class and check if their

predicates belong to these in the class definition. Next, *key* stands out of the remainder because it needs to employ a join on object (to find same key value for different instances) which is not well supported. *min* would suffer from the same issues as *max*, but the RDD file does not contain any *min* constraints with a high threshold.

We also investigated the effect of scoping on the validation, gradually moving constraints that are shared over almost all classes or can be shared with certain relaxations (e.g., a higher *max* value) to the global properties section. Clearly, these constraints now need to be tested over a larger set of data, but the number of tests will be smaller (one test per property, not one test per property and class) and the test queries themselves will be simpler (avoiding a join on the class type). In the first step, we moved *partial* and *range* constraints, if possible, from the classes section to the global properties section as they are non-conflicting. This change reduced the number of constraints by around 50 percent and yielded a runtime saving of around 25-30 percent. In the second step, we consolidated the *max* constraints of the same property in various classes into a single global property. Since not all classes had the same $max(n)$ value for the same property, we always chose the maximum n, thus weakening the precision of checking. The number of *max* constraints goes down from 10 to 2, but we mostly eliminate those with a small threshold. As a result, the savings are rather limited, yielding only 2 to 5 percent. The corresponding RDDs are also available for download on the project website.

6. OPTIMIZATIONS

It is fairly obvious that the one-to-one mapping of FOL rules representing RDD constraints to SPARQL queries, as described in Section 4, leaves a lot of leeway for optimizations in various directions. Since these optimization rely on deeper understanding of the constraint semantics, SPARQL optimizers cannot detect them. First, there is a potential for *intra-query* optimization, i.e. the individual SPARQL query. Second, *intra-constraint* optimization can reduce the number of SPARQL queries required for checking an individual constraint. And third, *inter-constraint* optimization may give the chance to check several constraints at once in a single query (e.g. several *max* constraints for different properties), also reducing the total number of queries.

In the following we give some insights on intra-query and intra-constraint optimization that we have identified in our *RDD Checker* evaluation (see Section 5). A study of inter-constraint optimization is left for future work.

6.1 Intra-Query Optimization

If we look at the meaning of RDD constraints, many of them are intended to restrict the number of occurrences of specific properties in one way or another, e.g. *min*, *max*, *part*, *total*. This kind of restriction naturally leads to grouping and counting the elements of a group, which is not provided by SPARQL 1.0 directly. Instead, it can be realized by a series of joins as described in the following.

Consider the maximum constraint (`MAX(2) foaf:mbox`) on class `foaf:Person` taken from Figure 1, assuring that every person has at most two mailboxes. This gives rise to the following (qualified) FOL rule:

$$max(\textbf{foaf:mbox}, \textbf{foaf:Person}, 2) :$$
$$G(\$s, \textbf{rdf:type}, \textbf{foaf:Person}) \wedge G(\$s, \textbf{foaf:mbox}, \$o_1) \wedge$$
$$G(\$s, \textbf{foaf:mbox}, \$o_2) \wedge G(\$s, \textbf{foaf:mbox}, \$o_3)$$
$$\rightarrow \$o_1 = \$o_2 \vee \$o_1 = \$o_3 \vee \$o_2 = \$o_3$$

If we look at the corresponding mapping to SPARQL 1.0 in Figure 5, following the strategy described in Section 4, we can see that it results in a graph pattern consisting of four triple patterns. To retrieve the result of this pattern, a query engine typically has to compute three joins between subsets of the data. In the following filter expression we then have to check whether any two variables are bound to the same entity. If not, there exists a person with at least three mailboxes violating the constraint. It is obvious that the complexity of the query increases with the specified maximum. In general, a $max(p, C, n)$ query following this strategy contains $n + 1$ joins and $\binom{n+1}{2}$ filter conditions.

SPARQL 1.0:

```
SELECT ?s {
  ?s rdf:type foaf:Person .
  ?s foaf:mbox ?o1 . ?s foaf:mbox ?o2 . ?s foaf:mbox ?o3
  FILTER (!(?o1=?o2 || ?o1=?o3 || ?o2=?o3))
} LIMIT 3
```

SPARQL 1.1:

```
SELECT ?s {
  ?s rdf:type foaf:Person . ?s foaf:mbox ?o1
} GROUP BY ?s HAVING (COUNT(?o1) > 2)
LIMIT 3
```

Figure 5: Intra-query optimization for constraint max(foaf:mbox, 2) on class foaf:Person

One of the functionalities introduced in SPARQL 1.1 is the support for groupings and aggregations similar to those in SQL. Using these features we can simplify the corresponding SPARQL query as also illustrated in Figure 5, reducing the number of joins to only a single one. In fact, the query structure in SPARQL 1.1 is independent from the specified maximum as it only affects the counting threshold. Assuming that query engines (which are typically based on simple algebraic rewritings) are not capable of performing such complex optimizations, one may expect the query in SPARQL 1.1 to be much more efficient than in SPARQL 1.0 for larger maximum values. This assumption could be clearly confirmed in our experiments (see Section 7). A very similar optimization is also possible for *min* constraints.

6.2 Intra-Constraint Optimization

Following the mapping from FOL rules to SPARQL queries described in Section 4, some RDD constraints generate more than one SPARQL query for verification. This raises the issue if we can combine some of these to reduce the total number of queries. Here, we focus on the combination of queries for an individual constraint, whereas generally it might also be possible to combine queries of different constraints.

As an example, consider again the totality constraint illustrated in Section 4, $total(\textbf{foaf:age}, \textbf{foaf:Person})$, and the corresponding queries listed in Figure 3. Limited to the functionality of SPARQL 1.0 we cannot usefully combine both queries as they are structurally different from each other. The only way would be to use a `UNION` of both patterns but it is very likely that query engines will not be able to combine them and just compute both patterns the same way as for different queries which would not reduce the computation cost at all.

But, as already illustrated in Section 6.1, with SPARQL 1.1 we can write *min* and *max* queries using groupings and aggregations. In this way, both queries can be easily combined as they exhibit a very similar structure. Figure 6 shows the combined query in SPARQL 1.1 that retrieves all persons (i.e. entities of type `foaf:Person`) where the number of occurrences of property `foaf:age` is not equal to one. If such a person exists, the *total* constraint is violated.

Combine MIN(1) and MAX(1) using SPARQL 1.1:

```
SELECT ?s {
  ?s rdf:type foaf:Person
  OPTIONAL { ?s foaf:age ?o1 }
} GROUP BY ?s HAVING (COUNT(?o1) != 1)
LIMIT 3
```

Figure 6: Intra-constraint optimization for constraint *total*(foaf:age) on class foaf:Person

The structural difference to the *max* query as illustrated in Section 6.1 is that we have to use an `OPTIONAL` clause for the triple pattern matching the constraint property (here `foaf:age`). This is required because the query must also retrieve those persons that have no specified age property as this is a violation of the *min*(1) requirement and thus also a violation of the *total* constraint. If we would use a simple basic graph pattern (as we can do it for a *max* constraint) these persons would not be part of the query result.

In the following experiments (Section 7) we demonstrate that an individual rewriting of both queries, *min*(1) and *max*(1), to use SPARQL 1.1 does not give a performance benefit compared to the representation in SPARQL 1.0 (the performance is actually even worse). However, the combination of both queries into a single one leads to a performance improvement between 30 and 50 percent.

7. OPTIMIZATION EXPERIMENTS

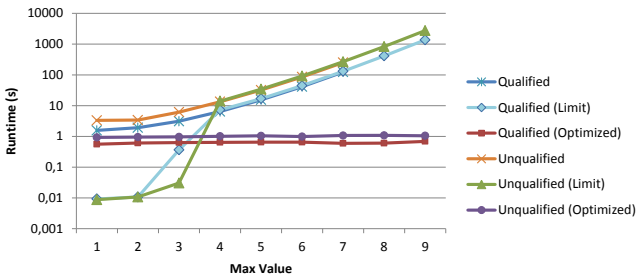
We have done some experiments on the impact of the aforementioned optimizations on the efficiency of the checking process, especially for *max* constraints as they have proven to be by far the most expensive constraint type in our evaluation (cf. Section 5).

Intra-Query Optimizations.

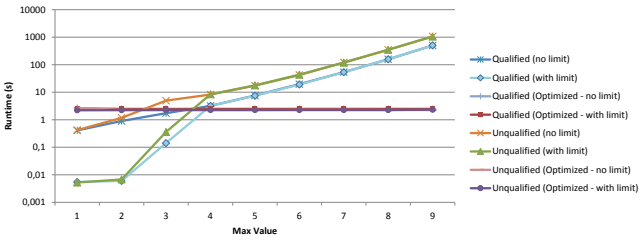
We begin our study with optimizations on individual queries. Our main focus is on *max*, given its cost and the potential benfits it can draw from SPARQL 1.1 features. Furthermore, *max* provides insights into *min*, *total*, *partial* and *key* constraints, which can be expressed at least partially as variants of *max*. For this purpose, we tested the optimized queries explained in Section 6.1.

We compare the join-based approach of *max* against the group-based approach (possible only in SPARQL 1.1), both in qualified (i.e, within a class definition) and unqualified (i.e, as a global property) variants. Furthermore, we use variants without the LIMIT clause to determine the cost of generating witnesses for all violations. Figure 7 show the results of this comparison when varying the *max*(*n*) threshold between 1 and 9, using a dataset with 1M triples. The overall results are the same for Sesame and Virtuoso, but there are subtle differences.

Without a LIMIT clause, the join based approach shows exponential growth, roughly tripling the cost when increasing the threshold by one. In contrast, the group based approach (Optimized) has a constant cost, since the the cost of creating groups does not depend on the threshold value, which in turn can be checked in constant time. In all these cases, the unqualified variant takes about twice as much time than the qualified variant, given the larger set of candidates to consider. For Sesame, the group-based variant always outperforms the join-based variant without limits, while for Virtuoso this is only true for threshold values greater than 4, as joins are faster, but grouping slower.



(a) Sesame



(b) Virtuoso

Figure 7: Cost of Max Constraints for 1M triples

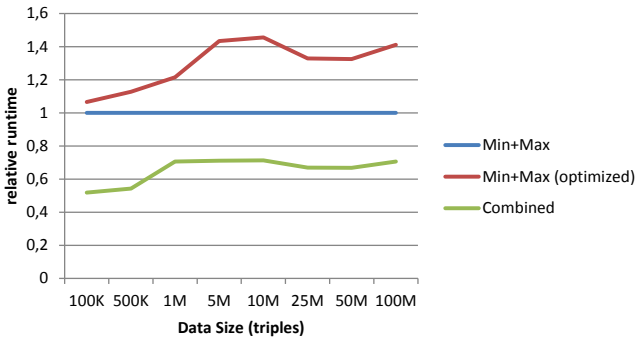
When we consider LIMIT clauses, we gain a number of insights on the intricacies of optimizing queries for validation: As long as the constraint is violated, the join-based approach now clearly outperforms the group-based approach, which is not at all affected by the LIMIT clause. Grouping always has to be performed over the whole dataset, while the optimizers of both systems can perform an early stop on joins, similar to the optimizations possible in SQL [7]. When there are no violations and thus no results, the entire dataset needs to be considered, explaining why at larger thresholds the benefit of LIMIT disappears. We performed this analysis on different dataset sizes, yielding the same overall results.

We also evaluated additional triple storage schemes (or indexes), but we did not determine any speedup: Nearly all queries use subject joins and predicate selections, which fits well with the default storage of both systems.

Overall, grouping is a well suited strategy if it is unclear if the constraint holds. Joins with limits work well if the threshold is small and a constraint is expected to not hold.

Intra-Constraint and Inter-Constraint Optimizations.

In our last experiment, we provide a first insight into optimization spanning multiple queries and constraints, enabling us to reduce the number of times the data needs to be accessed. The grouping-based optimization lends itself well for composition, as the same aggregated values can be checked against multiple constraints. We investigate the tradeoffs by following the example in Section 6.2, comparing the runtime



(a) Sesame

(b) Virtuoso

Figure 8: Query Combination Optimization

of the parts of a *total* constraint ($max(1)$ and $min(1)$) in their optimized and non-optimized form against a combined query. As we show in Figure 8, the runtime of the combined eventually falls below the sum of the runtimes of the individual queries. Furthermore, it shows that grouping comes at a cost: For Sesame, the difference is moderate, since the existential check needed for $min(1)$ can be performed faster than a grouping, for Virtuoso the cost of grouping is prohibitive for small scales.

8. CONCLUSION

In this paper, we presented the methodology as well a working system to validate an expressive RDF constraint language using standard SPARQL queries in a "bulk" fashion. Using pure SPARQL is not only conceptually desirable, but also allows the validation of such constraints without having to modify the often loosely coupled and heterogeneous RDF storage systems present in the Linked Open Data environment. The results of our evaluation on state-of-the-art SPARQL databases and a well-established RDF dataset show that such a SPARQL-based validation is feasible with acceptable cost, matching typical loading times. We did, however, identify certain classes of constraints which are expensive to validate using SPARQL 1.0. In turn, we investigated several direction on how to overcome this challenge. Within the scope of individual queries, using a number of SPARQL 1.1 features improves scalability. In the near future, we plan to cross-validate our results on different datasets (e.g., DBPedia, Linked Sensor Data or Bio2RDF) and other RDF constraint languages like RDF Shapes.

Our current work opens up several avenues of further research: Considering that currently several 100s of queries need to be run in order to validate a single RDD and each of these queries has to touch the full dataset, sharing as many validation steps as possible seems to be very promising. On the language side it is currently not clear if the expressiveness of SPARQL 1.1 is sufficient for this purpose, in particular with the flexibility and composability of GROUP BY. On a more conceptual side, we want to understand how far this combination can go and if we can determine a lower limit. Such a limit ties also into an evaluation of the expressive power and cost of some of the competing proposals (such as RDF shapes) in order to identify a "sweet spot" of expressive power and validation cost over these proposals.

Acknowledgements.

The implementation of the RDD prototype was supported by Deutsche Forschungsgemeinschaft grant LA 598/7-1.

9. REFERENCES

[1] OpenRDF Sesame. `http://www.openrdf.org/`.

[2] R2RML: RDB to RDF Mapping Language. `http://www.w3.org/TR/r2rml/`.

[3] Rdf 1.1 semantics. `http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/`.

[4] Rdf data shapes working group charter. `http://www.w3.org/2014/data-shapes/charter`.

[5] RDF Specification Overview (W3C). `http://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/`.

[6] Stardog. `http://Stardog.com/`.

[7] Michael J. Carey and Donald Kossmann. Reducing the Braking Distance of an SQL Query Engine. In *Proceedings of the 24rd International Conference on Very Large Data Bases, VLDB*, pages 158–169, 1998.

[8] Dimitris Kontokostas et al. Test-driven Evaluation of Linked Data Quality. In *Proceedings of the 23rd International World Wide Web Conference, WWW*, pages 747–758, 2014.

[9] Georg Lausen, Michael Meier, and Michael Schmidt. SPARQLing Constraints for RDF. In *Proceedings of the 11th International Conference on Extending Database Technology, EDBT*, pages 499–509, 2008.

[10] Eric Prud'hommeaux, Jose Emilio Labra Gayo, and Harold Solbrig. Shape Expressions: an RDF Validation and Transformation Language. In *SEM '14: Proceedings of the 10th International Conference on Semantic Systems*, pages 32–40, 2014.

[11] Arthur Ryman, Arnaud Le Hors, and Steve Speicher. OSLC Resource Shape: A Language for Defining Constraints on Linked Data. In *Proceedings of the WWW2013 Workshop on Linked Data on the Web, LDOW*, 2013.

[12] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. SP2 Bench: A SPARQL Performance Benchmark. In *Proceedings of the 25th International Conference on Data Engineering, ICDE*, pages 222–233, 2009.

[13] Michael Schmidt and Georg Lausen. Pleasantly Consuming Linked Data with RDF Data Descriptions. In *Proceedings of the Fourth International Workshop on Consuming Linked Data, COLD*, 2013.

[14] Michael Schmidt and Georg Lausen. Pleasantly Consuming Linked Data with RDF Data Descriptions. `http://arxiv.org/abs/1307.3419`, 2013.

[15] Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL Query Optimization. In *Proceedings of the 13th International Conference on Database Theory, ICDT*, pages 4–33, 2010.

[16] W3C. RDF Validation Workshop, Practical Assurances for Quality RDF Data. `http://www.w3.org/2012/12/rdf-val/`, 2013.

Peer-to-Peer Semantic Integration of Linked Data

Mirko Michele Dimartino
London Knowledge Lab
Birkbeck, University of London
mirko@dcs.bbk.ac.uk

Andrea Calì*
London Knowledge Lab
Birkbeck, University of London
andrea@dcs.bbk.ac.uk

Alexandra Poulovassilis
London Knowledge Lab
Birkbeck, University of London
ap@dcs.bbk.ac.uk

Peter Wood
London Knowledge Lab
Birkbeck, University of London
ptw@dcs.bbk.ac.uk

ABSTRACT

We propose a framework for peer-based integration of linked data sets, where the semantic relationships between data at different peers are expressed through mappings. We provide the theoretical foundations for such a setting and we devise an algorithm for processing graph pattern queries, discussing its complexity and scalability.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*distributed databases, query processing*

General Terms

Algorithms

Keywords

Rewriting, SPARQL, RDF, Peer-to-Peer, Semantic Web

1. INTRODUCTION

In recent years the World Wide Web has gradually expanded from a simple network of hyper-linked documents to a more complex structure where both documents and data are easily published, consumed and reused. As a result of this rapid transformation, new techniques are required in order to integrate these heterogeneous data into a single global data space, the so-called Linked Open Data (LOD) cloud [2], building on Web infrastructure (URIs and HTTP), Semantic Web standards (such as the Resource Description Framework (RDF) and RDF Schema (RDFS)), and vocabularies. These practices have led to the creation of a worldwide database covering a wide range of domains, varying in type from personal and corporate to statistical and scientific data and reviews [3]. Ideally, users should be able to access an open, global data space with an approach similar to how a local database is queried today, in order to obtain more extensive answers as new data sources appear on the Web. However, linked data poses challenges inherent to integrating and querying highly heterogeneous and distributed data, so the above-stated vision has yet to be entirely realised.

In the LOD environment, it is common for several datasets to describe overlapping domains, often using different standards of data modelling and naming. Therefore a global ontological conceptualisation is impracticable and a more flexible approach for semantic integration is needed. This represents a major research challenge for the web of data.

To cope with these limitations, some work in the literature addresses the problem of answering SPARQL queries over disparate sources, proposing new SPARQL rewriting algorithms that entail semantic mappings between RDF databases [18, 10, 19, 20]. These techniques address query rewriting from one source to another, while the LOD cloud is a dynamic environment that comprises several data sources with arbitrary mapping topologies in a peer-to-peer fashion. In fact, in this scenario, an implementation of the existing rewriting algorithms may lack computability, especially in the presence of mapping cycles. The open problem is then developing new data integration techniques to support SPARQL query answering over several heterogeneous RDF sources whose semantic mappings have arbitrary topologies.

The following example considers a typical use-case of querying Linked Data.

Example 1. Figure 1 illustrates an RDF graph containing triples from three different sources. Sources 1 and 2 contain data about films, while Source 3 describes people and their properties. We can see that URIs representing the same entities (e.g., `DB1:Spiderman` and `DB2:Spiderman2002`, for the film *Spiderman*) are linked by the built-in OWL property `sameAs`[1], which states that the linked URIs represent the same real-world entity (best practices for `owl:sameAs` are given in [15]). It is clear that there is a semantic equivalence mapping between URIs linked by `sameAs`. We can also see that there is a semantic equivalence mapping between pairs of triples of the form `(a starring _z)` and `(_z artist b)` in Source 1 and triples of the form `(a actor b)` in Source 2; both represent the relationship that "*actor b acted in the film*

*Also affiliated to the Oxford-Man Institute of Quantitative Finance, University of Oxford, `andrea.cali@oxford-man.ox.ac.uk` .

[1] `http://sameas.org`

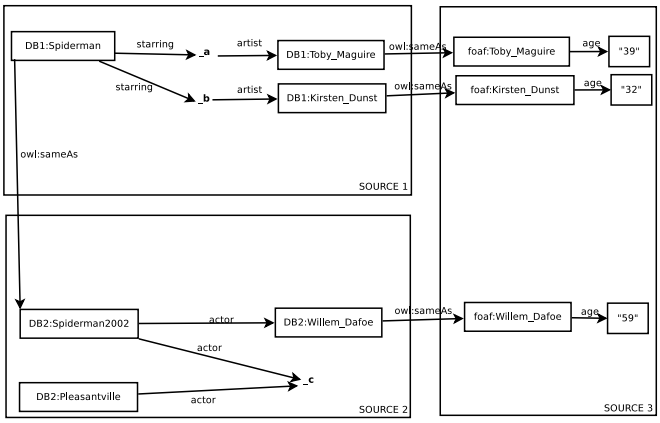


Figure 1: Example of an RDF graph from three data sources.

a". Now assume that a user poses the following SPARQL[2] query:

```
SELECT ?x ?y
WHERE { DB1:Spiderman starring ?z . ?z artist ?x .
        ?x age ?y }
```

This query returns an empty result on the data of Figure 1, since the `sameAs` property is missing from the query, and SPARQL does not automatically exploit semantic mappings between RDF resources. As stated above, adopting existing rewriting techniques to entail the semantic mappings is impractical for this scenario, since there are more than two RDF sources and the mapping topologies are arbitrary. In this regard, we propose a decentralised, easily extensible, RDF-oriented peer data management system. We provide the theoretical foundations for such a setting and we devise an algorithm for processing graph pattern queries, discussing its complexity and scalability.

Related work: As previously stated in this section, few papers in the literature deal with the challenges of answering SPARQL queries over data, leveraging the semantic mappings between similar vocabularies. For instance, the work in [18] presents a query rewriting algorithm over virtual SPARQL views; and, similarly, [10] introduces a SPARQL rewriting algorithm based on the encoding of rules for RDF patterns, involving entity equivalence functions for the semantics of the property `owl:sameAs`. In [19, 20] Makris et al. instead adopt Description Logic rules between overlapping OWL ontologies. These works address query answering over two-tiered architectures, while we wish to explore the most general case, where the number of sources and the mapping topologies are arbitrary.

Several peer-to-peer systems for RDF datasources can be found in the literature. For instance, in [5, 6] the authors describe a distributed RDF metadata storage, querying and subscription service as a structured P2P network. Similarly, work in [23] proposes routing strategies for RDF-based P2P networks. Similar work can be found in [21, 22, 17]. However, all of these papers focus on technical issues relating to peer networks (such as efficiency of query routing, network traffic load, etc.), and so leave a gap between the RDF data model and peer-to-peer semantic integration.

Paper outline: The paper is organised as follows. In Section 2 we present our new framework for RDF peer-to-peer integration. Then, in Section 3 we explore the query answering problem under RDF Peer Systems and we propose a query answering algorithm that terminates in polynomial time. In Section 4 we explore query rewriting techniques in our setting, showing that our mapping rules are not first-order-rewritable. We conclude with a discussion in Section 5.

[2] `http://www.w3.org/TR/rdf-sparql-query/`

2. THE FRAMEWORK

In this section, we introduce our framework for peer-to-peer RDF semantic data integration. We present a new peer mapping language suitable for the RDF data model that extends the goals of relational P2P models to achieve semantic integration in accordance with Linked Data technologies. Our goal is to leverage the techniques for specifying semantic mappings between RDF sources, extending them beyond a two-tiered architecture. In our framework, each peer is represented by its peer schema, comprising the set of URIs adopted in the peer to model data. Integration is achieved by means of mappings between these sets of URIs. To formally specify the problem of query answering, we generalize the notion of *certain answers* [1] to our context.

2.1 Graph pattern queries

To formalise the problem, we introduce the notion of graph pattern queries for RDF databases (for details of RDF formalisation, see [14]). Assume there are pairwise disjoint infinite sets I, B, and L (IRIs [11], Blank nodes, and Literals, respectively). A triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an RDF triple. In this triple, s is the *subject*, p the *predicate*, and o the *object*. Also, we assume the existence of an infinite set of variables V disjoint from $(I \cup B \cup L)$. An *RDF database* is then a set of RDF triples.

A *graph pattern* is defined recursively as follows:

1. A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a *graph pattern*. Specifically, it is a *triple pattern*.
2. If GP_1 and GP_2 are graph patterns, then the expression $(GP_1 \text{ AND } GP_2)$ is a *graph pattern*.

We denote by $var(GP)$ the set of variables $V_{GP} \subseteq V$ that appear in the graph pattern GP.

A *graph pattern query* Q of arity n is of the form

$$q(\mathbf{x}) \leftarrow GP$$

where GP is a graph pattern, and $\mathbf{x} = x_1, \ldots, x_n \in var(GP)$ denote the *free variables* of the query. All the elements in $var(GP)$ that are not free variables denote the *existentially quantified variables* of the query.

In order to define the semantics of graph pattern queries, we introduce some terminology from [24, 4] for the evaluation of a graph pattern over an RDF database.

A *mapping* μ from V to $(I \cup B \cup L)$ is a partial function $\mu : V \to (I \cup B \cup L)$. The domain of μ, denoted by $dom(\mu)$, is the subset of V where μ is defined. Given a mapping μ and a variable $v \in dom(\mu)$ we denote by $\mu(v)$ the value in $(I \cup B \cup L)$ obtained by applying the function μ to v. Also,

abusing notation, for a triple pattern t we denote by $\mu(t)$ the triple obtained by replacing the variables in t according to μ. Two mappings μ_1 and μ_2 are compatible when for all $x \in dom(\mu_1) \cap dom(\mu_2)$, it is the case that $\mu_1(x) = \mu_2(x)$, i.e. when $\mu_1 \cup \mu_2$ is also a mapping.

Let Ω_1 and Ω_2 be sets of mappings. Then the *join* of Ω_1 and Ω_2 is defined as follows [24, 4]:

$$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1, \mu_2 \text{ are compatible mappings}\}$$

The semantics of graph patterns is then defined by a function $[\![\cdot]\!]_D$ over a set of RDF triples D, also called an RDF graph or RDF database, which takes a graph pattern as input and returns a set of mappings that matches the database D [4, 24].

Definition 1. (Evaluation of a graph pattern). The evaluation of a graph pattern GP over an RDF dataset D, denoted by $[\![GP]\!]_D$, is defined recursively as follows:

1. If GP is a triple pattern t, then $[\![GP]\!]_D = \{\mu \mid dom(\mu) = var(t) \text{ and } \mu(t) \in D\}$
2. If GP is of the form (GP_1 AND GP_2), then $[\![P]\!]_D = [\![GP_1]\!]_D \bowtie [\![GP_2]\!]_D$.

We are ready to define the semantics of graph pattern queries. We denote by Q^D the set of n-tuples returned by the evaluation of the graph pattern query Q of arity n over the dataset D. We define the semantics of Q^D as follows:

$$Q^D := \{(\mu(x_1), \ldots, \mu(x_n)) \mid \mu \in [\![GP]\!]_D \text{ and } \mu(x_1), \ldots, \mu(x_n) \in (I \cup L)\},$$

where GP is the graph pattern of the query Q and $x_1 \ldots x_n$ are the free variables of Q.

As we can see from the above definition, tuples containing elements in B (blank nodes) are not returned from the evaluation of the query. Blank nodes are used in the RDF triples as placeholders for unknown resources [16]; in other words, they denote variables which may take values in the set of IRIs and literals ($I \cup L$). In this regard, a graph pattern query retrieves only full information, dropping all the tuples containing partial information.

In fact, we define the semantics of blank nodes so as to be equivalent to the semantics of *labelled nulls* in the relational model, which are placeholders for unknown values and are not included in query results. For completeness, we also define a semantics of graph pattern queries which does include blank nodes in the result set. This semantics will be used later on to exploit the expressiveness of equivalence mappings in our RDF Peer System. We denote this semantics by Q^{*D}, where

$$Q^{*D} := \{(\mu(x_1), \ldots, \mu(x_n)) \mid \mu \in [\![GP]\!]_D\}.$$

Note that the graph pattern query language can be seen as a "conjunctive fragment" of SPARQL, so a graph pattern query can always be translated to a conjunctive SPARQL query and vice versa.

2.2 RDF Peer Systems

An *RDF Peer System* (RPS) $\mathcal{P}$ constitutes a set of peers and a set of mappings that specify the semantic relationships between peers. Formally, an RPS $\mathcal{P}$ is defined as a tuple $\mathcal{P} = (\mathcal{S}, G, E)$, where:

- $\mathcal{S}$ is the set of the *peer schemas* in $\mathcal{P}$. Each peer schema $S \in \mathcal{S}$ is simply the set of all the constants $u \in I$ adopted by the corresponding peer to describe data in the form of RDF triples. Informally, the schema of a peer is a subset of I (the set of all the IRIs in Linked Data) comprising only the IRIs adopted by the peer. Two peer schemas then need not be disjoint sets: this is in accordance with real Linked Data sources, where two different RDF databases may share some IRIs in the RDF triples.
- G is a set of *graph mapping assertions*, each of which is an expression of the form $Q \rightsquigarrow Q'$, where Q and Q' are *graph pattern queries* of the same arity, expressed over the schemas S and S', respectively, of two peers in $\mathcal{P}$. Formally, the graph pattern GP in the query Q contains triple patterns from $(S \cup L \cup V) \times (S \cup V) \times (S \cup L \cup V)$, and the graph pattern GP' in the query Q' contains triple patterns from $(S' \cup L \cup V) \times (S' \cup V) \times (S' \cup L \cup V)$.
- E is a set of *equivalence mappings* of the form $c \equiv_e c'$, where $c \in S$ and $c' \in S'$ and $S, S' \in \mathcal{S}$.

2.3 Semantics of RDF Peer Systems

We assume that we are given an instance of the data stored in the peers in the form of a set of RDF triples for each peer in the system. Formally, for each peer defined by its schema $S \in \mathcal{S}$ in $\mathcal{P}$, we have a database d, that is, a set of triples $(s, p, o) \in (S \cup B) \times S \times (S \cup B \cup L)$. Consequently, a *stored database* D of an RPS $\mathcal{P}$ is the union of all the peer databases d of all the peers in $\mathcal{P}$. Then, a *peer-to-peer database* of an RPS $\mathcal{P}$ is simply an arbitrary RDF database containing triples $(s, p, o) \in (S_1 \cup \cdots \cup S_n \cup B) \times (S_1 \cup \cdots \cup S_n) \times (S_1 \cup \cdots \cup S_n \cup B \cup L)$, where $S_1, \ldots, S_n \in \mathcal{S}$ are the peer schemas in $\mathcal{P}$.

We also denote by $subjQ(c)$, $predQ(c)$ and $objQ(c)$ three special *graph pattern queries*:

- $subjQ(c) := q(x_{pred}, x_{obj}) \leftarrow (c, x_{pred}, x_{obj})$
- $predQ(c) := q(x_{subj}, x_{obj}) \leftarrow (x_{subj}, c, x_{obj})$
- $objQ(c) := q(x_{subj}, x_{pred}) \leftarrow (x_{subj}, x_{pred}, c)$

where $c \in (S_1 \cup \cdots \cup S_n \cup L)$.

The evaluation of $subjQ(c)$ over an RDF dataset is the set of pairs of the form $(t.pred, t.obj)$ containing the *predicate* and *object* of all triples in the dataset where the constant c occurs as the *subject*. The queries $predQ(c)$ and $objQ(c)$ are defined similarly, with the constant c now occurring as the *predicate* and the *object* of an RDF triple, respectively.

Below we give formal definitions for a *solution* for an RPS $\mathcal{P}$ and the set of *certain answers* for a query posed against $\mathcal{P}$. Informally, a peer-to-peer database is a solution of an RPS $\mathcal{P}$ if it contains the stored database of $\mathcal{P}$, as well as all triples inferred by the mappings of $\mathcal{P}$. The certain answers to a query against $\mathcal{P}$ are those which appear in all possible solutions of $\mathcal{P}$.

Definition 2. A peer-to-peer database I is said to be a *solution* for an RPS $\mathcal{P}$ based on a stored database D if:

1. For every stored database $d \in D$, we have that $d \subseteq I$.
2. For every graph mapping assertion in G of the form $Q \rightsquigarrow Q'$, we have that $Q^I \subseteq Q'^I$.
3. For every equivalence mapping in E of the form $c \equiv_e c'$, all of the following hold:

$$subjQ(c)^{*I} = subjQ(c')^{*I}$$
$$predQ(c)^{*I} = predQ(c')^{*I}$$
$$objQ(c)^{*I} = objQ(c')^{*I}$$

Definition 3. We define the *certain answers* $ans(q, \mathcal{P}, D)$ to an arbitrary graph pattern query q of arity n, based on a stored database D of an RPS $\mathcal{P}$, as the set of n-tuples $\mathbf{t}$ of constants in $(S_1 \cup \cdots \cup S_n \cup L)$ such that, for every peer-to-peer database I that is a solution for the system $\mathcal{P}$ based on D, we have that $\mathbf{t} \in q^I$.

The query answering problem is defined as follows: given an RPS $\mathcal{P}$, a stored database D and a graph pattern query q, find the certain answers $ans(q, \mathcal{P}, D)$.

3. QUERY ANSWERING

To evaluate the complexity of the query answering problem we show that the problem of finding $ans(q, \mathcal{P}, D)$ is subsumed by CQ answering in data exchange for the relational model. Specifically, we show that a solution of an RPS can be seen as a solution of a special data exchange setting.

A data exchange setting is defined by a *source* relational alphabet $\boldsymbol{S}$, a *target* relational alphabet $\boldsymbol{T}$, a set Σ_{st} of *source-to-target* dependencies and a set Σ_t of *target* dependencies. Instances over $\boldsymbol{S}$ are called *source* instances, while instances over $\boldsymbol{T}$ are called *target* instances. Given a source instance I, the problem is to find a solution J over the target schema such that $I \cup J$ satisfies the source-to-target dependencies and J satisfies the target dependencies [12].

For a given RPS $\mathcal{P} = (\mathcal{S}, G, E)$ and a stored database D for $\mathcal{P}$, we can define a data exchange setting such that a solution for the data exchange problem is a solution for $\mathcal{P}$. We define the relational alphabets $\boldsymbol{R_s} := \{t_s, r_s\}$ and $\boldsymbol{R_t} := \{t_t, r_t\}$, where t_s and t_t are ternary relational symbols and r_s and r_t are unary relational symbols. These relational alphabets describe the RDF triples (t_s) and the identified resources (r_s) stored by the peers in D, and the RDF triples (t_t) and the identified resources (r_t) inferred in a peer-to-peer database of $\mathcal{P}$.

Given a relational alphabet A, we denote by $\mathcal{L}_A$ the set of function-free first-order logic (FOL) formulas whose relation symbols are in A and whose constants are in $(I \cup B \cup L)$. Then, given a graph pattern query Q of the form $q(\mathbf{x}) \leftarrow GP$ we can define the term $Q_{body}(\mathbf{x}, \mathbf{y})$ as the conjunction of the atoms in $\mathcal{L}_{\boldsymbol{R_t}}$ representing triple patterns in GP, where $\mathbf{x} = x_1, \ldots, x_n \in var(GP)$ are the free variables, and $\mathbf{y} = y_1, \ldots, y_m \in var(GP)$ are the existentially quantified variables in Q. For example, given the following graph pattern query

$$Gfather \;:=\; q(x_1, x_2) \leftarrow (x_1, father, y) \text{ AND } (y, father, x_2),$$

$Gfather_{body}(\mathbf{x}, \mathbf{y})$ is the conjunction of atoms

$$t_t(x_1, father, y) \wedge t_t(y, father, x_2),$$

where $\mathbf{x} = x_1, x_2$ and $\mathbf{y} = y$. In this regard, evaluating a graph pattern query Q over an instance of a RPS is equivalent to evaluating the following conjunctive query (CQ) over an interpretation of the relations in $\boldsymbol{R_t}$:

$$\{\mathbf{x} \mid \exists \mathbf{y}\ Q_{body}(\mathbf{x}, \mathbf{y}) \wedge r_t(x_1) \wedge \cdots \wedge r_t(x_n)\}.$$

Given an RPS $\mathcal{P} = (\mathcal{S}, G, E)$, we are now ready to define a data exchange setting whose solution (seen as an RDF database) is also a solution for $\mathcal{P}$. The relational alphabets $\boldsymbol{R_s}$ and $\boldsymbol{R_t}$ are the source and target relational alphabets of the data exchange setting. The source-to-target dependencies express the semantics of item 1 in Definition 2, which states that the peer-to-peer database of the RPS must contain all the triples in the stored database. They are of the form:

$$\forall x \forall y \forall z\ t_s(x, y, z) \rightarrow t_t(x, y, z),$$
$$\forall x\ r_s(x) \rightarrow r_t(x).$$

The target dependencies express the semantics of the graph mapping assertions and the equivalence mappings. For each graph mapping assertion $Q \rightsquigarrow Q' \in G$, we have the dependency

$$\forall \mathbf{x} \exists \mathbf{y}\ Q_{body}(\mathbf{x}, \mathbf{y}) \wedge r_t(x_1) \wedge \cdots \wedge r_t(x_n) \rightarrow \exists \mathbf{z}\ Q'_{body}(\mathbf{x}, \mathbf{z}),$$

and for each equivalence mapping $c \equiv_e c' \in E$, we have the dependencies

$$\forall y \forall z\ t_t(c, y, z) \rightarrow t_t(c', y, z),$$
$$\forall y \forall z\ t_t(c', y, z) \rightarrow t_t(c, y, z),$$
$$\forall x \forall z\ t_t(x, c, z) \rightarrow t_t(x, c', z),$$
$$\forall x \forall z\ t_t(x, c', z) \rightarrow t_t(x, c, z),$$
$$\forall x \forall y\ t_t(x, y, c) \rightarrow t_t(x, y, c'),$$
$$\forall x \forall y\ t_t(x, y, c') \rightarrow t_t(x, y, c).$$

In this regard, query answering under an RPS is equivalent to the problem of CQ answering in data exchange, under the special data exchange setting defined above.

The set of certain answers in the data exchange problem is computed by evaluating queries over the so-called *universal solution* of the data exchange setting. To generate a universal solution, a source database is "chased" using the set of dependencies. Each step of the *chase* "extends" the database so that the chosen dependency is satisfied. For instance, given a dependency $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$ and a mapping h (from the variables in $\phi(\mathbf{x})$ to constants) for which the dependency is not satisfied, the chase step generates new facts in the target instance in order to satisfy the dependency. The new facts are generated by: (a) extending h to h' such that each existentially quantified variable in $\psi(\mathbf{x}, \mathbf{y})$ is assigned a freshly created constant, a *labelled null*, followed by: (b) taking the image of the atoms of ψ under h' (see [12], Section 3 for more details of the chase procedure).

In our specific data exchange setting, there are no atoms of type $r_t(x)$ in the head of any dependency such that the

variable x is existentially quantified. Therefore, the set of IRIs and literals remains constant during the chase procedure. Thus, the chase generates new blank nodes as labelled nulls. Without loss of generality, we will use the term *newly created blank nodes* when we want to denote labelled nulls.

In an RPS $\mathcal{P} = (\mathcal{S}, G, E)$, only dependencies in G contain existentially quantified variables in the body, therefore they are the only dependencies for which the chase may generate new constants, i.e., newly created blank nodes. Since newly created blank nodes cannot trigger any of these rules, the chase sequence is then bounded by a finite number of steps. This leads to the following theorem:

THEOREM 1. *The problem of finding all certain answers $ans(q, \mathcal{P}, D)$ to an arbitrary graph pattern query q, for a given RDF Peer System $\mathcal{P}$ and a stored database D, has PTIME data complexity.*

Due to lack of space, we omit a formal proof of the theorem, which will appear in an extended version of this paper.

In data exchange, the set of certain answers of a query is then computed by evaluating the query over the universal solution and eliminating all the tuples in the result that contain labelled nulls. In our RDF model, the semantics of a graph pattern query Q^D eliminates all the answer tuples containing blank nodes, so we can generate the certain answers by simply evaluating the graph pattern query over the universal solution. The algorithm that computes the certain answers, Algorithm 1, is listed in the Appendix.

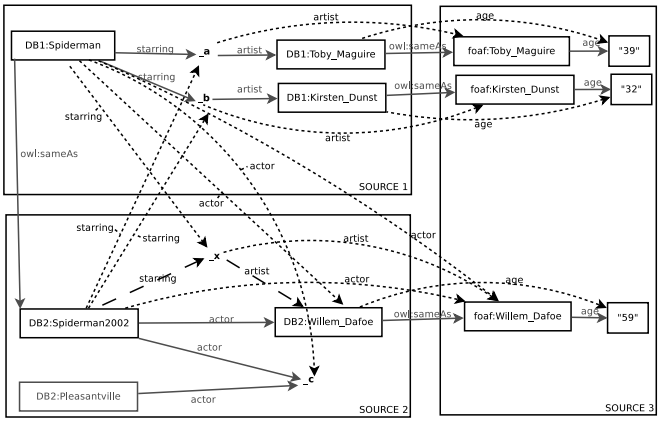


Figure 2: RDF graph of a universal solution for the peer system. Dotted arrows and dashed arrows represent triples inferred by the equivalence mappings and the graph mapping assertions, respectively.

Example 2. Let us consider again the RDF sources of Example 1. We define an RPS $\mathcal{P} = (\mathcal{S}, G, E)$ as follows:

- $\mathcal{S} := \{S_1, S_2, S_3\}$ where S_i is the set of IRIs in the i^{th} source. For example, $S_2 :=$ {DB2:Spiderman2002, DB2:Willem_Dafoe, DB2:Pleasantville, actor} (in this case, we consider `owl:sameAs` triples stored in Source 1 and Source 3).
- G is a single graph mapping assertion of the form $Q_2 \rightsquigarrow Q_1$, where:
  - $Q_1 := q(x, y) \leftarrow (x, starring, z)$ AND $(z, artist, y)$,
  - $Q_2 := q(x, y) \leftarrow (x, actor, y)$.
- E contains an equivalence mapping $c \equiv_e c'$ for each triple of the form $(c, sameAs, c')$.

Figure 2 illustrates an RDF database which is a universal solution for $\mathcal{P}$. Let us consider again the SPARQL query used in Example 1. Now, evaluating the query over the universal solution, we obtain the result in Listing 1. It is important to observe that the user poses a query over Sources 1 and 3 but retrieves additional information also from Source 2 in a transparent way. The RPS, in fact, not only captures the semantics of the `owl:sameAs` property, but also performs integration of similar sources in order to return additional answers to the user. This integration can be performed dynamically as new data sources appear, and requires no input from the user.

```
#Query

SELECT ?x ?y
WHERE { DB1:Spiderman starring ?z .
        ?z artist ?x .
        ?x age ?y }

#Result
DB1:Toby_Maguire "39"
foaf:Toby_Maguire "39"
DB1:Kirsten_Dunst "32"
foaf:Kirsten_Dunst "32"
DB2:Willem_Dafoe "59"
foaf:Willem_Dafoe "59"

#Result without redundancy

DB1:Toby_Maguire "39"
DB1:Kirsten_Dunst "32"
DB2:Willem_Dafoe "59"
```

Listing 1: SPARQL query over the universal solution.

4. QUERY REWRITING

The chase algorithm is a useful tool for query answering, however materialising the universal solution for an RDF peer system may be impractical in the Linked Data scenario due to the large volumes of data involved.

A more efficient approach would involve a rewriting of the original query that, when evaluated directly over the sources, returns the set of certain answers. In other words, given a stored database D, a query q and the set Σ of TGDs that entail the peer mappings, we want to compute a rewriting of q based on Σ, named q_Σ, such that ${q_\Sigma}^D = q^J$, where $J = chase(D, \Sigma)$ is the universal solution for the peer system. In this case q_Σ is a *perfect rewriting* of q since it preserves a sound and complete answer of the original query based on the extensional database D and the "*ontological theory*" Σ.

Several works have addressed query rewriting under TGDs. [8, 9] introduced sets of TDGs, namely *sticky* sets, that enjoy the property of being *FO-rewritable*, i.e., for every query that needs to be evaluated under such dependencies it is

possible to compute a first-order query as a perfect rewriting. The algorithm has as input a *Boolean CQ* q, a database D and a sticky set of TGDs Σ, and it outputs "Accept" if $chase(D, \Sigma) \models q$. We recall that the two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent.

Stickiness is a sufficient syntactic condition that ensures the so-called sticky property of the chase, which is as follows. For every instance D, assume that during the chase of D under a set Σ of TGDs, we apply a TGD σ that has a variable V appearing more than once in its body; assume also that V maps (via homomorphism) onto the constant z, and that by virtue of this application the atom a is generated by the chase step. In this case, for each atom b in the body of σ, we say that a is derived from b. Then, we have that z appears in a, and in all atoms resulting from some chase derivation sequence starting from a, "sticking" to them (hence the name "sticky sets of TGDs").

The formal definition of sticky sets of TGDs, given in [9], is an efficient testable condition involving variable marking.

Definition 4. Consider a set Σ of TGDs over a relational alphabet $\mathcal{R}$. A *position* $r[i]$ in $\mathcal{R}$ is identified by the predicate $r \in \mathcal{R}$ and its i-th argument (or attribute). We mark the variables that occur in the body of the TGDs of Σ according to the following procedure. First, for each TGD $\sigma \in \Sigma$ and for each variable V in $body(\sigma)$, if there exists an atom a in $head(\sigma)$ such that V does not appear in a, then we mark each occurrence of V in $body(\sigma)$. Given a predicate symbol r, $r[i]$ identifies its i-th argument (or attribute). Now, we apply exhaustively (i.e., until a fixpoint is reached) the following step: for each TGD $\sigma \in \Sigma$, if a marked variable in $body(\sigma)$ appears at a position π, then for every TGD $\sigma' \in \Sigma$ (including the case $\sigma' = \sigma$), we mark each occurrence of the variables in $body(\sigma')$ that appear in $head(\sigma')$ at the same position π. We say that Σ is *sticky* if and only if there is no TGD $\sigma \in \Sigma$ such that a marked variable occurs in $body(\sigma)$ more than once.

Given an RPS $\mathcal{P} = (\mathcal{S}, G, E)$, the set E of TGDs for equivalence mappings enjoys the sticky property of the chase, as well as linearity. Graph mapping assertions in G do not preserve the same property. We can easily show this by applying the variable marking on the following example of a graph mapping assertion:

$$\forall x \forall y \exists z\ t_t(x, A, \hat{z}) \wedge t_t(\hat{z}, B, y) \wedge r_t(x) \wedge r_t(y) \rightarrow t_t(x, C, y),$$

where A,B and C are URIs. Here, applying the variable marking results in the variable z appearing more than once in the body of the TGD. This violates the stickiness condition.

It is important to observe that the set Σ of TGDs in an RPS is neither sticky, nor linear, nor weakly-acyclic [12], nor guarded [7], nor weakly-guarded [7]. In fact our sets of TGDs are incomparable to the above known classes of TGDs under which query answering is decidable. ■

In [13] the authors propose the query rewriting algorithm *TGD-rewrite* which takes as input a BCQ and a set of TGDs each with just one head-atom, containing at most one existentially quantified variable, which occurs only once. The algorithm generates a union of BCQs (i.e., a FO-query) which is a perfect rewriting of the original query. It is shown that query answering under this setting is LOGSPACE-equivalent to query answering under (general) TGDs, and thus the result holds for arbitrary TGDs. The algorithm *TGD-rewrite* guarantees termination under linear, sticky or sticky-join sets of TGDs (a generalisation of sticky TGDs and linear TGDs).

PROPOSITION. 2. *Given an RPS $\mathcal{P} = (\mathcal{S}, G, E)$, a stored database D and a Boolean query q, if G is either linear, sticky, or sticky-join, then we can generate a FO-query $q_{\mathcal{P}}$, such that ${q_{\mathcal{P}}}^D = q^J$, where J is the universal solution for $\mathcal{P}$ based on D.*

Example 3. Consider again the RPS in Example 2. The set G of graph mapping assertions is linear, hence, following from Proposition 2, we can generate an FO-rewriting of a given Boolean query to entail the mapping assertions of the RPS. Listing 2 shows an example rewriting based on the SPARQL query and the RDF stored database shown in the introduction. To compute the set of certain answers of the given query, first we generate the set of all the possible 2-tuples from the stored database. Then we iterate over each 2-tuple t and decide whether or not t is in the set of certain answers, by substituting t into the SPARQL query to obtain a Boolean query (note that this is a polynomial-time reduction of the problem, since there are polynomially many k-tuples from the source database). Each Boolean query can be rewritten as an FO-query according to the mapping assertions in the RPS. In our case, the rewriting generates a union of SPARQL queries. Due to lack of space, we show only one possible step of the query rewriting, which makes use of the dependency

$$\forall y \forall z\ t_t(foaf{:}Toby\_Maguire, y, z) \rightarrow t_t(DB1{:}Toby\_Maguire, y, z)$$

to rewrite the triple pattern `(DB1:Toby_Maguire age "39")`.

```
#Original query

SELECT ?x ?y
WHERE { DB1:Spiderman starring ?z .
?z artist ?x .
?x age ?y }

#Boolean query:
#ask if the tuple (DB1:Toby_Maguire,"39")
#is in the query result.

ASK { DB1:Spiderman starring ?z .
?z artist DB1:Toby_Maguire .
DB1:Toby_Maguire age "39" }

false

#Rewritten query

ASK {{ DB1:Spiderman starring ?z .
?z artist DB1:Toby_Maguire .
DB1:Toby_Maguire age "39" }
UNION
{ DB1:Spiderman starring ?z .
?z artist DB1:Toby_Maguire .
foaf:Toby_Maguire age "39" }}

true
```

Listing 2: SPARQL Boolean query rewriting.

Let us now evaluate the general case.

Consider the TGD $\sigma = A(x,z) \wedge A(z,y) \rightarrow A(x,y)$ and observe that σ is not FO-rewritable since it captures the transitive closure of the relation A, which cannot be done using a finite number of first-order queries. Let us now consider an instance of a RPS $\mathcal{P}$ defined only by the following mapping assertion:

$$\forall x \forall y \exists z \; t_t(x,A,z) \wedge t_t(z,A,y) \wedge r_t(x) \wedge r_t(y) \rightarrow t_t(x,A,y),$$

We assume without loss of generality that the sources in D do not contain blank nodes. Now we transform the mapping assertion into the following set Σ of TGDs:

$$\begin{gathered} \forall x \forall y \; t_t(x,A,y) \rightarrow A(x,y) \\ \forall x \forall y \; A(x,z) \wedge A(z,y) \rightarrow A(x,y) \\ \forall x \forall y \; A(x,y) \rightarrow t_t(x,A,y). \end{gathered}$$

Note that we can drop the atoms $r_t(x)$, $r_t(y)$ in the body of the TGDs because for any D we have that $D \models \forall x \; r_t(x)$, and the same condition holds for every partial instance of the chase in each chase step.

The auxiliary predicates, being introduced only during the above construction, do not match any predicate symbol in any query q, and hence $chase(D,\Sigma) \models q$ if and only if $chase(D,\mathcal{P}) \models q$; therefore query answering under the RPS $\mathcal{P}$ is equivalent to query answering under Σ. Note that Σ now contains TGDs computing the transitive closure, so if we assume that the set of TGDs Σ are FO-rewritable, then the transitive closure is also FO-rewritable which is a contradiction.

This leads to the following result.

PROPOSITION. 3. *The sets of TGDs corresponding to the mapping assertions of RPSs are not FO-rewritable.*

5. DISCUSSION

In this paper we have addressed the problem of integrating RDF data sources in a peer-based fashion, where mappings are defined between arbitrary peers, without a centralised schema. We have proposed a formalisation of the notion of a peer-to-peer semantic integration system, where RDF triples are represented as relational tuples and mappings are expressed as tuple-generating dependencies. Following that, we have shown that answering graph pattern queries on an RDF peer system can be done in polynomial time in terms of data complexity. Finally, we have shown that it is not possible to process queries in general RDF peer systems by rewriting them into first-order queries.

This is a preliminary report which poses several new challenges. As future work, we plan the following.

1. We want to improve the efficiency of the query processing algorithm. Our query answering algorithm is naïve as it generates the whole universal solution under the given dependencies; this is far from ideal, as mappings may be subject to change and we might need to compute the information inferred from the TGDs dynamically. We intend to investigate the possibility of adopting a *combined* approach, where only part of the universal solution is computed, and queries are rewritten according to some of the dependencies only. Another possible approach is to devise a rewriting algorithm that produces rewritten queries in a language more expressive than FO-queries, for instance Datalog.

2. We intend to investigate the query answering problem for more expressive query languages, in particular for larger subsets of SPARQL.

3. We want to be able to discover mappings between peers automatically. We are investigating relevant areas such as probabilistic logics and state-of-the-art techniques for automatic schema/ontology-alignment and for managing uncertain semantic mappings.

4. Finally, we are building a prototype system to validate our techniques on real and synthetic data sets and determine their scalability properties. Our prototype is a SPARQL query engine which provides unified access to the mapped sources. The user poses a query expressed in any vocabulary known by the system, and the query is processed as follows:

   (a) A query rewriting module rewrites the original SPARQL query in order to retrieve all the certain answers.

   (b) A query module performs federated querying over the sources. It stores SPARQL access points of the RDF sources, up-to-date RDF data dumps and other information in order to query federated sources in a transparent way for the user. After query rewriting, sub-queries are posed to the relevant RDF sources and sub-query results are joined, taking into account efficiency of the join operations between the RDF triple patterns. The final result is returned to the user.

Acknowledgments.

Andrea Calì acknowledges support by the EPSRC project "Logic-based Integration and Querying of Unindexed Data" (EP/E010865/1).

6. REFERENCES

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS*, pages 254–263, 1998.

[2] S. Auer, J. Lehmann, and A.-C. N. Ngomo. Introduction to Linked Data and its lifecycle on the web. In *Proc. of RW*, pages 1–75, 2011.

[3] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[4] C. Buil-Aranda, M. Arenas, and O. Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. In *Proc. of ESWC*, pages 1–15, 2011.

[5] M. Cai and M. Frank. RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network. In *Proc. of WWW*, pages 650–657, 2004.

[6] M. Cai, M. R. Frank, B. Yan, and R. M. MacGregor. A subscribable peer-to-peer RDF repository for distributed metadata management. *J. Web Sem.*, 2(2):109–130, 2004.

[7] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of KR*, pages 70–80, 2008.

[8] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.

[9] A. Calì, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in Datalog+/-. In *Proc. of RR*, pages 1–17, 2010.

[10] G. Correndo, M. Salvadores, I. Millard, H. Glaser, and N. Shadbolt. SPARQL query rewriting for implementing data integration over linked data. In *Proc. of EDBT/ICDT*, 2010.

[11] M. Duerst and M. Suignard. RFC 3987: Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard), January 2005.

[12] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *TCS*, 336(1):89–124, 2005.

[13] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization (extended version). *CoRR*, abs/1112.0343, 2011.

[14] C. Gutierrez, C. Hurtado, and A. O. Mendelzon. Foundations of semantic web databases. In *Proc. of PODS*, pages 95–106, 2004.

[15] H. Halpin, P. Hayes, J. McCusker, D. McGuinness, and H. Thompson. When owl:sameas isn't the same: An analysis of identity in linked data. In *Proc. of ISWC*, volume 6496, pages 305–320, 2010.

[16] P. Hayes and B. McBride. RDF semantics. W3C recommendation, Feb. 2004.

[17] G. Kokkinidis and V. Christophides. Semantic query routing and processing in P2P database systems: The ICS-FORTH SQPeer middleware. In *Proc. of EDBT*, pages 486–495, 2004.

[18] W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang. Rewriting queries on SPARQL views. In *Proc. of WWW*, pages 655–664, 2011.

[19] K. Makris, N. Bikakis, N. Gioldasis, and S. Christodoulakis. SPARQL-RW: transparent query access over mapped RDF data sources. In *Proc. of EDBT*, pages 610–613, 2012.

[20] K. Makris, N. Gioldasis, N. Bikakis, and S. Christodoulakis. Ontology mapping and SPARQL rewriting for querying federated RDF data sources. In *Proc. of OTM*, pages 1108–1117, 2010.

[21] W. Nejdl. Design issues and challenges for RDF- and schema-based peer-to-peer systems. In *Proc. of DBISP2P*, page 1, 2003.

[22] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. S. A. Naeve, M. Nilsson, M. Palmer, and T. Risch. Edutella: A P2P networking infrastructure based on RDF. In *Proc. of WWW*, pages 604–615, 2002.

[23] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing strategies for RDF-based peer-to-peer networks. *WWW*, 1(2):177–186, 2003.

[24] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *TODS*, 34(3):16:1–16:45, 2009.

APPENDIX

Algorithm 1: Using the chase to compute the certain answers $ans(q, \mathcal{P}, D)$.

Data: Graph pattern query q, system $\mathcal{P}$, stored instance D.
Result: The set $\mathbf{t}$ of the certain answers $ans(q, \mathcal{P}, D)$.
Initialize instance $J = \emptyset$;
`/* Chase procedure to generate a universal solution */`
while *some of the mappings of* $\mathcal{P}$ *are not satisfied in* J **do**
  case *(*$d \not\subseteq J$ *for some* $d \in D$*):*
    add d to J;
  case *(for some graph mapping assertion in* $\mathcal{P}$*, we have* $Q^J \not\subseteq Q'^J$*):*
    for *each tuple* $t \in Q^J \setminus Q'^J$ **do**
      generate the boolean query bQ' by substituting t in the free variables Q';
      add triples to J generating new blank nodes, such that $bQ'^J = true$;
    case *(for some equivalence mapping to* $\mathcal{P}$*, we have* $c \not\equiv_e c'$*)*
      switch $subjQ(c)^{*J} \not\subseteq subjQ(c')^{*J}$ **do**
        for *each tuple* $(p, o) \in (subjQ(c)^{*J} \setminus subjQ(c')^{*J})$ **do**
          add the triple (c', p, o) to J;
      switch $subjQ(c')^{*J} \not\subseteq subjQ(c)^{*J}$ **do**
        for *each tuple* $(p, o) \in (subjQ(c')^{*J} \setminus subjQ(c)^{*J})$ **do**
          add the triple (c, p, o) to J;
      switch $predQ(c)^{*J} \not\subseteq predQ(c')^{*J}$ **do**
        for *each tuple* $(s, o) \in (predQ(c)^{*J} \setminus predQ(c')^{*J})$ **do**
          add the triple (s, c', o) to J;
      switch $predQ(c')^{*J} \not\subseteq predQ(c)^{*J}$ **do**
        for *each tuple* $(s, o) \in (predQ(c')^{*J} \setminus predQ(c)^{*J})$ **do**
          add the triple (s, c, o) to J;
      switch $objQ(c)^{*J} \not\subseteq objQ(c')^{*J}$ **do**
        for *each tuple* $(s, p) \in (objQ(c)^{*J} \setminus objQ(c')^{*J})$ **do**
          add the triple (s, p, c') to J;
      switch $objQ(c')^{*J} \not\subseteq objQ(c)^{*J}$ **do**
        for *each tuple* $(s, p) \in (objQ(c')^{*J} \setminus objQ(c)^{*J})$ **do**
          add the triple (s, p, c) to J;
`/* End of chase */`
compute the certain answers $\mathbf{t} := q^J$;
`/* The certain answers are generated by evaluating the query over the universal solution */`
return $\mathbf{t}$;

Interpreting linked data search results using Markov Logic

Duhai Alshukaili
School of Computer Science
University of Manchester
Oxford Road, Manchester
M13 9PL, UK
duhai.alshukaili@gmail.com

Alvaro A.A. Fernandes
School of Computer Science
University of Manchester
Oxford Road, Manchester
M13 9PL, UK
alvaro@cs.man.ac.uk

Norman W. Paton
School of Computer Science
University of Manchester
Oxford Road, Manchester
M13 9PL, UK
norm@cs.man.ac.uk

ABSTRACT

Linked Data (LD) follows the web in providing low barriers to publication, and in deploying web-scale keyword search as a central way of identifying relevant data. As in the web, searches initially identify results in broadly the form in which they were published, and the published form may be provided to the user as the result of a search. This will be satisfactory in some cases, but the diversity of publishers means that the results of the search may be obtained from many different sources, and described in many different ways. As such, there seems to be an opportunity to add value to search results by providing users with an integrated representation that brings together features from different sources. This involves an on-the-fly and automated data integration process being applied to search results, which raises the question as to what technologies might be most suitable for supporting the integration of LD search results. In this paper, we investigate the use of Markov Logic, which brings together first order logic and probabilistic graphical models to support both learning and inference in uncertain domains. Specifically, we: (i) characterise key features of LD search results that are relevant to their integration; (ii) discuss how these motivate the use of an approach based on Markov Logic; (iii) describe some initial experiences in the use of Markov Logic for interpreting search results; and (iv) present some avenues for future investigation.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed databases; H.3.5 [**Online Information Services**]: Data sharing; H.3.5 [**Online Information Services**]: Web-based services

1. INTRODUCTION

Linked Data (LD) seeks to do for data what the web did for documents. In essence, LD involves publication of data according to a small collection of principles that indicate how data resources are identified and represented, and that encourage the creation of links [1]. Publishers make data available following the principles, and users access or process the resulting data using either generic tools or bespoke applications.

As in the web of documents, keyword search engines are an important element in the tool set. However, although search results in the web of documents provide a result that was designed for human use, search results in the web of data tend to be collections of RDF resources that may be time consuming and cumbersome to explore. Furthermore, the results of a search may involve values of different types, which are interleaved in a search result. For example, a search for *Bob Dylan* using the *Sindice* search engine [22] returns results that represent a person, (several) albums and (several) songs. A manually created report over such a search result might pull together the properties of the individual *Bob Dylan* from several resources into a heading and a list of properties, and then might provide separate tables for the collections of albums and songs about which information was retrieved.

Could such a report be generated automatically? The creation of such a report requires, at a minimum, identifying:

- the (real-world) entity types that are represented in the search result;
- the individuals entity instances that are represented in the search result, and in particular which ones are individuals and which ones belong to collections; and
- the properties of each of the relevant entity instances.

Although this is a data integration problem, it is quite unlike classical data integration, in which typically there is a known target (or global) schema to which source data should be mapped, and there is some level of human engagement in the data integration. This raises the question as to what approaches might be suitable for interpreting and integrating search results.

We know of one previous proposal to address this problem, namely Sig.ma [24], which generated a report that integrated the top hits from the Sindice search engine. In Sig.ma, several steps were followed to integrate the data, as discussed more fully in Section 2, but the overall approach assumed that the result of the search described a single entity instance. Although Sig.ma was important in identifying the

opportunity and in developing an initial realisation of the vision, the data integration component of Sig.ma was quite restrictive and seemed rather *ad hoc*. As such, there seems to be scope to explore additional approaches that seek to make the result integration both more general and more systematic. We would expect there to be several very different ways of addressing this problem, one of which is explored in this paper.

Here we discuss an approach based on Markov Logic [6], in which we: (i) postulate rules that capture relationships that exist within search results, which are expressed using logic; (ii) learn weights for these rules that represent their strengths as constraints; and (iii) use the resulting weighted rules over search results to infer (with uncertainty) the entity types and instances that are represented in the result. We are motivated to use Markov Logic as it provides a well founded approach to integrating evidence of different types to support conclusions that can inform data integration. Markov logic has been applied to a range of tasks of relevance to data integration, including classification, entity resolution and knowledge-base construction, as discussed more fully in Section 2.

This paper investigates the application of Markov Logic to the integration of LD search results. In so doing, it uses the following structure. In Section 2, we review related work in linked data search, linked data integration and applications of Markov Logic. In Section 3 we provide an overview of Markov Logic, and in Section 4 we describe how it can be applied for identifying entity types and instances in linked data search results. In Section 5 we draw some conclusions and outline directions for future work.

2. RELATED WORK

In this section we discuss work that is relevant to the problem of linked data search result integration, specifically by reviewing results on *Linked Data Search*, *Data Integration for Linked Data* and *Markov Logic for Data Integration*.

Linked Data Search. The increase in the amount of RDF data published in the web has given rise to a number of LD search proposals. Swoogle [5] is an early LD search engine that indexed RDF(s) and OWL ontologies using inverted indexes. Given a set of terms, Swoogle returns ontologies that mention these terms. As such, Swoogle adopts a document-centric approach for indexing LD ontologies [12]. Unlike Swoogle, the Falcons [3] search engine was tailored towards the search of arbitrary data while providing an entity-centric search approach in which the objective is to identify the most relevant RDF resources rather than the most relevant documents that contain them. Falcons included components for crawling, parsing, organizing, ranking, sorting and querying RDF data. In order to extract terms from RDF documents, it employed the notion of a virtual document as an intermediate representation that enables the consolidation of data from multiple sources. This consolidation is based on URI reuse and the mention of an entity in different sources. Falcons also includes a reasoning component aimed at inferring class hierarchies of indexed entities. These hierarchies provide a means for restricting the search result based on the types of resources. Sindice [22] adopts a document-centric approach and aimed to provide a range of search services over RDF documents. The services offered include keyword search over RDF, searching for entities (classes, properties and individuals) matching a term in RDF documents, and providing APIs to expose search services to software agents. Sindice consolidates entities while indexing, based on inverse functional properties. It also implements a localized reasoning component for discovering additional information about entities. From this brief review, it can be seen that LD search engines often carry out some preliminary result consolidation tasks, but falls short of a concerted approach to result integration.

Sig.ma [24], however, does address result integration in a more comprehensive manner. Sig.ma uses search results from Sindice [22] to collect RDF data and build an aggregated view of the results in the form of entity profiles. Sig.ma uses a recursive search step that collects RDF data which contains resource identifiers that match a search term. A first search step collects the source URLs that contain the search terms. A second step is performed to search for sources containing the URI identifiers found in the results of the first step. The collected RDF data is decomposed into chunks (called resource descriptions) that describe distinct entities, and ranked against the search term. Sig.ma collects additional data when it encounters an *owl:sameAs* predicate. The resulting resource descriptions are consolidated by combining the values of lexically similar attributes. Hand-crafted rules such as the removal of "has" from "has title" or replacement of attributes that may share similar values such as "web page" and "homepage" with the term "Web page" are applied in the consolidation step. In addition, Sig.ma allows users to interact with the resulting entity profiles, either through navigating to other sources of information, or by refining the results. The refinement capabilities allow users to reject or accept the sources that contribute data to the generated entity profiles. However, Sig.ma does not provide any means for resolving semantic heterogeneities in the data before attempting to construct the integrated view. For example, if a user is interested in information about *Manchester University*, Sig.ma combines data on the university, on Manchester Grammar School, and on a railway station. While it brings together lots of correct information, some incorrect data is often included. Our aim is to develop a more principled approach that enables us to resolve such heterogeneities through the identification entities and their types using Markov Logic.

Data Integration for Linked Data. Low barriers to publication, as well as diversity of publishers without central coordination, have led to LD being published with inconsistencies both at the conceptual and at the instance levels. At the conceptual level, this comes in the form of different conceptualizations of the same domain, inconsistencies in the structural representation of concepts in LD terminologies, etc. At the instance level, there may be many different resources that describe the same real world entities, redundant information, and contradictory attribute values. There is a plethora of approaches that deal with the alignment and matching of RDF sources. These approaches can be divided into two broad categories: ontology matching and instance matching approaches [2]. The goal of ontology matching ap-

proaches is to align schema level elements of RDF sources using information from the schema level, the instance level, or both [7]. On the other hand, instance level approaches try to resolve the multiplicity of data resources that describe the same real world entity [8]. RDF instance matching tools such Silk [16, 15], ObjectCoref [14], and LIMES [18] discover *owl:sameAs* identity links at the instance level. Our work aims not just to resolve identities between pairs of resources in the search results but also to infer a ER schema structure for the results and populate with data provided in the instances of the search. In this regard, there have been a number of proposals for structure inference from RDF sources [4, 27, 28, 25]. Such approaches take as an input a data graph and produce a structural summary that is homomorphic to the original data graph using techniques such as hierarchical clustering [4, 28], association rule mining [25], and inference using Bayesian Networks [27]. However, these approaches are often evaluated on a specific dataset at time (i.e. Magnatune or DBpedia). This is different from inferring a structure from search results because the relevant sources in the results often vary in terms of the datasets they originate from. For example, a search for a film title (e.g. Godfather) on Sindice [22] returns results from at least three datasets: DBpedia[1], freebase[2], and LinkedMDB [3]. This makes the structure inference problem harder as mappings between the dataset need to be inferred or incorporated as evidence.

In addition to research in linking RDF at the conceptual and instance levels, there have been some studies on mapping properties across RDF sources [9, 26, 10]. The aim of such approaches is to find similar [9] or equivalent [26, 10] properties using statistical measures that utilize subject and object overlap of properties. These approaches, as well as approaches that map between ontology concepts and instances across datasets can be utilized as additional sources of evidence in in our approach (see Section 5).

Markov Logic for Data Integration. Markov Logic brings together two prominent paradigms within artificial intelligence, namely first-order logic and Markov networks [6]. The basic idea is that the syntax of first order logic is used to describe constraints that hold on the set of possible worlds, but that the constraints are no longer necessarily hard. Instead, each formula is associated with a weight that indicates how strong the constraint is; the higher the weight the more likely a constraint is to hold. A set of weighted formulas is referred to as a Markov logic network (MLN).

Markov Logic (ML) has been applied to data integration tasks such as entity resolution [23], knowledge base construction [20] and ontology matching [19]. Sigla and Domingos [23] described a domain-specific MLN that performs an entity resolution task on bibliography entries. They used an MLN to encode knowledge about the similarity of the publications based on the similarity of the venue, authors and titles. They demonstrated that such an MLN, combined with predicate equivalence and reverse predicate equivalence axioms, can achieve superior results to established approaches.

[1]http://www.dbpedia.org
[2]http://rdf.freebase.com
[3]http://data.linkedmdb.org/

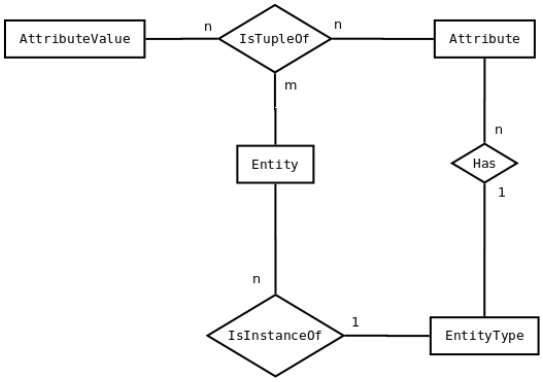


Figure 1: The ER metamodel that is to be populated from search results.

In contrast with this approach, the rules we use in our MLN do not currently encode domain-specific knowledge. In seeking to integrated data from multiple sources, our work is similar to Elementary [20]. The goal of Elementary is the construction of a knowledge base of entities based on information extracted from web pages. It uses an MLN inference engine for discovering co-referent entity mentions of people and organizations. It also links between such entities by using the co-occurrence pattens as evidence in the inference process. Elementary combine various forms of evidence, such as data extracted from standard NLP toolkits, domain knowledge, lexical matching, and user feedback in the inference process. In the context of LD, Niepert *et al.* [19] utilized ML for ontology and RDF instance matching problems. Their approach to ontology matching was combining logical axioms expressed in ontologies with lexical similarities to map between the concepts and attributes in different ontologies. In their instance matching problem they utilized a similarity metric [21] to infer matches between instances. They used the constraints defined in the ontologies of these instances to prevent the MLN from making incorrect inferences. While this approach demonstrates how semantic and syntactic evidence can be combined using ML, this approach assumes that ontology definitions are available with the instance data, which is not necessarily the case for LD search results.

3. MARKOV LOGIC

Markov Logic combines *first-order logic* and Markov networks in a unifying representation for the definition of probabilistic models.

Formally, a Markov logic network (MLN) is a set of pairs (F_i, w_i), where F_i is a first-order logic formula and w_i is a real value representing its weight. Formulas can be seen as constraints on a set of possible worlds. The higher the weight, the stronger the constraint is, and therefore, the less probable is a world that violates the constraint. In an MLN, a formula with a negative weight w can be replaced with its negated formula with a weight of $-w$. A formula can also be assigned an infinite weight to indicate a constraint that should not be violated. Given as set of constants in some domain, an MLN defines a ground Markov network where the nodes correspond to ground predicates.

//Evidence Predciates		
Triple1(uri, uri, uri)	Triple1(s,rdf:type,o) => Entity(s) ^ EntityType(o) ^ IsInstanceOf(s,o)	R_1
Triple2(uri, uri, literal)	Triple2(s,p,o) ^ Attribute(p) ^ AttributeValue(o) => Entity(s)	R_2
	Triple1(s,p,o) ^ Attribute(p) ^ LnkAttributeValue(o) => Entity(s)	R_3
//Query Predicates	Triple2(s,p,o) ^ Entity(s) ^ AttributeValue(o) => Attribute(p)	R_4
Entity(uri)	Triple1(s,p,o) ^ Entity(s) ^ LnkAttributeValue(o) => Attribute(p)	R_5
EntityType(uri)	Triple1(s,p,o) ^ Entity(s) ^ Attribute(p) => LnkAttributeValue(o)	R_6
Attribute(uri)	Triple2(s,p,o) ^ Entity(s) ^ Attribute(p) => AttributeValue(o)	R_7
AttributeValue(literal)	Triple1(s,rdf:type,type) ^ Triple2(s,p,o)	R_8
LnkAttributeValue(uri)	^ EntityType(type) ^ Attribute(p) => Has(type,p)	
Has(uri,uri)	Triple1(s,rdf:type,type) ^ Triple1(s,p,o)	R_9
IsInstanceOf(uri,uri)	^ EntityType(type) ^ Attribute(p) => Has(type,p)	

(a) Predicates (b) Rules

Figure 2: An MLN rule set that uses RDF triples to derive ER construct extensions

An MLN formula is defined over a set of predicates. The predicates can be categorized into *query* and *evidence* predicates. The MLN formulas define relationships using these predicates. The MLN in Figure 2 describes relationships between *evidence* predicates that represent search results and *query* predicates that represent the constructs of the entity relationship (ER) meta model in Figure 1. The evidence and query predicates are then related to each other by formulas such as:

$$Triple1(s, "rdf : type", o) \implies EntityType(s).$$

In this formula, s and o are variables that represent the subject and object of a triple in a search result. The formula states that where there is a triple in which s is related to o by $rdf : type$, we can infer that s is an $EntityType$. Now, in fact, this may not always be the case, and the weight associated with the formula in the MLN captures the strength of the constraint represented by the rule. We discuss the evidence and query predicates, as well as the formulas in our MLN model, in more detail in Section 4.

Given a domain of interest, there are three tasks to be performed by the modeller: structure learning, weight learning and inference. An overview of these tasks now follows.

Structure Learning

Given a set of predicates and example data for the domain of interest, the ML structure learning process learns first-order logic formulas that define the relationships between the given declared predicates from the evidence provided in the form of example data in the domain. The structure learning process uses a beam search strategy to find the best clauses to add to the MLN [6]. In theory, structure learning provides an alternative to relying on domain experts to write rules that capture the semantics of the domain. However, the structure learning process is known to give rise to scalability issues for large datasets [17]. Given that, in this case, the presumed MLN structure is known *a priori*, i.e., we have developed a meta-model of ER models, and written rules for populating this meta-model based on the evidence in the form of triples from the search result. Thus, we have not performed structure learning and hence do not report any results in this respect.

Weight Learning

Given a set of rules and a database of evidence from the domain of interest, the weights of the rules can be learned. In this process, one or more predicates whose truth values are unknown are designated as query predicates. The learning procedure optimizes the learned weights with respect to such predicates assuming that all the truth values of the remaining predicates are given. The weight produced for each rule can be either positive, negative, or zero. A positive weight is an indication that a rule is supported in the domain given the evidence. On the other hand, a learned negative weight $-w$ is an indication that a rule is not supported by domain evidence, and in fact its negation is supported in the domain with weight w. Finally, a weight 0 suggests that a rule has no evidence (either for or against) in the domain. This occurs when groundings of the rule are not provided in the evidence.

Inference

The inference process takes as input a weighted MLN and a database consisting of ground evidence predicates, and outputs the marginal probabilities of query predicates of the most likely world given the evidence. This involves finding the truth assignment that maximises the sum of the weights of satisfied clauses [6].

4. INTERPRETING LINKED DATA SEARCH RESULTS

The problem we address is that of inferring, from search results, the entities, entity types, attributes and relationships that are described in the results. By these terms, we mean the constructs that are familiar from entity-relationship (ER) conceptual modelling. Identifying what entities and attributes are described from the user's point of view is a task with uncertain outcomes. This is because not every resource described using RDF is seen as an entity by the user. For example, the RDF resource depicted in Figure 3(a) that describes an organization named "Universities UK" may not be considered an entity in a search for members of a collection of UK universities. Also, not all RDF predicates that are used in the description of a resource can be seen as attributes of interest to the end user. An example of this is *dbo:wikiPageID*, which denotes a Wikipedia page identifier from the DBpedia dataset.

To model such uncertainty, we use ML to define a number of hypotheses as to what the URIs and literals in the resources

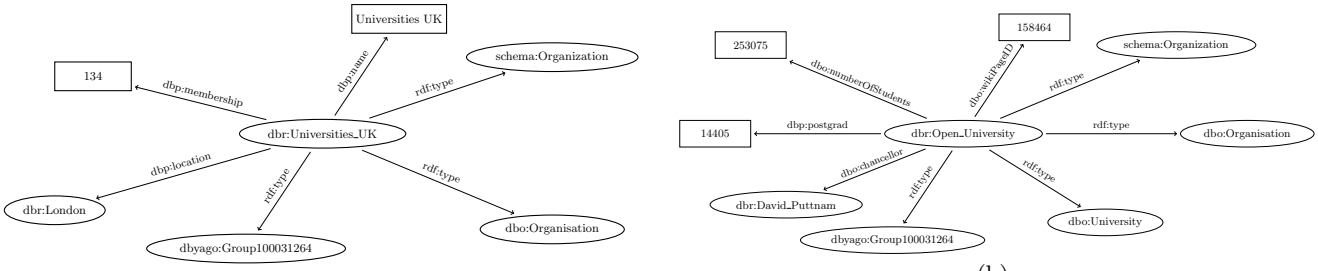


Figure 3: An Example of search result for a search of "Universities UK"

denote in terms of ER constructs. Inference from such hypotheses allows us to build an ER view of the data in the search results, and thereby to organize the results in a form that is suitable for the user. A key advantage of using ML for this task is that it provides an opportunity for incorporating different forms of evidence, including using feedback as typical of the *pay-as-you-go* approach to data integration [11]. We now describe an MLN for learning the uncertainty of constructing such ER views from the search results. We also describe the learning and inference processes. Finally, we present the initial experimental results of our investigation.

4.1 An MLN for search results

We now describe the MLN in Figure 2 that we use to interpret RDF search results. The evidence predicates represent observed variables with known truth values in the ground Markov network. The query predicates represent the unobserved variables for which the inference process estimates a probability distribution given the evidence. In this MLN, the evidence predicates represent the observed triple patterns in the RDF data. These predicates are defined over the *uri* and *literal* domains. Partitioning the space of triples provides a simpler, more direct way for writing rule bodies. As such, we use *Triple1* to represent RDF triples that have URIs in the object position, and *Triple2* to represent RDF triples what have literals in the object position. Then, for example, we can use the *Triple2* predicate in Rule R_7 in Figure 2 for the query predicate *AttributeValue* because we expect a literal in the object position. Such partitioning can be extended to include BNodes, although, here we only use *uri* and *literal* domains.

There are seven query predicates that characterize the hypotheses we want to substantiate using the evidence, which represent the constructs in the meta-model in Figure 1. The *Entity*, *EntityType*, and *Attribute* predicates model whether a given URI represents an entity, entity type, or attribute. For example, given the RDF graph shown in Figure 3(b), the following are true:

Entity(dbr:Open_University),
Attribute(foaf:name), and
EntityType(schema:Organization).

Conversely, given the graph shown in Figure 3(a),the following are false:

Entity(dbp:name),
Attribute(dbr:London), and
EntityType(dbr:Universities_UK).

We also define predicates for *Has* and *IsInstanceOf*. *Has* models a relationship between two URIs where the first is an entity type and the second is an attribute of that type. *IsInstanceOf* models a relationship between two URIs where the first is an entity, the second is an entity type, and the first is an instance of the second. Examples from Figure 3 are:

Has(dbo:Organisation,dbp:location), and
IsInstanceOf(dbr:Open_University,dbo:Organisation).

Finally, we use two predicates to interpret values in the object position of RDF triples, viz., *AttributeValue* and *LnkAttributeValue*.

In addition to the described predicates, the MLN contains rules that encode knowledge about the relationships between MLN predicates. Rules R_1, R_8 and R_9 utilize the *rdf:type* construct for the inference of *Entity*, *EntityType*, *IsInstanceOf* and *Has*. Note that R_1 makes a direct inference from the evidence, whereas R_8 and R_9 additionally rely on *EntityType* and *Attribute*.

4.2 Experiments

Dataset

To our knowledge, there is no publicly available standard dataset that allows us to evaluate our proposed approach. In order to learn the weights for the MLN described in Section 4.1 we conducted 10 searches using the Sindice [22] LD search engine. The terms used in these searches are shown in Table 1. The results were pre-processed by removing triples containing *rdf:type* objects that belong to the *yago*[4] and *dbyago*[5] name-spaces. The reason is that such types are used for categorizing resources as opposed to assigning real-world entity types to resources. These types cannot be easily assigned specific attributes. Also triples which contain domain-independent RDF predicates and have literal objects were removed: *dct:abstract*, *rdfs:comment*, *rdfs:label*, *skos:prefLabel*, *skos:altLabel*, *skos:note* and *dce:description*.

[4]http://yago-knowledge.org/resource/
[5]http://dbpedia.org/class/yago/

Predicate	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Entity	0.41	**0.51**	0.41	**0.61**	**0.90**	**0.568** ± 0.203
EntityType	**0.95**	**0.97**	**0.88**	**0.92**	**0.97**	**0.938** ± 0.038
Attribute	0.25	0.15	0.24	0.14	0.17	0.190 ± 0.051
Has	0.06	0.08	0.04	0.04	0.07	0.058 ± 0.018
InstanceOf	**0.85**	**0.89**	**0.90**	**0.90**	**0.77**	**0.862** ± 0.055

Table 3: AUC PR scores per fold

Domain	Search Terms
Cities	Berlin,Manchester
Movies	Godfather,Casablanca
Organizations	Apple Inc.,Microsoft
People	Tim Berners-Lee, Chris Bizer
Collections	Godfather actors, UK universities

Table 1: 10 search terms used in constructing the learning/evaluation dataset

```
Entity(dbr:Open_University)
Entity(dbr:Open_University)
EntityType(dbo:Organisation)
EntityType(dbo:University)
Has(dbo:University,dbp:postgrad)
Has(dbo:University,dbo:chancellor)
IsInstanceOf(dbr:Open_University,dbo:University)
IsInstanceOf(dbr:Open_University,dbo:Organisation)
```

Figure 4: Ground truth annotation for the RDF in Figure 3(b)

From each search, the top five results were selected to be annotated with the ground truth. Figure 4 shows an example of the ground truth annotation for the RDF graph shown in Figure 3(b).

Methodology and Results

To learn the weights of the rules in the MLN, we used a 5-fold cross-validation procedure on the annotated dataset. To ensure that the weights are not skewed towards a particular domain of search, we randomized the search results to ensure that every split contained search results from every one of the domains shown in Table 1. Table 2 shows the average weights learned for the MLN rules. A positive weight of a rule is an indication that the rule is supported by the evidence present in the data. The higher the weight, the stronger the evidence for the corresponding rule. One obvious observation is that the weight of R_1 is much higher than the weight of other rules. The reason for this is that *rdf:type* provides the strongest signal that URI represents an entity as opposed to the strength of the signal for the other query predicates in the MLN.

As mentioned in Section 3, the ML inference engine returns the probability that a query atom is true. To evaluate the inference results we measured the area under precision/recall (AUC PR) curves for all query predicates in the MLN model. The precision/recall curve is computed by varying the threshold above which a query atom is predicted to be true. Table 3 shows the AUC PR scores obtained for every query predicate for different test folds. We note that while the standard deviation for *Entity* is rather high, for all other query predicates it is relatively small.

Rule ID	Average Weight
1	55.259 ± 2.255
2	1.433 ± 0.151
3	1.793 ± 0.188
4	4.051 ± 0.121
5	4.161 ± 0.111
6	2.903 ± 0.248
7	3.227 ± 0.086
8	1.140 ± 0.358
9	1.229 ± 0.249

Table 2: Average weights learned for each rule

The MLN seems to perform well on *Entity*, *EntityType*, and *IsInstanceOf*; it is able to extract the signal from the data that allows the inference of these predicates. On the other hand, the MLN does not perform well on *Attribute* and *Has*. This could be attributed to inference chains in the body of the rules that define these predicates. In ML, longer chains mean that the inference engine has a larger space to sample from, which reduces the likelihood of finding the correct answer for the query predicate. Note that *Attribute* and *Has* atoms correspond to schema elements with weak signal from the data (e.g., no one-step inference from *rdf:type* as with rule R1). Previous research has shown that schema inference from instance data is challenging [4]. In LD search, this problem is even harder because of the variability in the data resources in terms of the descriptions they use. In Section 5 we discuss proposals for improving the performance of the MLN.

5. CONCLUSIONS AND FUTURE WORK

Search results over the web of data consist of collections of triples from a range of sources, and typically contain RDF resources that describe real-world entities of different types. In addition, search results often contain assertions that provide additional metadata about the entities, which means that the result of a search is a complex data set that may be difficult to interpret automatically.

With a view to managing this complexity, we have investigated the use of Markov Logic to infer, with uncertainty, which triples in a search result represent types, individuals, attributes and attribute values. This we have done by learning the weights of an MLN, where the associated rules express various hypotheses about the relationships between the triples in a search result and the roles the elements in those triples may be able to play in an entity relationship diagram. The reason for targeting an entity relationship

diagram is that we would expect to be able to generate intuitive tabular reports capturing features of search results from such a representation.

The initial results might be considered to be somewhat disappointing. Although we have been able to identify entity types and instance-of relationships from the search domain with high confidence, entity instances and attributes are not being identified reliably by our MLN. Although there are different possible reasons for this (e.g. that the rules for identifying such features could be improved upon), our preferred interpretation is that the rules are plausible, it is simply that the evidence to support them in actual search results is relatively weak. In this context, the evidence may be lacking because, for example, different publishers publish the data in different ways, or a significant fraction of the data retrieved is not directly concerned with the structure of the data in the domain. This in turn suggests that the problem of capturing the domain knowledge in a search result, from the contents of that result, is a difficult one.

Given these challenges, what might be the way ahead? The Markov Logic framework is quite a general one, and we selected it for use with this problem in part because this seems to be an evidence-rich problem, in which the results of the search can be combined with additional information to enable well founded inferences to be drawn. We envisage that the following avenues can be pursued:

- *Additional generic integration rules.* To date, the integration rules have focused on the identification of concepts from entity relationship diagrams using the data from the search result. However, it would be possible to write additional generic rules. For example, none of the current rules make use of the search terms, and no attempt is made to identify duplicate triples or entities across resources retrieved by the search. Additional rules that capture such relationships may be useful in distinguishing between the domain knowledge in a result and associated metadata.
- *Domain-specific integration rules.* Successful applications of Markov Logic, for example in entity resolution or knowledge base construction, have often made use of domain-specific rules. As such, although searches are generic, it would be possible to write rules that know about certain domains, and the widely used terminologies in such domains. For example, rules could be written that are informed by common searches in search logs, or that cover terminologies that are widely used in practice [13]. Another possibility here is writing rules that use mappings between instance level or schema level elements produced by exiting tools.
- *Integration of results of other analyses.* The current rules act directly on the search results. However, it would be possible to run additional analyses on these search results, which in turn could be reflected in rules. For example, analyses could carry out ontology alignment between search results, or could cluster triples based on attribute values. The results of such analyses could then be used as evidence predicates, and included in additional generic or domain-specific integration rules.
- *Integration of feedback.* In Sig.ma, users are able to refine the reports produced by ruling in/out specific sources of data. However, in pay-as-you-go data integration, feedback of different forms can be provided, for example on the correctness or relevance of specific results. Such feedback could be used as evidence by an MLN to inform the inference of results for different query predicates.

6. REFERENCES

[1] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[2] Silvana Castano, Alfio Ferrara, Stefano Montanelli, and Gaia Varese. Ontology and instance matching. In Georgios Paliouras, ConstantineD. Spyropoulos, and George Tsatsaronis, editors, *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution*, volume 6050 of *Lecture Notes in Computer Science*, pages 167–195. Springer Berlin Heidelberg, 2011.

[3] Gong Cheng and Yuzhong Qu. Searching linked objects with falcons: Approach, implementation and evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):49–70, 2009.

[4] Klitos Christodoulou, Norman W Paton, and Alvaro AA Fernandes. Structure inference for linked data sources using clustering. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 60–67. ACM, 2013.

[5] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659. ACM, 2004.

[6] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, 2009.

[7] Jérôme Euzenat, Pavel Shvaiko, et al. *Ontology matching*, volume 18. Springer, 2007.

[8] Alfio Ferraram, Andriy Nikolov, and François Scharffe. Data linking for the semantic web. *Semantic Web: Ontology and Knowledge Base Enabled Tools, Services, and Applications*, page 169, 2013.

[9] Linyun Fu, Haofen Wang, Wei Jin, and Yong Yu. Towards better understanding and utilizing relations in dbpedia. *Web Intelligence and Agent Systems*, 10(3):291–303, 2012.

[10] Kalpa Gunaratna, Krishnaprasad Thirunarayan, Prateek Jain, Amit Sheth, and Sanjaya Wijeratne. A statistical and schema independent approach to identify equivalent properties on linked data. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 33–40. ACM, 2013.

[11] Cornelia Hedeler, Khalid Belhajjame, Norman W Paton, Alessandro Campi, Alvaro AA Fernandes, and Suzanne M Embury. Dataspaces. In *Search Computing*, pages 114–134. Springer, 2010.

[12] Aidan Hogan. *Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora*. PhD thesis, National University of Ireland,

Galway, 2011.

[13] Aidan Hogan, Jürgen Umbrich, Andreas Harth, Richard Cyganiak, Axel Polleres, and Stefan Decker. An empirical survey of linked data conformance. *J. Web Sem.*, 14:14–44, 2012.

[14] Wei Hu, Jianfeng Chen, and Yuzhong Qu. A self-training approach for resolving object coreference on the semantic web. In *Proceedings of the 20th international conference on World wide web*, pages 87–96. ACM, 2011.

[15] Robert Isele and Christian Bizer. Active learning of expressive linkage rules using genetic programming. *Web Semantics: Science, Services and Agents on the World Wide Web*, 23:2–15, 2013.

[16] Robert Isele, Anja Jentzsch, and Christian Bizer. Silk server-adding missing links while consuming linked data. In *COLD*, 2010.

[17] Hassan Khosravi and Bahareh Bina. A survey on statistical relational learning. In *Advances in Artificial Intelligence*, pages 256–268. Springer, 2010.

[18] Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes: a time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pages 2312–2317. AAAI Press, 2011.

[19] Mathias Niepert, Jan Noessner, Christian Meilicke, and Heiner Stuckenschmidt. Probabilistic-logical web data integration. In *Reasoning Web. Semantic Technologies for the Web of Data*, pages 504–533. Springer, 2011.

[20] Feng Niu, Ce Zhang, Christopher Ré, and Jude Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 8(3):42–73, 2012.

[21] Jan Noessner, Mathias Niepert, Christian Meilicke, and Heiner Stuckenschmidt. Leveraging terminological structure for object reconciliation. In *The Semantic Web: Research and Applications*, pages 334–348. Springer, 2010.

[22] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *IJMSO*, 3(1):37–52, 2008.

[23] P. Singla and P. Domingos. Entity resolution with markov logic. In *Data Mining, 2006. ICDM '06. Sixth International Conference on*, pages 572–582, Dec 2006.

[24] G. Tummarello, R. Cyganiak, M. Catasta, S. Danielczyk, R. Delbru, and S. Decker. Sig.ma: Live views on the web of data. *J. Web Semantics*, 8(4):355 – 364, 2010.

[25] Johanna Völker and Mathias Niepert. Statistical schema induction. In *The Semantic Web: Research and Applications*, pages 124–138. Springer, 2011.

[26] Ziqi Zhang, Anna Lisa Gentile, Isabelle Augenstein, Eva Blomqvist, and Fabio Ciravegna. Mining equivalent relations from linked data. In *ACL (2)*, pages 289–293, 2013.

[27] Man Zhu, Zhiqiang Gao, Jeff Z Pan, Yuting Zhao, Ying Xu, and Zhibin Quan. Ontology learning from incomplete semantic web data by belnet. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pages 761–768. IEEE, 2013.

[28] Nansu Zong, Dong-Hyuk Im, Sungkwon Yang, Hyun Namgoon, and Hong-Gee Kim. Dynamic generation of concepts hierarchies for knowledge discovering in bio-medical linked data sets. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, page 12. ACM, 2012.

TripleGeo-CSW: A Middleware for Exposing Geospatial Catalogue Services on the Semantic Web

Spiros Athanasiou[§] Nikos Georgomanolis[§] Kostas Patroumpas[†,§]
Michalis Alexakis[§] Thodoris Stratiotis[§]
[§]Institute for the Management of Information Systems, "Athena" Research Center, Hellas
[†]School of Electrical & Computer Engineering, National Technical University of Athens, Hellas
{spathan, ngeorgomanolis, kpatro, alexakis, stratiot}@imis.athena-innovation.gr

ABSTRACT

A wealth of data and services are available on the Web, and often have geographical context as well. But the vast quantity of offered geospatial information is rather difficult to explore, and its quality hard to assess, due to lack of sufficient metadata. Hence, the Open Geospatial Consortium has specified application profiles for publishing, accessing, and searching over collections of spatial metadata with standardized *Catalogue Services for the Web* (CSW). Unfortunately, existing spatial metadata remain largely unexploited by Semantic Web technologies. In this paper, we introduce TripleGeo-CSW, a middleware that can be used to discover metadata from existing CSWs through a virtual SPARQL endpoint. Acting as broker between a request (in SPARQL) and catalogue services (in XML/GML), this platform can provide on-the-fly information (as RDF triples) on available geodata according to multiple, user-specified criteria (e.g., area of interest, date of last update, keywords). As a proof of concept, we have set up an instance of this middleware against CSWs from public authorities across Europe, which involve datasets complying with the EU INSPIRE Directive. Our experience testifies that TripleGeo-CSW can assist stakeholders to repurpose existing CSWs with minimal overhead and readily expose spatial metadata on the Semantic Web.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms

Design, Management, Standardization

Keywords

Catalogue services, geospatial data, metadata, CSW, RDF, GeoSPARQL, INSPIRE.

1. INTRODUCTION

Proliferation of location-aware devices (smartphones, car navigators, etc.) over the past decade has led to an unprecented offering of geospatial information on the Web. Not only maps of the finest detail or satellite imagery of the entire planet, but also geotagged photographs and geolocation hashtags in social networking underscore the importance of geography in our everyday life and activities. Crowdsourcing has also become a valuable means of providing up-to-date geodata for free, thanks to initiatives such as OpenStreetMap [21], GeoNames [10], or Wikimapia [34] that engage thousands of volunteers worldwide.

However, all this geospatial information comes in so many different formats, heterogeneous schemas, proprietary systems, customized services, etc., such that assessing its quality becomes a burden even for experts. For example, choosing an unreliable road network for a routing application may disappoint users despite its friendly interface; having updated locations for points of interest (restaurants, cinemas, bars, etc.) in a digital city guide could be the key to commercial success; and an accurate geological map is indispensable in mineral exploration or when constructing transport infrastructures. With so many geospatial data coming from commercial vendors, governmental agencies, or crowdsourcing, the need for precise *metadata* is indisputable. Such metadata can provide a brief summary about the content, purpose, quality, location of the spatial data, and also report on its creation procedures. Indeed, information about the geographical reference (i.e., its Coordinate Reference System– CRS), resolution (i.e., the map scale used in digitization), date of last update, or textual keywords describing the content of digital maps, can greatly assist users to choose the geospatial features that best suit their needs.

ISO standard 19115:2003 [12] (recently updated to ISO 19115-1:2014 [13]) offers specifications for *standardized metadata* that can support users in effective discovery and retrieval of geodata. With the endorsement of the Open Geospatial Consortium (OGC), this standard establishes a common terminology for metadata elements on geospatial features, properties, and entire collections of geodata. Furthermore, *catalogue services* are important in publishing and searching collections of metadata for geospatial data and related web services. Metadata in catalogues represent resource characteristics that can be queried and presented for evaluation and further processing by both humans and software. OGC standard on *Catalogue Services for the Web* (CSW) [18] specifies a framework and interfaces for defining application profiles of services based on geospatial metadata.

This metadata can be queried in order to return results in well-known content models (metadata schemas) and encodings, e.g., in Geography Markup Language (GML) [19]. For example, returned metadata records may contain information about the title of datasets, their format, geographical extent (i.e., their Bounding Box in latitude and longitude coordinates), the Coordinate Reference System, licensing policies, as well as links to other associated metadata.

Unfortunately, accessing such spatial catalogue services is currently disjoint from the Semantic Web, without any means to repurpose the contents of existing catalogues according to the Linked Data paradigm [3]. Having high-quality linked metadata resources on available geodata could offer great advantages for users and applications. Catalogue contents would become machine reabable and potentially interlinked with information from third parties. Fortunately, the recent OGC GeoSPARQL standard [20] proposes structures for storing RDF geometries, querying them through a SPARQL extension [31] equipped with a variety of spatial operations [2], as well as with support for spatial reasoning on Linked Open Data (LOD). We regard this as a great opportunity to expose spatial metadata from catalogues encoded in RDF [30] and queried through GeoSPARQL.

Yet another development may also act as a catalyst for publishing linked spatial metadata. By 2020, implementation of the INSPIRE Directive (*INfrastructure for SPatial InfoRmation in Europe*) [7] will enable discovery, download, and visualization of geospatial information across the European Union in a common, cross-boundary manner. Paving the way towards geospatial data interoperability and dissemination, several public organizations across Europe have begun publishing metadata in spatial catalogues according to the ISO and OGC specifications. Availability of such official, diachronic, high-quality information can have major benefits to governance, research, and enterpreneurship. In case such metadata were made accessible via SPARQL endpoints, they would certainly offer great perspectives for extracting spatial knowledge and interlinking.

Towards these goals, we introduce TripleGeo-CSW [1], which is essentially an open-source *CSW-to-RDF middleware*. Easily coupled with a web interface so as to constitute a virtual GeoSPARQL enpoint, it allows users to explore the quantity and quality of spatial datasets available from several existing Catalogue Services according to multiple search criteria. With TripleGeo-CSW, GeoSPARQL queries are translated on-the-fly into requests against CSW on remote servers over HTTP protocols. Using RDF mappings for XML/GML encodings of standard geospatial metadata, the server response is suitably transformed via XSL stylesheets [33] into RDF triples that are finally returned as answers. To the best of our knowledge, this is the first attempt to build an abstraction layer on top of the CSW and INSPIRE infrastructures based on GeoSPARQL concepts, thus making spatial catalogues accessible and discoverable as linked metadata with geometries. Our contribution can be summarized as follows:

- We have implemented TripleGeo-CSW, a middleware that enables searching for available geodata through a virtual GeoSPARQL interface for CSW.
- We have specified application profiles that can be used as templates for transforming geospatial XML/GML metadata into RDF by an XSLT parser.
- As a proof of concept, we exposed existing INSPIRE-aligned catalogue services as linked data sources in RDF. With minimal overhead, this web interface enables queries in GeoSPARQL for discovering geospatial resources across Europe.

The remainder of this paper proceeds as follows. In Section 2, we survey related work on standards regarding spatial metadata and catalogue services. In Section 3, we present the architecture of TripleGeo-CSW by examining its components, the processing flow, and its current implementation status. In Section 4, we present a working case study on data discovery over INSPIRE catalogue services. Section 5 concludes the paper.

2. BACKGROUND & RELATED WORK

2.1 Catalogue Services for the Web (CSW)

Catalogue Services for the Web (CSW) is an OGC standard [18] that describes application profiles for publishing, accessing, and searching over collections of metadata on geospatial data, services, and related resources. This metadata must be encoded in XML and the schema of its records is usually conformant to more specific standards (like ISO 19139 [15], Dublin Core [6], or INSPIRE [7]). The spatial extent of a dataset is given with its bounding box encoded in GML [19]. Users may submit a number of different requests (either `GET` or `POST` HTTP methods) to a CSW and the response is encoded in an XML document as well. Typical requests that must be always supported by a CSW are:

- `GetCapabilities` can be used to retrieve metadata describing the type of requests the CSW can accept (e.g., version, acceptable parameters, output formats, etc.).
- `DescribeRecord` returns a description of the metadata records' model, i.e., an XML schema definition (XSD).
- `GetRecords` retrieves actual metadata records that satisfy criteria and filters specified in the request. Figure 4 illustrates one such request to CSW, asking for available geodata within a rectangular area (given in longitude/latitude coordinates) and matching specific textual criteria on the subject of the dataset and its associated keywords.
- `GetRecordsById` returns records matching a list of specific identifiers given as parameters in the request.

Other requests are non-mandatory for CSWs, like:

- `GetDomain` returns the range of values of a metadata record field or a request parameter.
- `Transaction` can be used to create metadata records, as well as to edit or delete existing ones.
- `Harvest` pulls metadata from third-party sources to create new records or update existing ones in the CSW.

2.2 Spatial Metadata as Linked Data Sources

There are mainly two (often complementary) approaches to *cataloguing linked metadata*. Data Catalogue Vocabulary (DCAT) [28] is an RDF vocabulary designed to facilitate interoperability between data catalogues published on the

```
<dct:conformsTo>
  <dct:Standard>
    <dct:title xml:lang='en'>
      <xsl:value-of select='//gmd:report//gmd:title/gco:CharacterString'/>
    </dct:title>
    <dct:issued rdf:datatype='http://www.w3.org/2001/XMLSchema#date'>
      <xsl:value-of select='//gmd:report//gco:Date' />
    </dct:issued>
  </dct:Standard>
</dct:conformsTo>
```

Figure 1: Excerpt of XSL stylesheet for transforming metadata elements into RDF.

Web. The VoID Vocabulary (VoID) [32] makes use of an RDF Schema vocabulary to express metadata about RDF datasets, and aims at data discovery, cataloguing and archiving. However, both approaches make extensive use of terms from other vocabularies, in particular Dublin Core [6].

Based on similar vocabularies, a few initiatives and case studies headed towards *linked metadata on spatial datasets*. Among them, the Mimas Linked Data Project for LandMap Spatial Discovery in the UK has made some preliminary work [17], mostly by identifying vocabularies and defining RDF mappings for a subset of their datasets. An open source prototype for Data Catalogue Vocabulary services based on DCAT is being implemented in GeoNetwork [11], and would eventually provide support to harvest, search and link catalogue contents with other interlinked resources. Public authorities across Europe have also begun publishing spatial metadata through SPARQL enpoints, such as the municipality of Zaragoza in Spain [35].

With regard to the particular task of exposing spatial metadata as linked open data, the crosswalking approach is suggested in [24, 16]. *Metadata crosswalking* involves mappings from popular geospatial metadata schemas to the Dublin Core vocabulary, addition of extra metadata elements, and finally expressing the metadata terms as RDF. The authors in [24] suggest an alternative method for publishing geospatial metadata provisioned by custom catalogue services as linked open metadata. In this case, RDF metadata terms are published directly from the UML representation of the underlying custom schemas.

The Joint Research Centre (JRC) of the European Commission has begun an exploratory investigation [23] regarding geospatial metadata on the Semantic Web. Of course, they mainly focus on alignment with the EU INSPIRE Directive [7] towards a LOD-enabled INSPIRE prototype, by creating a corpus of RDF metadata exposed via a SPARQL endpoint. Still, their preliminary version of RDF mappings for INSPIRE metadata elements offers a concrete RDF representation [9] for spatial metadata based on DCAT-AP and other relevant vocabularies (such as DCT, SKOS, vCard, etc.). In this work, we take advantage of such mappings and we offer generic stylesheets in XSL (EXtensible Stylesheet Language) [33], which can be used to transform XML files with OGC-compliant spatial metadata into an equivalent RDF representation. To the best of our knowledge, ours is the first attempt to offer application profiles in RDF for standardized geospatial metadata through catalogue services.

3. MIDDLEWARE ARCHITECTURE

In this Section, we present TripleGeo-CSW, an open-source middleware for data discovery from geospatial catalogue services on the Semantic Web. We first describe the way that spatial metadata elements can be translated into RDF triples. Then, we analyze the processing flow in TripleGeo-CSW, as well as its capabilities of searching against CSW with multiple criteria via a virtual GeoSPARQL endpoint.

Table 1: Some RDF mappings for spatial metadata.

Metadata element	RDF mapping of attribute
Resource title	`dct:title`
Resource language	`dct:language`
Keyword	`dcat:keyword`
Geographic Bounding Box	`dct:spatial`
Responsible organization – Owner	`dct:rightsHolder`

3.1 Metadata Application Profiles in RDF

Although still a work-in-progress, the RDF mappings suggested by the JRC [9] offer a valuable basis to develop a methodology for transforming spatial metadata elements into RDF. Our goal is to facilitate such transformations on-the-fly, such that contents from existing CSWs can be made accessible through (Geo)SPARQL. We are mostly interested in exposing the spatial coverage of data, as well as the temporal range of their lifecycle (i.e., when data was created, published or modified). However, many more metadata elements are important as well, such as descriptions (e.g., title, abstract, subject, keywords), content assessments (like quality, provenance, or conformity), as well as their legal status (owner, license, point of contact, etc.). A few RDF mappings from indicative metadata elements to vocabularies such as DCAT and DCT are shown in Table 1. Once this metadata gets exposed on the Semantic Web, it may be potentially interlinked with other features, such as terms in code lists or multilingual thesauri [23].

In practice, we manually created an application profile for such metadata as a set of templates employed in XSLT transformation [33]. Our custom XSL stylesheet[1] accepts an XML file with metadata records obtained as a response from a request to a CSW. Once invoked with an XSLT parser, the stylesheet turns metadata elements into suitable RDF statements according to the mapping; the result is an RDF/XML representation of original OGC-compliant metadata records. The XSL stylesheet is generic, covers all elements, and can be reused against any metadata conforming to OGC/ISO specifications. The excerpt shown in Figure 1 refers to handling of elements related with dataset conformity. Regarding the geographical coverage, its bounding box can be suitably expressed either as a GeoSPARQL polygon

[1]Stylesheet `Metadata2RDF.xsl` is included in the source code [1]; it has been also integrated into our TripleGeo tool [22] for directly transforming locally stored metadata files from XML into RDF.

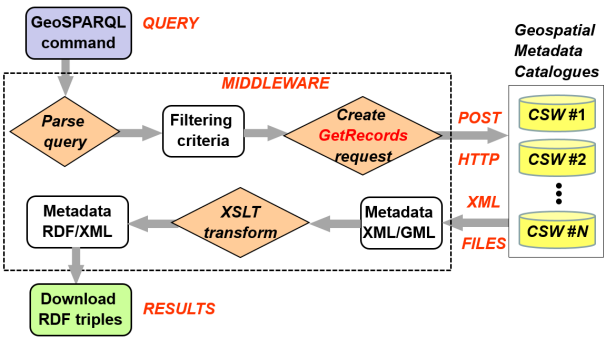


Figure 2: Flow diagram for processing GeoSPARQL queries in the TripleGeo-CSW middleware.

or a 2-dimensional rectangle `BOX2D`.

Our design adheres to reusing existing URIs as much as possible, especially in statements concerning spatial, temporal, and identification elements. However, blank nodes exist in the resulting triples, since these RDF mappings are based on the abstract schema of metadata elements. Such blank nodes are used as locally-scoped artifacts, which need not be explicitly labelled. Provided that the user is aware of the underlying schema (ISO/OGC), formulating (Geo)SPARQL queries against such metadata is straightforward.

3.2 Processing GeoSPARQL Queries over CSW

We assume that a list of catalogue services (CSW) exists, and each service is operational and accepts HTTP requests. In order to facilitate discovery of matadata from such CSWs through (Geo)SPARQL, we have implemented the TripleGeo-CSW middleware. The processing flow of this middleware is illustrated in Figure 2. It is triggered by a (Geo)SPARQL query, where the user can specify one or more conditions according to the spatial metadata ontology, as explained in Section 3.1. We developed a parser, which identifies several types of such conditions, including spatial predicates as documented next. The OGC standard defines a specific model for CSW requests (`POST/GET` HTTP protocols), which covers several cases. However, our major concern here is the CSW `<Filter>` element, which controls whether metadata should be retained according to specific criteria. Hence, the user-specified GeoSPARQL conditions must be internally rewritten and then integrated into the `<Filter>` element of a `GetRecords` request for CSW. Thanks to the OGC standard for CSW [18], an identical such request will be submitted simultaneously via `POST` HTTP protocol against each of the listed catalogues. Once each CSW provides its response as a collection of qualifying metadata records in a separate XML file. With the XSL stylesheet described in Section3.1, these metadata records (conforming to ISO 19115) are finally converted into XML/RDF triples and are available for download by the user.

Note that integrity and consistency of metadata information is a responsibility of the owners (governments, organizations, etc.), so each metadata element is supposed to come from a single CSW. Thus, resolving conflicts is not employed when compiling the resulting RDF triples from multiple sources. Of course, the final output is OGC-compliant metadata, since the XSLT transformation uses templates that map each metadata element into DCAT elements.

This open-source software has been developed in Python 2.7.3, and it makes use of several additional libraries:

- `urllib2`[2], a Python module for fetching URLs (Uniform Resource Locators) and posing requests;
- `re`[3] provides Perl-style regular expression pattern matching and is used for parsing such expressions in users' SPARQL requests;
- `etree`[4] performs XML parsing using the concepts of the ElementTree API for Python.

In its current release, TripleGeo-CSW can support user requests to discover whether there are any available, updated spatial datasets according to criteria that may involve a given geographical area, a certain thematic category (e.g., transport, hydrography), or particular keywords (e.g., "water", "rail"). More specifically, a (Geo)SPARQL query that can be handled by this middleware consists of two parts:

- a `SELECT` or `CONSTRUCT` clause identifies the attributes that will appear in the query results, and
- a `WHERE` clause provides the basic graph pattern to match against the metadata, as well as `FILTER` criteria.

Typically, a graph pattern in a `WHERE` clause consists of a triple with subject, predicate and object; this pattern is checked for matching against the metadata records. A pattern is formatted as `?s ?p ?o`, where `?s` is the sought element and `?o` is either a specific value (e.g., a string literal like "Environment") or a binding variable. Hence, search involves only triples satisfying match patterns `?s ?p ?o` (*Case 1*) or `?s ?p "literal"` (*Case 2*). In order to handle the matching candidates, we make use of a List and a Dictionary structure. The list is used for handling all triples with a variable as their object (*Case 1*). The dictionary is a set of `<key:value>` pairs with the requirement that keys are unique; so, it actually contains `<element:value>` pairs with unique metadata elements (*Case 2*). In the evaluation, triple patterns are checked one by one for matches, since multiple such criteria may be present in a query. Currently, no query optimization or check for syntax errors is performed; we defer dealing with such issues in future releases.

Concerning filtering, TripleGeo-CSW supports GeoSPARQL queries that may include any of the following criteria:

- *Matching regular expressions* (`REGEX`) against string literals. Through `FILTER` conditions in SPARQL, the user can check wildcard pattern matchings of string values (e.g., `"^water*"`) with keywords, titles, subjects, and other textual properties in the metadata.
- *Date comparisons* make use of typical operators (>, <, <=, >=) and a constant date value as an argument, in order to identify datasets issued, modified or published before or after that particular date.
- *Spatial filtering.* OGC-compliant metadata include the `BoundingBox` of the geographical extent for each dataset

[2] `https://docs.python.org/2/library/urllib2.html`
[3] `https://docs.python.org/2/library/re.html`
[4] `https://docs.python.org/2/library/xml.etree.elementtree.html`

```
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/geosparql/function/>
CONSTRUCT { ?m dcat:keyword ?k .       ?s dc:subject ?sub .      ?f geo:hasGeometry ?fWKT }
WHERE { ?m dcat:keyword ?k .
        ?s dc:subject "Environment" .
        ?f geo:hasGeometry ?fWKT .
        FILTER (REGEX(str(?k),"^water*") && geof:sfWithin(?fWKT, "BOX2D(-8.24 54.02,-5.18 55.32)"^^geo:wktLiteral)) };
```

Figure 3: Example GeoSPARQL query against spatial metadata exposed in CSW.

as an indication of its coverage area. Hence, it makes much sense to allow users discover availability of data in their region of interest, specified as a 2-dimensional rectangle (`BOX2D`) with four geographical coordinates. The parser recognizes typical GeoSPARQL topological predicates [20] like `sfWithin()`, `sfContains()`, `sfIntersects()`, `sfOverlaps()`, etc., which can be translated into equivalent CSW spatial filters over rectangles. For instance, operator `sfWithin()` checks whether a user-specified `BOX2D` is totally within the coverage of a dataset, `sfIntersects()` identifies whether a given `BOX2D` intersects the coverage of a dataset, etc.

Logical operators for conjunction (`&&`) and disjunction (`||`) can be used to combine filtering criteria, whereas a `UNION` clause can bind statements specifying alternative patterns (e.g., searching for datasets characterized by subjects like "road" or "rail"). For instance, issuing the GeoSPARQL query in Figure 3 will retrieve spatial metadata (in any of the listed CSWs) for environmental features related to water and within the given rectangular area. Note that no RDF graph is specified, as neither do we make use of a physically stored semantic repository nor any RDF triples get materialized or permanently retained. The TripleGeo-CSW middleware automatically rewrites this query into an equivalent `GetRecords` request (shown in Figure 4), which may be submitted to each of the available CSWs. Consequently, all RDF results are generated on-the-fly by XSLT transformation of the XML response received from these CSWs.

3.3 Implementation Status

TripleGeo-CSW is free software and its current version 1.0 is publicly available [1], including the source code in Python and several query examples. TripleGeo-CSW can be redistributed and modified under the terms of the GNU General Public License. The software can work in standalone mode, but it can also be coupled with a web interface.

A basic web interface consists of a client-side JavaScript application that allows the user to edit a query, specify the format for results, and download the qualifying RDF triples. In addition, a server-side PHP application acts both as an API proxy and an abstraction layer. Once it receives a query, this proxy validates it, then builds the properly formulated HTTP request against the list of available CSWs, and finally sends back the results to the user.

Adding or removing a CSW simply involves editing a configuration file in TripleGeo-CSW, i.e., adding or deleting the related URL of that catalogue service. In essence, all processing components are wrapped under this virtual GeoSPARQL endpoint, thus offering flexibility to interact directly with any number of remote catalogues and repurpose

```
<?xml version='1.0' encoding='utf-8'?>
<GetRecords
     xmlns="http://www.opengis.net/cat/csw/2.0.2"
     xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
     xmlns:ogc="http://www.opengis.net/ogc"
     xmlns:ows="http://www.opengis.net/ows"
     xmlns:dcat="http://www.w3.org/ns/dcat#"
     xmlns:dc="http://purl.org/dc/terms/"
     xmlns:gml="http://www.opengis.net/gml"
     xmlns:gmd="http://www.isotc211.org/2005/gmd"
     xmlns:apiso="http://www.opengis.net/cat/csw/apiso/1.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     service="CSW"
     version="2.0.2"
     startPosition="1"
     resultType="results"
     maxRecords="100"
     outputFormat="application/xml"
     outputSchema="http://www.isotc211.org/2005/gmd"
     xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2
     http://schemas.opengis.net/csw/2.0.2/CSW-discovery.xsd">
  <Query typeNames="gmd:MD_Metadata">
    <ElementSetName typeNames="gmd:MD_Metadata">full
    </ElementSetName>
    <Constraint version="1.1.0">
      <ogc:Filter>
        <ogc:And>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>dc:subject</ogc:PropertyName>
            <ogc:Literal>Environment</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsLike wildCard="^" singleChar="_">
            <ogc:PropertyName>dcat:keyword</ogc:PropertyName>
            <ogc:Literal>^water*</ogc:Literal>
          </ogc:PropertyIsLike>
          <ogc:Within>
            <ogc:PropertyName>ows:BoundingBox</ogc:PropertyName>
            <gml:Envelope>
              <gml:lowerCorner>-8.24 54.02</gml:lowerCorner>
              <gml:upperCorner>-5.18 55.32</gml:upperCorner>
            </gml:Envelope>
          </ogc:Within>
        </ogc:And>
      </ogc:Filter>
    </Constraint>
  </Query>
</GetRecords>
```

Figure 4: The `GetRecords` request to CSW corresponding to the GeoSPARQL query in Figure 3. Note that the spatial condition is translated into an equivalent enclosure within a `gml:Envelope` specified with the given geographical coordinates. The graph pattern and regular expression matching criteria in the query are respectively transformed into equivalent conditions `ogc:PropertyIsEqualTo` and `ogc:PropertyIsLike`, recognizable by CSW services.

Table 2: INSPIRE-aligned metadata available through several CSWs across Europe.

INSPIRE Discovery Service in the Czech Republic:	http://geoportal.cuzk.cz/SDIProCSW/service.svc/get?request=GetCapabilities&service=CSW
Estonian National Geoportal:	http://inspire.maaamet.ee/geoportal/csw/discovery?request=GetCapabilities&Service=csw&language=eng
Irish Spatial Data Exchange:	http://catalogue.isde.ie/geonetwork/srv/en/csw?request=GetCapabilities&service=CSW
National CSW for Norway:	http://www.geonorge.no/geonetwork/srv/nor/csw-inspire?service=CSW&request=GetCapabilities
Discovery Service for the UK Location catalogue:	http://csw.data.gov.uk/geonetwork/srv/en/csw?request=GetCapabilities&service=CSW
Spanish National Geographic Institute:	http://www.ign.es/csw-inspire/srv/eng/csw?Service=CSW&Request=GetCapabilities
Metadata Catalogue of the SDI for Spain:	http://www.idee.es/csw-inspire-idee/srv/eng/csw?request=GetCapabilities&service=CSW

their spatial metadata. The only prerequisite for such catalogues is that they must be compatible with the OGC standard for CSW [18] and thus support the related requests, as discussed in Section 2.1.

4. A USE CASE: DISCOVERING INSPIRE THROUGH GEOSPARQL QUERIES

In this Section, we present a use case where TripleGeo-CSW has been applied in practice. This validation of the middleware concerns discovery of INSPIRE-aligned spatial datasets from catalogue services across Europe through a virtual GeoSPARQL endpoint.

4.1 INSPIRE as a Source for Linked Data

The INSPIRE Directive 2007/2/EC [7] sets a unified framework for Spatial Data Infrastructures (SDI) across the EU, so that by 2020 spatial information can be shared among European public authorities in order to assist in environmental policies. Its foundations include technical interoperability standards for geospatial metadata, data and online services, as well as uniform legal rules for data interchange and reuse. Towards establishing such a pan-European SDI, INSPIRE specifications prescribe catalogues of available resources using metadata, common access policies and standards, as well as network services for discovery, viewing, downloading, transformation, etc. for spatial datasets.

Implementing Rules [8] for INSPIRE-compliant metadata propose a schema for describing datasets, dataset series, services and thematic layers across Europe. This schema is designed according to ISO standards [12, 13] and contains metadata elements for data regarding its identification, topic, quality, geographical and temporal extent, as well as points of contact with the responsible parties. In addition, ISO-19119 [14] defines a framework for developing services that can be used to access and process geospatial data. This framework supports access to different data sources through a generic, platform-neutral application interface. INSPIRE metadata should not violate these ISO standards, but since the latter require many more elements (e.g., points of contact, restrictions) these have to be provisioned as well. On the other hand, metadata published according to the ISO-19115 core is not guaranteed to conform with the INSPIRE ontology, so an alignment is necessary.

Unfortunately, no complete INSPIRE ontology in RDF/OWL [29] currently exists. This reflects the difficulty of bridging the "closed world" assumption of UML models in INSPIRE with the "open world" view of RDF. Admittedly, this limitation refers not only to INSPIRE, since exposing geospatial information as open linked data is a relatively new research topic. Especially for INSPIRE SDIs, some prominent opportunities of utilizing linked open data have been highlighted [25] by the Joint Research Centre of the European Commission, along with the requirements for achieving it. Exposing INSPIRE datasets as linked data has attracted some research interest. The proposed approaches either translate INSPIRE-compliant GML data models as semantic OWL ontologies [26], or generate an ontology model mixing a number of different existing ontologies and vocabularies along with tools for RDF extraction and interlinking [27], or even deriving linked data from GML data and reusing existing concepts from vocabularies [5].

In contrast to the aforementioned approaches on spatial data, there has not been any attempt to expose INSPIRE *metadata* from existing catalogues according to the GeoSPARQL standard [20], as we present next. Our TripleGeo-CSW suite for the Semantic Web can not only be used by stakeholders that wish to make their SDI contents accessible in RDF, but also for discovering available third-party data via GeoSPARQL requests against CSWs.

4.2 Data Discovery from INSPIRE CSWs

Catalogue services for INSPIRE-compliant metadata have become already available in various European countries, even in non-EU member states like Norway, as indicated in Table 2. Our work is focused on exposing such CSWs on the Semantic Web through our CSW-to-RDF middleware TripleGeo-CSW. In short, we wish to enable GeoSPARQL queries with user-specified criteria against the contents of such catalogues, so as to facilitate INSPIRE data discovery. In this case, TripleGeo-CSW acts as a broker between a virtual GeoSPARQL endpoint and a list of INSPIRE-compliant CSWs, and undertakes to request any available information from the CSWs, collect the partial XML results, and finally return any qualifying metadata as RDF triples.

Towards this goal, we have made use of INSPIRE metadata from CSWs across Europe (Table 2). We stress that this is just an indicative list of currently operating CSWs. Of course, this list may be extended as more INSPIRE-compliant such services become available, without necessitating absolutely any change in our existing framework. It only requires including any additional CSW into the list of such services, i.e., editing the respective configuration file that is accessible by the middleware.

In order to provide a simple and uniform interface to end-users, we have implemented a web application that offers the ability to issue GeoSPARQL queries against CSWs and receive response in a variety of formats (RDF/XML, CSV, HTML, etc.). This web interface is publicly available at:

`http://geodata.gov.gr/sparql/`

Users wishing to explore available INSPIRE geodata across Europe must choose *"A collection of INSPIRE CSW catalogues"* as their (virtual) target store. We stress that no triple store is used to physically retain any RDF metadata received from such CSW services. Instead, qualifying metadata records are collected in XML and transformed on-the-fly into a RDF serialization.

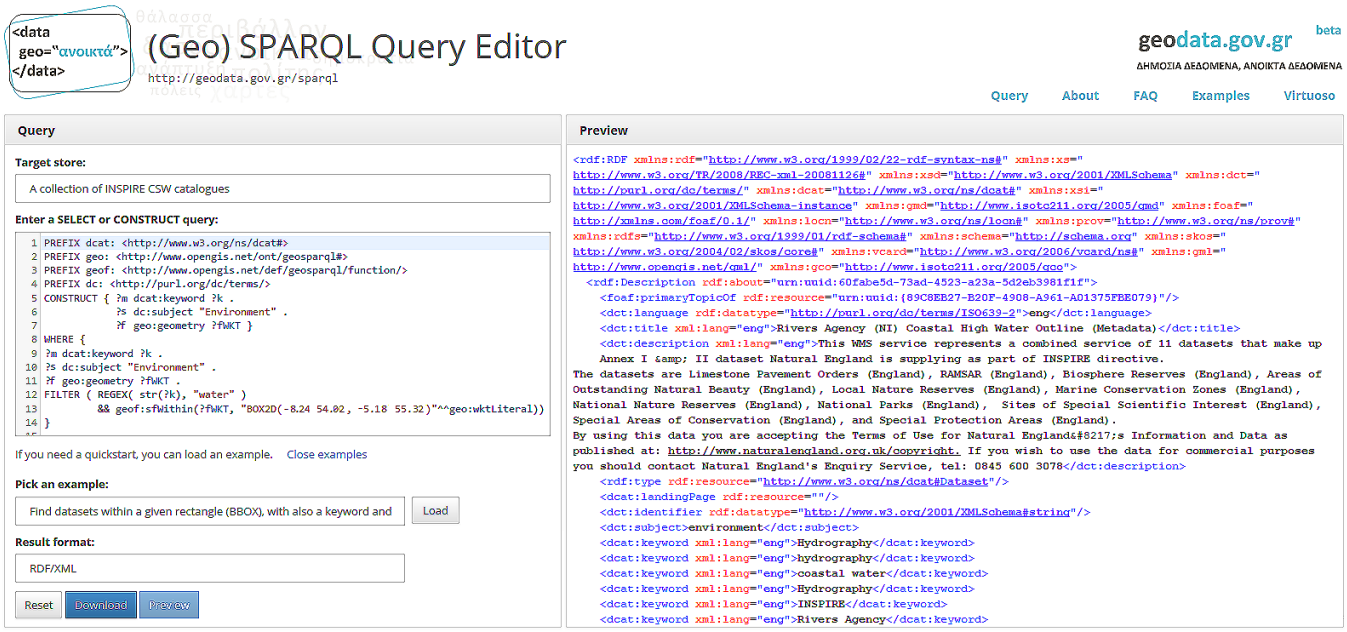


Figure 5: The virtual GeoSPARQL endpoint at `http://geodata.gov.gr/sparql/` over INSPIRE CSWs.

This web interface (illustrated in Figure 5) includes a few predefined (Geo)SPARQL query examples against these CSW services. We have employed `CONSTRUCT` queries in order to receive results as RDF triples, and also verify the robustness of our middleware and validate its functionality. These queries explore a wide range of metadata features, e.g., keywords, subjects, titles, as well as the geographical area covered by the INSPIRE datasets referenced in the CSWs. Indicatively, users can:

- Search for datasets tagged with a given *keyword* (e.g., "administrative");
- Find available datasets that specify the given *subject* (like "Environment") in the metadata;
- Find datasets with spatial coverage inside a given rectangle (i.e., *Bounding Box*);
- Identify datasets on a given *subject* (e.g., "Environment") and whose *title* includes a particular term (e.g., "network").

Users may submit such queries "as is", modify them to reflect their specific search criteria, and of course, write their own queries in order to discover INSPIRE-compliant datasets offered by the available CSWs.

5. SUMMARY

In this paper, we introduced an open-source software that can be used to repurpose existing catalogue services (CSW) on geospatial metadata as high quality Linked Data sources. In effect, TripleGeo-CSW acts as a CSW-to-RDF middleware, which translates a given GeoSPARQL query into an equivalent request for available metadata records against multiple CSWs. As soon as the response is collected, the original XML metadata elements are transformed on-the-fly into RDF triples and returned as answers.

As a proof of concept, we have successfully enabled users to search for INSPIRE datasets from remote CSW services across Europe, by providing a virtual GeoSPARQL interface on top of TripleGeo-CSW. This ensures that INSPIRE Catalogue Services are accessible with Semantic Web technologies and thus INSPIRE data are discoverable with negligible overhead from stakeholders.

6. ACKNOWLEDGEMENTS

This work was partially supported by the European Commission under FP7-ICT-2011-8 grant #318159 *"GeoKnow – Making the Web an Exploratory Place for Geospatial Knowledge"* and FP7-ICT-2013-SME-DCA grant #609608 *"PublicaMundi – Scalable and Reusable Open Geospatial Data"*. We also wish to thank Andrea Perego and Michael Lutz (European Commission – JRC) for helpful discussions regarding RDF mappings of INSPIRE-aligned metadata.

7. REFERENCES

[1] Athena R.C. TripleGeo-CSW open source middleware. `https://github.com/GeoKnow/TripleGeo-CSW`

[2] R. Battle and D. Kolas. GeoSPARQL: Enabling a Geospatial Semantic Web. *Semantic Web Journal*, 3(4): 355-370, 2012.

[3] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data – The Story So Far. *IJSWIS*, 5(3): 1-22, 2009.

[4] DBpedia. `http://dbpedia.org`

[5] L. van den Brink, P. Janssen, W. Quak, and J. Stoter. Linking spatial data: semi-automated conversion of geo-information models and GML data to RDF. *IJSDIR*, 9: 59-85, 2014.

[6] Dublin Core Metadata Initiative. Dublin Core Metadata element set, Version 1.1. July 1999. `http://dublincore.org/documents/dcmi-terms/`

[7] European Commission (EC). INSPIRE Directive –

Infrastructure for Spatial Information in the European Community. http://inspire.jrc.ec.europa.eu/
[8] EC. INSPIRE Implementing Rules. http://inspire.ec.europa.eu/index.cfm/pageid/47
[9] EC. Alignment of INSPIRE metadata with DCAT-AP. https://ies-svn.jrc.ec.europa.eu/projects/metadata/wiki/Alignment_of_INSPIRE_metadata_with_DCAT-AP
[10] GeoNames database. http://www.geonames.org/
[11] GeoNetwork Data Catalog Vocabulary services. http://trac.osgeo.org/geonetwork/wiki/proposals/DCATandRDFServices
[12] ISO 19115:2003. Geographic information – Metadata. http://www.iso.org/iso/catalogue_detail.htm?csnumber=26020
[13] ISO 19115-1:2014. Geographic information – Metadata – Part 1: Fundamentals. http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=53798
[14] ISO 19119:2005. Geographic Information – Services. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=39890
[15] ISO/TS 19139:2007. Geographic Information – Metadata – XML schema implementation. http://www.iso.org/iso/catalogue_detail.htm?csnumber=32557
[16] F.J. Lopez-Pellicer, A.J. Florczyk, J. Nogueras-Iso, P.R. Muro-Medrano and F. J. Zarazaga-Soria. Exposing CSW Catalogues as Linked Data. In *Geospatial Thinking*, pp. 183-200, 2010.
[17] Mimas Linked Data Project (UK). http://mimasld.wordpress.com/
[18] Open Geospatial Consortium (OGC). Catalogue Service. http://www.opengeospatial.org/standards/cat
[19] OGC Geography Markup Language Encoding Standard, 2007. http://portal.opengeospatial.org/files/?artifact_id=20509
[20] OGC GeoSPARQL Standard - A Geographic Query Language for RDF Data, 2012. https://portal.opengeospatial.org/files/?artifact_id=47664
[21] OpenStreetMap project. http://www.openstreetmap.org/
[22] K. Patroumpas, M. Alexakis, G. Giannopoulos, and S. Athanasiou. TripleGeo: an ETL Tool for Transforming Geospatial Data into RDF Triples. In *LWDM*, pp. 275-278, 2014.
[23] A. Perego. Inspiring Data? Cross-domain Interoperability for EU Spatial Data. In *Using Open Data Workshop*, Brussels, Belgium, June 2012.
[24] J. Reid, W. Waites, and B. Butchart. An Infrastructure for Publishing Geospatial Metadata as Open Linked Metadata. In *AGILE*, 2012.
[25] S. Schade and M. Lutz. Opportunities and Challenges for using Linked Data in INSPIRE. In *Workshop on Linked Spatiotemporal Data*, 2010.
[26] S. Tschirner, A. Scherp, and S. Staab. Semantic access to INSPIRE – How to publish and query advanced GML data. In *Terra Cognita*, pp. 75-87, 2011.
[27] L.M. Vilches-Blázquez, B. Villazón-Terrazas, V. Saquicela, A. de León, O. Corcho, and A. Gómez-Pérez. GeoLinked Data and INSPIRE through an Application Case. In *ACM SIGSPATIAL GIS*, pp. 446-449, November 2010.
[28] W3C. Data Catalog Vocabulary (DCAT). http://www.w3.org/TR/vocab-dcat/
[29] W3C. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl2-overview/
[30] W3C. Resource Description Framework 1.1. http://www.w3.org/TR/rdf11-new/
[31] W3C. SPARQL 1.1 Query Language for RDF. http://www.w3.org/TR/sparql11-query/
[32] W3C. VoID Vocabulary (3/3/2011). http://www.w3.org/TR/void/
[33] W3C. XSL Transformations (XSLT). http://www.w3.org/TR/xslt
[34] Wikimapia. http://wikimapia.org
[35] Zaragoza municipality SPARQL endpoint. http://www.zaragoza.es/datosabiertos/sparql

Frequent Subgraph Mining from Streams of Linked Graph Structured Data

Alfredo Cuzzocrea
ICAR-CNR & Uni. Calabria
Rende (CS), Italy
cuzzocrea@si.deis.unical.it

Fan Jiang
University of Manitoba
Winnipeg, MB, Canada
umjian29@cs.umanitoba.ca

Carson K. Leung
University of Manitoba
Winnipeg, MB, Canada
kleung@cs.umanitoba.ca

ABSTRACT

Nowadays, high volumes of high-value data (e.g., semantic web data) can be generated and published at a high velocity. A collection of these data can be viewed as a big, interlinked, dynamic graph structure of linked resources. Embedded in them are implicit, previously unknown, and potentially useful knowledge. Hence, efficient knowledge discovery algorithms for mining frequent subgraphs from these dynamic, streaming graph structured data are in demand. Some existing algorithms require very large memory space to discover frequent subgraphs; some others discover collections of frequently co-occurring edges (which may be disjoint). In contrast, we propose—in this paper—algorithms that use *limited memory* space for discovering collections of frequently co-occurring *connected* edges. Evaluation results show the effectiveness of our algorithms in frequent subgraph mining from streams of linked graph structured data.

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures—*graphs and networks*; E.2 [**Data**]: Data Storage Representations—*linked representations*; H.2.8 [**Database Management**]: Database Applications—*data mining*

General Terms

Algorithms; Design; Experimentation; Management; Performance; Theory

Keywords

Data mining, frequent patterns, graph structured data, linked data, extending database technology, database theory

1. INTRODUCTION

Nowadays, high volumes of valuable semantic web, life science, social network, or bibliographical network data can be generated from diverse real-life applications [3, 14, 23]. For example, semantic web data—such as blogs, forums, wikis, and users' reviewers—can be published and connected at a high velocity as the web enables users to link related resources (e.g., related documents and related data). These *linked data* [19] are commonly published by using technologies like (i) uniform resource identifiers (URIs) which identify resources, (ii) hypertext transfer protocol (HTTP) which retrieves or describes resources, and (iii) resource description framework (RDF) which graphically models linkage among resources. A collection of these data can be viewed as a big, interlinked, and dynamic graph structure of linked resources. Embedded in these data are implicit, previously unknown, and potentially useful knowledge. Having techniques for modelling, querying, and reasoning these linked data [13, 15] is desirable. In this paper, we focus on mining frequent subgraphs from these dynamic streaming graph structured data. Note that some existing algorithms require very large memory space to mine frequent subgraphs; some others discover collections of frequently co-occurring edges (which may be disjoint). In contrast, we propose—in this paper—algorithms that use *limited memory* space for discovering collections of frequently co-occurring *connected* edges.

1.1 Related Works

Since the introduction of the frequent pattern mining problem [2], numerous algorithms have been proposed [25, 27, 28]. For example, FP-growth [18] uses an in-memory extended prefix-tree structure called Frequent Pattern tree (FP-tree)—which captures the content of the transaction database—for mining sets of frequently co-occurring items (e.g., shopper market baskets of frequently purchased merchandise items) from traditional static databases (e.g., containing shopper market transactions). Some works [6, 17] use disk-based structure for mining. However, they mine from *static* databases.

As technology advances, *dynamic* streams of graph structured data (e.g., streams of semantic web, sensor network, social network, and road network data [10]) can be easily generated at high velocity. When comparing with mining from traditional *static* databases, mining from *dynamic* data streams [20, 29, 30] is more challenging due to the following properties of data streams: (i) *Data streams are continuous and unbounded.* To find frequent patterns from streams, we no longer have the luxury of performing multiple data scans. Once the streams flow through, we lose them. Hence, we need some data structures to capture the important contents of the streams (e.g., recent data—because users are usually more interested in recent data than older ones [11, 12]). (ii) *Streaming data are not necessarily uniformly distributed; their distributions are usually changing with time.*

A currently infrequent pattern may become frequent in the future, and vice versa. So, we have to be careful not to prune infrequent patterns too early; otherwise, we may not be able to get complete information such as frequencies of certain patterns (as it is impossible to retract those pruned patterns). To mine frequent patterns from data streams, both approximate and exact algorithms have been proposed. For instance, *approximate* algorithms (e.g., FP-streaming [16], TUF-streaming [24]) focus mostly on efficiency. However, due to approximate procedures, these algorithms may find some infrequent patterns or miss frequency information of some frequent patterns (i.e., some false positives or negatives). An *exact* algorithm mines only truly frequent patterns (i.e., no false positives and no false negatives) by (i) constructing a Data Stream Tree (DSTree) [26] to capture contents of the streaming data and then (ii) recursively building FP-trees for projected databases based on the information extracted from the DSTree.

The aforementioned properties play an important role in the mining of data streams in general; they play a more challenging role in the mining of a specific class of streaming data—namely, *streams of graph structured data*. State-of-the-art solutions to these challenges include the following: Aggarwal et al. [1] studied the research problem of mining dense patterns in graph streams, and they proposed probabilistic algorithms for determining such structural patterns effectively and efficiently. Bifet et al. [4] mined frequent closed graphs on evolving data streams. Their three innovative algorithms work on coresets of closed subgraphs, compressed representations of graph sets, and maintain such sets in a batch-incremental manner. Moreover, Valari et al. [31] discovered top-k dense subgraphs in dynamic graph collections by means of both exact and approximate algorithms. Furthermore, Chi et al. [9] proposed a fast graph stream classification algorithm that uses discriminative clique hashing (DICH), which can be applicable for OLAP analysis over evolving complex networks. We [5, 7] previously mined frequent patterns—in the form of collections of frequently co-occurring edges—from dense graph streams.

1.2 Our Contributions

Our previous solution [7] finds collections of frequently co-occurring edges, which include *connected* as well as *disjoint* edges. In many real-life situations (e.g., social or business applications [21, 22]), it is desirable to obtain collections of frequent disjoint edges so as to help the discovery of the missing links (e.g., connect two or more disjoint groups of social entities sharing common research or business interests). However, in some other situations, it is more efficient to find only the collections of frequent *connected* edges. Hence, in this paper, we propose algorithms that find collections of frequently co-occurring *connected* edges from streaming graph structured data. The algorithms either prune irrelevant (disjoint) edges at a post-processing step or push such a prune step early in the mining process. Consequently, only relevant patterns (i.e., frequent *connected* subgraphs) are returned to users. Moreover, as high volumes of streaming graph structured data can be generated at a high velocity, data may be too big to fit into memory. Our algorithms were designed in such a way that they use *limited memory*.

This paper is organized as follows. Background is provided in Section 2. Section 3 presents our algorithms that first build an on-disk data structure to capture and maintain relevant streaming graph structured data, recursively discover collections of frequent edges, and then prune those disjoint edges at a post-processing step. Section 4 presents an improved algorithm that pushes the prune step early in the mining process. Section 5 shows experimental results. Finally, conclusions are given in Section 6.

2. BACKGROUND

In this section, we provide background information on three different structures for capturing streaming data.

2.1 DSTree

When mining frequent patterns from streaming data, an exact algorithm [26] first constructs a *Data Stream Tree* (*DSTree*), which is then used as a *global tree* for recursive generation of smaller FP-trees (as *local trees*) for projected databases. Due to the dynamic nature of data streams, frequencies of items are continuously affected by the insertion of new batches (and the removal of old batches) of transactions. Arranging items in frequency-dependent order may lead to swapping—which, in turn, can cause merging and splitting—of tree nodes when frequencies change. Hence, in the DSTree, transaction items are arranged according to some canonical order (e.g., alphabetical order), which can be specified by the user prior to the tree construction or mining process. Consequently, the DSTree can be constructed using only a single scan of the streaming data. Note that the DSTree is designed for processing streams within a sliding window. For a window size of w batches, each tree node keeps (i) an item and (ii) a list of w frequency values (instead of a single frequency count in each node as in the FP-tree for frequent pattern mining from *static* databases). Each entry in this list captures the frequency of an item in each batch of dynamic streams in the current window. By so doing, when the window slides (i.e., when new batches are inserted and old batches are deleted), frequency information can be updated easily. Consequently, the resulting DSTree preserves the usual tree properties that (i) the total frequency (i.e., sum of w frequency values) of any node is at least as high as the sum of total frequencies of its children and (ii) the ordering of items is unaffected by the continuous changes in item frequencies.

After the construction of the (global) DSTree, it is always kept up-to-date when the window slides. The actual mining process is "delayed" until it is needed. To start mining, the algorithm first traverses relevant tree paths upwards and sums the frequency values of each list in a node representing an item (or a set of items)—to obtain its frequency in the current sliding window—for forming an appropriate projected database. Afterwards, the algorithm constructs a (local) FP-tree for the projected database of each of these frequent patterns of only 1 item (i.e., 1-itemset) such as an $\{x\}$-projected database (in a similar fashion as in the FP-growth algorithm for mining static data [18]). Thereafter, the algorithm recursively forms subsequent FP-trees for projected databases of frequent k-itemsets where $k \geq 2$ (e.g., $\{x, y\}$-projected database, $\{x, z\}$-projected database, etc.) by traversing paths in these FP-trees. As a result, the algorithm finds all frequent patterns. Note that, as items are consistently arranged according to some canonical order, the algorithm guarantees the inclusion of all *frequent* items using just upward traversals. Moreover, there is also no worry about possible omission or double-counting of items during

the mining process. Furthermore, as the DSTree is always kept up-to-date, all frequent patterns—which are embedded in batches within the current sliding window—can be found effectively.

2.2 DSTable

The success of mining with the DSTree mainly relies on the assumption—usually made for many tree-based algorithms [18]—that all tree (i.e., the global tree together with subsequent FP-trees) fit into the memory. For example, when mining frequent patterns from the $\{x, y, z\}$-projected database, the global tree and three subsequent local FP-trees (for the $\{x\}$-, $\{x, y\}$- and $\{x, y, z\}$-projected databases) are all assumed to be fit into memory. However, there are situations (e.g., for streaming graph structured data) where the memory is so limited that not all these trees can fit into memory. To handle these situations, the *Data Stream Table* (*DSTable*) [8] was proposed. The DSTable is a two-dimensional table that captures on the disk the contents of transactions in all batches within the current sliding window. Each row of the DSTable represents a domain item. Like the DSTree, items in the DSTable are arranged according to some canonical order (e.g., alphabetical order), which can be specified by the user prior to the construction of the DSTable. As such, table construction requires only a single scan of the stream. Each entry in the resulting DSTable is a "pointer" that points to the location of the table entry (i.e., which row and which column) for the "next" item in the same transaction. When dealing with streaming data, the DSTable also keeps w boundary values (to represent the boundary between w batches in the current sliding window) for each item. By doing so, when the window slides, transactions in the old batch can be removed and transactions in the new batch can be added easily.

Once the DSTable is constructed and updated, the algorithm first extracts relevant transactions from the DSTable. Then, the algorithm (i) constructs an FP-tree for the projected database of each of these 1-itemsets and (ii) recursively forms subsequent FP-trees for projected databases of frequent k-itemsets (where $k \geq 2$) by traversing the paths of these FP-trees. On the positive side, the algorithm finds all frequent patterns. On the negative side, to facilitate easy insertion and deletion of contents in the DSTable when the window (of size w batches) slides, the DSTable keeps w boundary values for each row (representing each of the m domain items). Hence, the DSTable needs to keep a total of $m \times w$ boundary values. Moreover, each table entry is a "pointer" that indicates the location in terms of row name and column number of the table entry for the "next" item in the same transaction. When the data stream is sparse, only a few "pointers" need to be stored. However, when the graph stream is dense, many "pointers" need to be stored. Given a total of $|T|$ transactions in all batches within the current sliding window, there are potentially $m \times |T|$ "pointers" (where m is the number of domain items). Furthermore, during the mining process, multiple FP-trees need to be constructed and kept in memory (e.g., FP-trees for all $\{a\}$-, $\{a, c\}$- and $\{a, c, d\}$-projected databases are required to be kept in memory).

2.3 DSMatrix

The use of a two-dimensional structure called *Data Stream Matrix* (*DSMatrix*) [7] solves the aforementioned problems while mining frequent patterns from data streams with limited memory because this matrix structure captures the contents of transactions in all batches within the current sliding window by storing them on the disk. The DSMatrix is a binary matrix, which represents the presence of an item x in transaction t_i by a "1" in the matrix entry (t_i, x) and the absence of an item y from transaction t_j by a "0" in the matrix entry (t_j, y). With this binary representation of items in each transaction, each column in the DSMatrix captures a transaction. Each column in the DSMatrix can be considered as a bit vector. The DSMatrix keeps track of any boundary between two batches so that, when the window slides, transactions in the older batches can be easily removed and transactions in the newer batches can be easily added. Unlike the DSTable (in which boundaries may vary from one row representing an item to another row representing another item due to the potentially different number of items present), boundaries in DSMatrix are the same from one row to another because we put a binary value (0 or 1) for each transaction. Hence, the DSMatrix only keeps w boundary values (where $w \ll m \times w$) for the entire matrix, regardless how many domain items (m) are here. Moreover, as DSMatrix uses a bit vector to indicate the presence or absence of items in a transaction, the computation does not require us to keep track of the index of the last item in every row and thus incurring a lower computation cost. Given a total of $|T|$ transactions in all batches within the current sliding window, there are $|T|$ columns in our DSMatrix. Each column requires only m bits. In other words, the DSMatrix takes $m \times |T|$ bits (cf. potentially $64m \times |T|$ bits for dense data streams required by the DSTree).

3. FREQUENT CONNECTED SUBGRAPH MINING WITH A POST-PROCESSING STEP

To find collections of frequent edges in streams of graph structured data, our proposed algorithms first construct a DSMatrix to capture and maintain within the current window those relevant streaming data. When a new batch of streaming graph structured data comes in, the window slides. Transactions in the oldest batch in the sliding window are then removed from the DSMatrix so that transactions in this new batch can be added. In other words, the mining is "delayed" until it is needed. Once the DSMatrix is constructed, it is kept up-to-date on the disk. See Example 1.

Example 1. For illustrative purpose, let us consider a sliding window of size $w = 2$ batches (i.e., only two batches are kept) and the following stream of graphs, where each graph $G = (V, E)$ consists of $|V| = 4$ vertices (Vertices v_1, v_2, v_3 and v_4) and $|E| \leq 6$ edges:

- At time T_1, $E_1 = \{(v_1, v_4), (v_2, v_3), (v_3, v_4)\}$;
- At time T_2, $E_2 = \{(v_1, v_2), (v_2, v_4), (v_3, v_4)\}$;
- At time T_3, $E_3 = \{(v_1, v_2), (v_1, v_4), (v_3, v_4)\}$;
- At time T_4, $E_4 = \{(v_1, v_2), (v_1, v_4), (v_2, v_3), (v_3, v_4)\}$;
- At time T_5, $E_5 = \{(v_1, v_2), (v_2, v_3), (v_2, v_4), (v_3, v_4)\}$;
- At time T_6, $E_6 = \{(v_1, v_2), (v_1, v_3), (v_1, v_4)\}$;
- At time T_7, $E_7 = \{(v_1, v_2), (v_1, v_4), (v_3, v_4)\}$;
- At time T_8, $E_8 = \{(v_1, v_2), (v_1, v_4), (v_2, v_3), (v_3, v_4)\}$;
- At time T_9, $E_9 = \{(v_1, v_3), (v_1, v_4), (v_2, v_3)\}$.

See Figure 1. These graphs may represent some insertions, deletions, and/or updates on the linkages among linked data or documents in a semantic web. For simplicity, we represent these edges by six symbols a, b, c, d, e and f. Consequently, we get (i) edges $E_1 = \{c, d, f\}$, $E_2 = \{a, e, f\}$ and $E_3 = \{a, c, f\}$ in the first batch B_1; as well as (ii) edges $E_4 = \{a, c, d, f\}$, $E_5 = \{a, d, e, f\}$ and $E_6 = \{a, b, c\}$ in the second batch B_2. Then, the DSMatrix stores the following information at the end of time T_6:

DSMatrix (capturing E_1–E_6):

Boundaries: Cols 3 & 6	
Row	Contents
Row a:	0 1 1; 1 1 1
Row b:	0 0 0; 0 0 1
Row c:	1 0 1; 1 0 1
Row d:	1 0 0; 1 1 0
Row e:	0 1 0; 0 1 0
Row f:	1 1 1; 1 1 0

DSMatrix keeps track of the global boundary information (which is applicable for all rows).

When the third batch B_3 of streaming graph structured data flows in, the window slides. DSMatrix uses the boundary information to remove data in all columns up to Col 3 while keeping data in Col (3+1) to Col 6 (or more precisely, shifting all columns from Cols 4–6 to Cols 1–3. After the removal of the first three columns, DSMatrix appends three columns representing (iii) edges $E_7 = \{a, c, f\}, E_8 = \{a, c, d, f\}$ and $E_9 = \{b, c, d\}$ in the third batch B_3. In other words, DSMatrix stores the following information for E_4–E_9 in batches B_2 & B_3 at the end of time T_9.

DSMatrix (capturing E_4–E_9):

Boundaries: Cols 3 & 6	
Row	Contents
Row a:	1 1 1; 1 1 0
Row b:	0 0 1; 0 0 1
Row c:	1 0 1; 1 1 1
Row d:	1 1 0; 0 1 1
Row e:	0 1 0; 0 0 0
Row f:	1 1 0; 1 1 0

Again, DSMatrix keeps track of the global boundary information (which is applicable for all rows). □

3.1 Mining with Multiple FP-trees

After constructing a DSMatrix, our first algorithm extracts columns from the DSMatrix to build a tree in memory for each $\{x\}$-projected database (which is a collection of all the edges containing x). Afterwards, the algorithm recursively finds collections of frequent edges from the tree for this projected database. See Example 2.

Example 2. At the end of time T_9, our first algorithm mines frequent patterns with the DSMatrix capturing E_4–E_9 in Example 1 by first forming the $\{a\}$-projected database. We examine Row a. For every column with a value "1", we extract its column downwards (e.g., from edges/items b to e if they exist). Specifically, when examining Row a, we notice that columns 1, 2, 3, 4 and 6 contain values "1" (which means that edges a appears in those five graphs in the two batches of streaming graph structured data in the current sliding window). Then, from Column 1, we extract $\{c, d, f\}$. Similarly, we extract $\{d, e, f\}$ and $\{b, c\}$ from Columns 2 and 3. We also extract $\{c, f\}$ and $\{c, d, f\}$ from Columns 4 and 5. All these form the $\{a\}$-projected database, from which an FP-tree can be built. From this FP-tree for the $\{a\}$-projected database, we find that edge-pairs $\{a, c\}, \{a, d\}$ and $\{a, f\}$ are frequent. Hence, we then form $\{a, d\}$- and $\{a, f\}$-projected databases, from which FP-trees can be built. (Note that we do not need to form the $\{a, c\}$-projected database as it is empty after forming both $\{a, d\}$- and $\{a, e\}$-projected databases.) When applying this step recursively in a depth-first manner, we obtain frequent edge-triplets $\{a, c, d\}, \{a, c, f\}$ and $\{a, d, f\}$, which leads to FP-trees for the $\{a, d, c\}$-projected database. (Again, we do not need to form the $\{a, f, c\}$- or $\{a, d, f\}$-projected databases as they are both empty.) At this moment, we keep FP-trees for the $\{a\}$-, $\{a, d\}$- and $\{a, d, c\}$-projected databases. Afterwards, we also find that edge-quadruplet $\{a, c, d, f\}$ is frequent. In the context of graph streams, this is a frequent collection of 4 edges—namely, Edges a, c, d and f. To recap, in addition to the five frequent singletons (i.e., edges a, b, c, d and f), a total of seven collections of frequent edges were found from the $\{a\}$-projected database: $\{a, c\}, \{a, c, d\}, \{a, c, d, f\}, \{a, c, f\}, \{a, d\}, \{a, d, f\}$ & $\{a, f\}$.

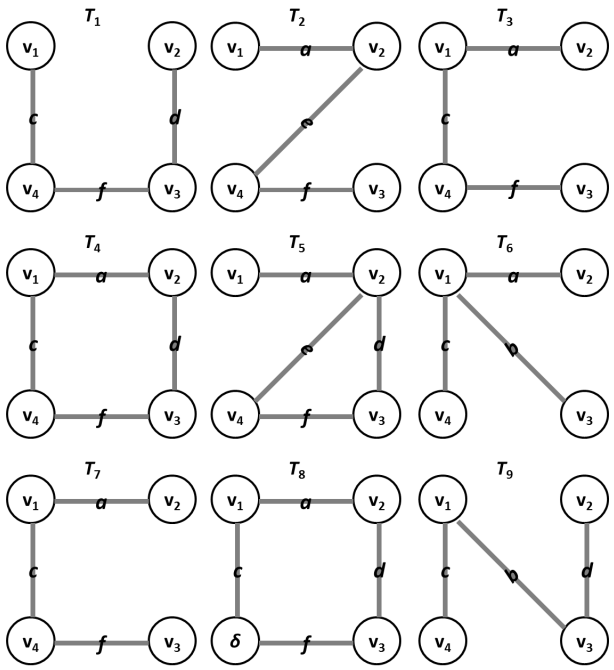


Figure 1: A stream of graph structured data (Example 1).

Afterward, we backtrack and examine the next frequent singleton $\{b\}$. For Row b, we notice that Columns 3 and 6 contain values "1" (which means that b appears in those two graphs in the current sliding window). For these two columns, we extract downward to get $\{c\}$ and $\{c, d\}$ that appear together with b (to form the $\{b\}$-projected database). The corresponding FP-tree contains $\{c\}$:2 meaning that c occurs twice with b (i.e., edge-pair $\{b, c\}$ is frequent with frequency 2). To recap, a total of 1 collection of frequent edges was found from the $\{b\}$-projected database: $\{b, c\}$.

Similar steps are applied to other frequent singletons $\{c\}$, $\{d\}$ and $\{f\}$ in order to discover all collections of frequent edges. For instance, a total of 3 collections of frequent edges

were found from the $\{c\}$-projected database: $\{c, d\}$, $\{c, d, f\}$ and $\{c, f\}$. Similarly, a total of 1 collection of frequent edges was found from the $\{d\}$-projected database: $\{d, f\}$. Consequently, our first algorithm found a total of 5+7+1+3+1 = 17 collections of frequent edges, which include some connected edges like $\{a, d\} \equiv \{(v_1, v_2), (v_2, v_3)\}$ as well as some disjoint edges such as $\{a, f\} \equiv \{(v_1, v_2), (v_3, v_4)\}$. □

3.2 Frequency Counting on a Single FP-tree

In Example 2, the mining process requires multiple FP-trees to be kept in the memory during the mining process. However, when the memory space is limited, *not* all of the multiple FP-trees can fit into the memory. One way to solve this problem is to apply an effective *frequency counting technique*: Once an FP-tree for the projected database of a frequent singleton is built, the algorithm traverses every tree node in a depth-first manner (e.g., pre-order, in-order, or post-order traversal). For every first visit of a tree node, the algorithm generates the collection of edges represented by the node and its subsets. We also compute their frequencies.

Example 3. Revisit Example 2 but with the frequency counting techniques applied to a single FP-tree. Specifically, our second algorithm first constructs an FP-tree for the $\{a\}$-projected database. It then traverses every node in such an FP-tree. When traversing the leftmost branch $\langle c{:}4,\ b{:}1\rangle$, we visit nodes "c:4" (which represents edge-pair $\{a, c\}$ with frequency 4) and "b:1" (which gives $\{a, b\}$ with frequency 1 and $\{a, b, c\}$ with frequency 1). Next, we traverse the middle branch $\langle c{:}4,\ f{:}3,\ d{:}2\rangle$. By visiting nodes "f:3" and "d:2", we get $\{a, f\}$ and $\{a, c, f\}$ both with frequencies 3, as well as $\{a, d\}, \{a, c, d\}, \{a, d, f\}$ and $\{a, c, d, f\}$ all with frequencies 2. Finally, we visit nodes "f:1" and "d:1" in the rightmost branch $\langle f{:}1,\ d{:}1\rangle$, from which we get the frequency 1 for both $\{a, d\}, \{a, d, f\}$ and $\{a, f\}$. This frequency value is added to the existing frequency count of 2 (from the middle branch) to give the frequency of $\{a, d\}$ and $\{a, d, f\}$ equal to 3. Hence, with the *minsup* threshold set to 2, we obtain frequent patterns $\{a, c\}{:}4,\ \{a, c, d\}{:}2,\ \{a, c, d, f\}{:}2,\ \{a, c, f\}{:}3,\ \{a, d\}{:}3,\ \{a, d, f\}{:}3$ and $\{a, f\}{:}4$. Note that, during this mining process for the $\{a\}$-projected database, we count frequencies of subgraphs without recursive construction of FP-trees.

Afterwards, we build an FP-tree for the $\{b\}$-projected database and count frequencies of all frequent subgraphs containing item b. Similar steps are applied to the FP-trees for the $\{c\}$- and $\{d\}$-projected databases. Consequently, our second algorithm found the same 17 collections of frequent edges as those in Example 2. However, at any moment during the mining process, only one FP-tree needs to be constructed and kept in the memory (cf. multiple FP-trees required by our first algorithm described in Section 3.1). □

3.3 Mining a Single FP-tree in a Top-Down Fashion

An alternative way to avoid the construction of multiple FP-trees is to apply top-down tree mining (similar to that of the TD-FP-growth algorithm [32]). Specifically, we (i) form only a projected database for each frequent singleton (cf. Section 3.1, in which projected databases for singletons and non-singletons are recursively formed) and (ii) in reverse order—i.e., the top-down order (cf. bottom-up fashion as in the FP-growth algorithm or that described in Section 3.1).

Example 4. When applying this top-down tree-based mining, our third algorithm found the same 17 collections of frequent edges as those in Examples 2 and 3. □

3.4 Vertical Mining

With the representation of relevant graph structured data in the DSMatrix, it is logical to mine frequent subgraphs *vertically*. Specifically, our fourth algorithm examines each row (representing an edge). The *row sum* (i.e., total number of 1s) gives the frequency of the edge represented by that row. Once the frequent singleton edges are found, we *intersect the bit vectors* for two edges. If the row sum of the resulting intersection $\geq$ the user-specified *minsup* threshold, then we find a frequent edge-pair. We repeat these steps by intersecting two bit vectors of frequent patterns to find frequent subgraphs consisting of multiple edges.

Example 5. Revisit Examples 2, 3 and 4. Our fourth algorithm first computes the row sum for each row (i.e., for each domain item). As a result, we find that edges a, b, c, d and f are all frequent with frequencies 5, 2, 5, 4 and 4, respectively. Afterwards, the algorithm intersects the bit vector of a (i.e., Row a) with any one of the remaining four bit vectors (i.e., any one of the four rows) to find frequent edge-pairs $\{a, c\}, \{a, d\}$ and $\{a, f\}$ with frequencies 4, 3 and 4, respectively, because (i) the intersection of $\vec{a}$ and $\vec{c}$ gives a bit vector 101110, (ii) the intersection of $\vec{a}$ and $\vec{d}$ gives a bit vector 110010, and (iii) the intersection of $\vec{a}$ and $\vec{f}$ gives a bit vector 110110. Next, we intersect (i) $\overrightarrow{ac}$ with $\overrightarrow{ad}$, (ii) $\overrightarrow{ac}$ with $\overrightarrow{af}$ and (iii) $\overrightarrow{ad}$ with $\overrightarrow{af}$ to find frequent edge-triplets $\{a, c, d\}, \{a, c, f\}$ and $\{a, d, f\}$. We also intersect $\overrightarrow{acd}$ with $\overrightarrow{acf}$ to find frequent edge-quadruplet $\{a, c, d, f\}$. These are all collections of frequent edges containing a.

Afterwards, we repeat similar steps with the bit vectors for other edges. For instance, we intersect $\vec{b}$ with $\vec{c}, \vec{d}$ and $\vec{f}$. We find out that, among them, only $\{b, c\}$ is frequent with frequency 2. We also intersect $\vec{c}$ with $\vec{d}$ and $\vec{f}$ to find frequent edge-triplets $\{c, d\}$ and $\{c, f\}$, each with frequencies of 3. We also find frequent edge-quadruplet $\{c, d, f\}$ by intersecting $\overrightarrow{cd}$ and $\overrightarrow{cf}$. Finally, we intersect $\vec{d}$ and $\vec{f}$ to find frequent edge-pair $\{d, f\}$ with frequency 3. Consequently, our fourth algorithm found the same 17 collections of frequent edges as those in Examples 2, 3 and 4. □

3.5 Post-Processing Step

So far, we have described how our four algorithms find collections of all frequent edges, which include *connected* edges such as $\{a, d\} \equiv \{(v_1, v_2), (v_2, v_3)\}$ as well as *disjoint* edges such as $\{a, f\} \equiv \{(v_1, v_2), (v_3, v_4)\}$. To filter out disjoint edges, we apply the following post-processing step to check every frequent edges. We look up the vertex information of each edge such as (v_1, v_2) for edge a. See Table 1. Let X represent a collection of multiple frequent edges. Then, for each edge $e \equiv (v_i, v_j) \in X$ (where $|X| \geq 2$), count the frequency (or occurrence) of v_i and v_j in X. If frequency of v_i (or v_j) is at least 2 in X, then v_i (or v_j) is a vertex connecting at least 2 edges (i.e., these 2 edges are connected):

- $\forall e \equiv (v_i, v_j) \in X$, [*frequency*$(v_i) \geq 2$ or *frequency*$(v_j) \geq 2$] $\Rightarrow$ X is a *connected* subgraph.

Otherwise—i.e., there exists an edge $e' \equiv (v'_i, v'_j)$—such that frequency of v'_i and that of v'_j are both less than 2 in X, such an edge e' is disjoint (i.e., an isolated edge):

Table 1: Table capturing vertices of each edge

Edge	Vertices
a	(v_1, v_2)
b	(v_1, v_3)
c	(v_1, v_4)
d	(v_2, v_3)
e	(v_2, v_4)
f	(v_3, v_4)

- $\exists e' \equiv (v'_i, v'_j) \in X'$, [*frequency*$(v'_i) < 2$ and *frequency*$(v'_j) < 2$] $\Rightarrow X'$ is *not* a connected subgraph.

For instance, we check and keep $\{a, d\}$ because it is a pair of *connected* edges; we check and prune away $\{a, f\}$ because it is a pair of *disjoint* edges.

Example 6. Continue with Examples 2, 3, 4, or 5. Before the post-processing step, each algorithm finds a total of 17 collections of frequent edges from the streaming graph structured data. Among them, let us consider $\{a, c\} \equiv \{(v_1, v_2), (v_1, v_4)\} = X$. (i) For (v_1, v_2), *frequency*$(v_1) = 2$ (and *frequency*$(v_2) = 1$); (ii) for (v_1, v_4), *frequency*$(v_1) = 2$ (and *frequency*$(v_4) = 1$). So, for each edge in X, it satisfies the condition that [*frequency*$(v_i) \geq 2$ or *frequency*$(v_j) \geq 2$]. Hence, X is a connected subgraph.

In contrast, consider $\{a, f\} \equiv \{(v_1, v_2), (v_3, v_4)\} = X'$. For (v_1, v_2), *frequency*$(v_1) = 1$ and *frequency*$(v_2) = 1$. Hence, there exists an edge $(v_1, v_2) \in X'$ such that [*frequency*$(v_1) < 2$ and *frequency*$(v_2) < 2$]. Hence, X' is *not* a connected subgraph.

Similarly, consider $\{c, d\} \equiv \{(v_1, v_4), (v_2, v_3)\} = X''$. For (v_1, v_4), *frequency*$(v_1) = 1$ and *frequency*$(v_4) = 1$. Hence, there exists an edge $(v_1, v_4) \in X''$ such that [*frequency*$(v_1) < 2$ and *frequency*$(v_4) < 2$]. Hence, X'' is *not* a connected subgraph.

Applying a similar post-processing step to check all 17 collections of frequent edges, we find that $\{a, f\} \equiv \{(v_1, v_2), (v_3, v_4)\}$ (consisting of two disjoint edges $a \equiv (v_1, v_2)$ and $f \equiv (v_3, v_4)$) and $\{c, d\} \equiv \{(v_1, v_4), (v_2, v_3)\}$ (consisting of two disjoint edges $c \equiv (v_1, v_4)$ and $d \equiv (v_2, v_3)$) are both not connected subgraphs, and thus can be pruned. Consequently, only 15 frequent *connected* subgraphs are then returned to the user. □

4. DIRECT FREQUENT CONNECTED SUBGRAPH MINING

So far, we have described how to mine frequent connected subgraphs by finding all collections of frequent edges and then pruning collections of disjoint edges in a post-processing step. When the number of vertices increases, chances of having disjoint edges also increase. Consequently, a lot of time and effort may have been spent on mining all collections of frequent edges including many disjoint edges, which are then pruned. To deal with this issue, we propose an alternative algorithm that mines frequent connected subgraphs directly.

Specifically, our fifth algorithm *directly* mines frequent connected subgraphs vertically. First, to mine frequent singletons, we examine each row (representing an edge). The *row sum* (i.e., total number of 1s) gives the frequency of the edge represented by that row. If the row sum $\geq$ the user-specified *minsup* threshold, then we find a frequent edge.

Table 2: Table capturing neighbors of each edge

Edge	Neighboring edges
a	b, c, d, e
b	a, c, d, f
c	a, b, e, f
d	a, b, e, f
e	a, c, d, f
f	b, c, e, d

Once the frequent singleton edges are found, we *intersect the bit vectors* for two *connected* edges based on the neighborhood information. See Table 2. If the row sum of the resulting intersection $\geq$ the user-specified *minsup* threshold, then we find a frequent *connected* subgraph consisting of 2 edges. We repeat these steps by intersecting two bit vectors of frequent connected subgraphs to find frequent connected subgraph of multiple edges.

During the mining process, the neighborhood information for frequent edge can be looked up from Table 2. The neighborhood information for a frequent connected pair $\{x, y\}$ can be computed by the following:

$$\begin{aligned} & neighbor(\{x, y\}) \\ = \; & neighbor(\{x\}) \cup neighbor(\{y\}) - \{x, y\}, \end{aligned} \quad (1)$$

where $y \in neighbor(\{x\})$. Similarly, the neighborhood information for a frequent connected subgraph $X \cup \{y\}$ consisting of k edges can be computed by the following:

$$\begin{aligned} & neighbor(X \cup \{y\}) \\ = \; & neighbor(X) \cup neighbor(\{y\}) - X - \{y\}, \end{aligned} \quad (2)$$

where (i) $y \in neighbor(X)$ and (ii) $|X| = k - 1$.

Example 7. Revisit Example 6. Our direct algorithm first computes the row sum for each row (i.e., for each edge). As a result, we find that edges a, b, c, d and f are all frequent with frequencies 5, 2, 5, 4 and 4, respectively. Afterwards, we intersect the bit vector of a (i.e., Row a) with bit vectors of any of its neighbor $neighbor(\{a\}) = \{b, c, d, e\}$ to find the following:

- connected subgraph $\{a, b\}$ consisting of 2 edges a & b and with frequency 1 and thus infrequent;
- connected subgraph $\{a, c\}$ consisting of 2 edges a & c and with frequency 4 and thus frequent; as well as
- connected subgraph $\{a, d\}$ consisting of 2 edges a & d and with frequency 3 and thus frequent.

Note that, as the algorithm only intersects vectors of frequent edges, it does not intersect with infrequent edge e even though $e \in neighbor(\{a\})$. Moreover, when compared with Example 5, our direct algorithm does not produce $\{a, f\}$. Although single edge f is frequent, it is not in the neighborhood of $\{a\}$ and thus not connected with a.

Next, we intersect (i) $\overrightarrow{ac}$ with $\overrightarrow{d}$ to find frequent connected edge-triplet $\{a, c, d\}$ because $d \in neighbor(\{a, c\})$, which can be computed as $neighbor(\{a\}) \cup neighbor(\{c\}) - \{a, c\} = \{b, d, e, f\}$. Then, we intersect (i) $\overrightarrow{acd}$ with $\overrightarrow{f}$ to get connected edge-quadruplet $\{a, c, d, f\}$ because $f \in neighbor(\{a, c, d\})$, which is computed as $neighbor(\{a, c\}) \cup neighbor(\{d\}) - \{a, c, d\} = \{b, e, f\}$. Similarly, we intersect (i) $\overrightarrow{ac}$ with $\overrightarrow{f}$ to find frequent connected edge-triplet

$\{a, c, f\}$ as $f \in neighbor(\{a, c\})$. We also intersect (i) $\overrightarrow{ad}$ with $\vec{f}$ to get frequent connected edge-triplet $\{a, d, f\}$ because $neighbor(\{a, d\}) = neighbor(\{a\}) \cup neighbor(\{d\}) - \{a, d\} = \{b, c, e, f\}$ contains f. These are all collections of frequent *connected* edges containing a. In the above procedure, we only extend on connected subgraphs.

Afterwards, we repeat similar steps with the bit vectors for other edges. For instance, we intersect $\vec{b}$ with $\vec{c}, \vec{d}$ and $\vec{f}$. We find out that, among them, only $\{b, c\}$ is frequent with frequency 2. We also intersect $\vec{c}$ with $\vec{f}$ to find frequent connected edge-pair $\{c, f\}$ with frequency 3. Note that we do not intersect $\vec{c}$ with $\vec{d}$ because $d \notin neighbor(\{c\}) = \{a, b, e, f\}$. However, we find frequent edge-triplet $\{c, d, f\}$ by intersecting $\overrightarrow{cf}$ and $\vec{d}$ because $d \in neighbor(\{c, f\}) = neighbor(\{c\}) \cup neighbor(\{f\}) - \{c, f\} = \{a, b, d, e\}$. Finally, we intersect $\vec{d}$ and $\vec{f}$ to find frequent edge-pair $\{d, f\}$ having frequency 3. □

5. EXPERIMENTAL EVALUATION

To acquire streams of linked graph structured data, we first generated random graph models via a Java-based generator by varying model parameters (e.g., topology, average fan-out of nodes, edge centrality, etc.). We then generated graph streams as nodes and node-edge relationships derived from the above graph models, and obtained node values from popular data stream sets available in literature (stored in the projected database). In addition, we also used many different databases including IBM synthetic data, real-life databases (e.g., connect4) from the UC Irvine Machine Learning Depository as well as those from the Frequent Itemset Mining Implementation (FIMI) Dataset Repository. For example, connect4 is a dense data set containing 67,557 records with an average transaction length of 43 items, and a domain of 130 items. Each record represents a graph of legal 8-ply positions in the game of connect 4. All experiments were run in a time-sharing environment in a 1 GHz machine. We set each batch to be 6K records and the window size w=5 batches. The reported figures are based on the average of multiple runs. Runtime includes CPU and I/Os; it includes the time for both tree construction and frequent pattern mining steps.

In the first experiment, we measured the accuracy of mining with the following structures: (i) DSTree [26], (ii) DSTable [8], and (iii) DSMatrix. Experimental results show that the four mining algorithms that use the DSMatrix with the post-processing steps (Section 3) gave the same mining results as the direct algorithm (Section 4) that uses the DSMatrix without the post-processing step. Experimental results also show that these five algorithms (which all use the DSMatrix) gave the same mining results as any algorithms that conduct mining with the DSTree or DSTable.

In the second experiment, we measured the space efficiency. Experimental results show that mining with the DSTree stored one global DSTree and multiple local FP-trees in main memory, and thus took the largest main memory space. Mining with the DSTable and DSMatrix required less memory because the DSTable and DSMatrix were kept on disk. Among those algorithms that mine with the DSMatrix, the first algorithm (i.e., the one mines with multiple FP-trees and described in Section 3.1) required the largest amount of memory space because it keeps at most k FP-trees in the memory during the entire mining process, where k is the maximum number of edges in any collection of frequent edges. The algorithms that mine with a single FP-tree (Sections 3.2 and 3.3) required less space because they keep at most one FP-tree in the memory during the entire mining process. The two vertical mining algorithms (Sections 3.4 and 4) required the least amount of memory space because they both work with bit vectors.

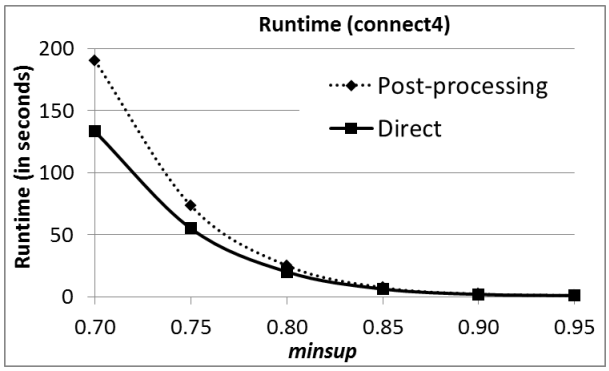


Figure 2: Experimental results on vertical mining.

In the third experiment, we measured the time efficiency. Among those algorithms that mine with the DSMatrix, the first algorithm (i.e., the one mines with multiple FP-trees and described in Section 3.1) required the longest runtime because it recursively constructs FP-trees during the entire mining process. The algorithms that mine with a single FP-tree (Sections 3.2 and 3.3) required shorter runtime because they construct at most one FP-tree for each frequent edge (i.e., for a total of at most m FP-trees, one for each of the $|E|$ edges) during the entire mining process. The two vertical mining algorithms (Sections 3.4 and 4) required the shortest runtime because they both work with bitwise and set intersection operators. Between these two vertical mining algorithms, as expected, the one with the post-processing step required longer runtime than the direct algorithm because the latter mines frequent connected subgraphs directly. Figure 2 shows the runtimes of our fourth algorithm (i.e., vertical mining with *post-processing* step) and our fifth algorithm (i.e., *direct* vertical mining).

We also performed some additional experiments (e.g., evaluating the effect of *minsup*). Results show that the runtime decreased when *minsup* increased. In another experiment, we tested scalability with the number of batches in the stream of graph structured data. The results show that the scalability of our (five) algorithms, especially the two vertical mining algorithms.

As future work, we plan to conduct more extensive experiments on various datasets (including Big data) with different parameter settings (e.g., varying *minsup*, the number of vertices and edges in graph structured data and/or linked data).

6. CONCLUSIONS

Motivated by the demand of having algorithms that use *limited memory* space for discovering collections of frequently co-occurring *connected* edges from big, interlinked, dynamic graph structures of linked data, we proposed five algorithms for frequent subgraph mining. All our algorithms use a DSMatrix to capture important contents of streams of graph structured linked data. The DSMatrix is updated when the window slides. The discovery of frequent connected sub-

graphs is "delayed" until the mining is needed. Three of our algorithms use horizontal tree-based mining approaches: (i) The first algorithm builds multiple FP-trees recursively in a bottom-up fashion; (ii) the second algorithm builds FP-trees in a bottom-up fashion, but builds only a single FP-tree for each singleton; and (iii) the third algorithm also builds only a single FP-tree for each singleton, but builds in a top-down fashion. The fourth algorithm uses a vertical bitwise mining approach. Note that all these four algorithms mine collections of all frequent (connected or disjoint) edges, and prune those disjoint edges at a *post-processing* step. In contrast, our fifth algorithm also uses a vertical bitwise mining approach, but *directly* mines collections of all *connected* edges. Experimental results show the space and time efficiency of vertical frequent subgraph mining from streams of linked graph structured data.

7. ACKNOWLEDGEMENTS

This project is partially supported by NSERC (Canada) and University of Manitoba.

8. REFERENCES

[1] C.C. Aggarwal, Y. Li, P.S. Yu, & R. Jin. On dense pattern mining in graph streams. *PVLDB*, **3**(1–2), pp. 975–984 (2010)

[2] R. Agrawal & R. Srikant. Fast algorithms for mining association rules. In *Proc. VLDB 1994*, pp 487–499.

[3] D. Bianchini, S. Castano, V. de Antonellis, A. Ferrara, E. Quintarelli, & L. Tanca. RUBIK: proactive, entity-centric and personalized situational web application design. *TLDKS*, **13**, pp. 123–157 (2014)

[4] A. Bifet, G. Holmes, B. Pfahringer, & R. Gavaldà. Mining frequent closed graphs on evolving data streams. In *Proc. ACM KDD 2011*, pp. 591–599.

[5] P. Braun, J.J. Cameron, A. Cuzzocrea, F. Jiang, & C.K. Leung. Effectively and efficiently mining frequent patterns from dense graph streams on disk. *Procedia Computer Science*, **35**, pp. 338–347 (2014)

[6] G. Buehrer, S. Parthasarathy, & A. Ghoting. Out-of-core frequent pattern mining on a commodity. In *Proc. ACM KDD 2006*, pp. 86–95.

[7] J.J. Cameron, A. Cuzzocrea, F. Jiang, & C.K. Leung. Frequent pattern mining from dense graph streams. In *Proc. EDBT/ICDT 2014 Workshops*, pp. 240–247.

[8] J.J. Cameron, A. Cuzzocrea, & C.K. Leung. Stream mining of frequent sets with limited memory. In *Proc. ACM SAC 2013*, pp. 173–175.

[9] L. Chi, B. Li, & X. Zhu. Fast graph stream classification using discriminative clique hashing. In *Proc. PAKDD 2013, Part I*, pp. 225–236.

[10] A. Cuzzocrea. CAMS: OLAPing multidimensional data streams efficiently. In *Proc. DaWaK 2009*, pp. 48–62.

[11] A. Cuzzocrea & S. Chakravarthy. Event-based lossy compression for effective and efficient OLAP over data streams. *DKE*, **69**(7), pp. 678–708 (2010)

[12] A. Cuzzocrea, F. Furfaro, G.M. Mazzeo & D. Saccà. A grid framework for approximate aggregate query answering on summarized sensor network readings. In *Proc. OTM Workshops 2004*, pp. 144–153.

[13] A. Cuzzocrea, C.K. Leung, & S.K. Tanbeer. Mining of diverse social entities from linked data. In *Proc. EDBT/ICDT 2014 Workshops*, pp. 269-274.

[14] R. de Virgilio & D. Bianchini. SeeVa: a model based framework for semantic web service discovery. *TLDKS*, **14**, pp. 51–82 (2014)

[15] A. Ferrara, L. Genta, & S. Montanelli. Linked data classification: a feature-based approach. In *Proc. EDBT/ICDT 2013 Workshops*, pp. 75–82.

[16] C. Giannella, J. Han, J. Pei, X. Yan, & P.S. Yu. Mining frequent patterns in data streams at multiple time granularities. In *Data Mining: Next Generation Challenges and Future Directions*, ch. 6 (2004)

[17] G. Grahne & J. Zhu. Mining frequent itemsets from secondary memory. In *Proc. IEEE ICDM 2004*, pp. 91–98.

[18] J. Han, J. Pei, & Y. Yin. Mining frequent patterns without candidate generation. In *Proc. ACM SIGMOD 2000*, pp. 1–12.

[19] T. Heath & C. Bizer *Linked data: evolving the web into a global data space*. Synthesis lectures on the semantic web: theory and technology, Morgan & Claypool, 2011.

[20] R. Jin & G. Agrawal. An algorithm for in-core frequent itemset mining on streaming data. In *Proc. IEEE ICDM 2005*, pp. 210–217.

[21] F. Jiang & C.K. Leung. A business intelligence solution for frequent pattern mining on social networks. In *Proc. IIEEE ICDM Workshops 2014*, pp. 789–796.

[22] F. Jiang, C.K. Leung, D. Liu, & A.M. Peddle. Discovery of really popular friends from social networks. In *Proc. IEEE BDCloud 2014*, pp. 342–349.

[23] W. Lee, C.K. Leung, & J.J. Song. Reducing noises for recall-oriented patent retrieval. In *Proc. IEEE BDCloud 2014*, pp. 579–586.

[24] C.K. Leung, A. Cuzzocrea, & F. Jiang. Discovering frequent patterns from uncertain data streams with time-fading and landmark models. *LNCS TLDKS*, **8**, pp. 174–196 (2013)

[25] C.K. Leung & F. Jiang. A data science solution for mining interesting patterns from uncertain big data. In *Proc. IEEE BDCloud 2014*, pp. 235–242.

[26] C.K. Leung & Q.I. Khan. DSTree: a tree structure for the mining of frequent sets from data streams. In *Proc. IEEE ICDM 2006*, pp. 928–932.

[27] C.K. Leung, R.K. MacKinnon, & S.K. Tanbeer. Fast algorithms for frequent itemset mining from uncertain data. In *Proc. IEEE ICDM 2014*, pp. 893–898.

[28] R.K. MacKinnon, T.D. Strauss, & C.K. Leung. DISC: efficient uncertain frequent pattern mining with tightened upper bounds. In *Proc. IIEEE ICDM Workshops 2014*, pp. 1038–1045.

[29] O. Papapetrou, M. Garofalakis, & A. Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *PVLDB*, **5**(10), pp. 992–1003 (2012)

[30] S. Tirthapura & D.P. Woodruff. A general method for estimating correlated aggregates over a data stream. In *Proc. IEEE ICDE 2012*, pp. 162–173.

[31] E. Valari, M. Kontaki, & A.N. Papadopoulos. Discovery of top-k dense subgraphs in dynamic graph collections. In *Proc. SSDBM 2012*, pp. 213–230.

[32] K. Wang, L. Tang, J. Han, & J. Liu. Top down FP-growth for association rule mining. In *Proc. PKDD 2002*, pp. 334–340.

Privacy and Anonymity in the Information Society (PAIS)

Traian Marius Truta, (Northern Kentucky University),
Li Xiong, Emory University),
Farshad Fotouhi (Wayne State University)

Transparency and Disclosure Risk in Data Privacy

Vicenç Torra
University of Skövde, Sweden

ABSTRACT

k-Anonymity and differential privacy can be considered examples of Boolean definitions of disclosure risk. In contrast, record linkage and uniqueness are examples of quantitative measures of risk. Record linkage is a powerful approach because it can model different types of scenarios in which an adversary attacks a protected database with some information and background knowledge.

Transparency holds in data privacy when data is published together with details on their processing. This includes the data protection method used and its parameters. Intruders can use this information to improve their attacks. Specific record linkage algorithms can be defined to take into account this information, and to define more accurate disclosure risk measures.

Machine learning and optimization techniques also permits us to increase the effectiveness of record linkage algorithms.

This talk will be focused on disclosure risk measures based on record linkage. We will describe how we can improve the performance of the algorithms under the transparency principle, as well as using machine learning and optimization techniques.

Short Bio

Vicenç Torra is a professor in the School of Informatics at the U. of Skövde in Sweden. Until 2014 he was Associate Prof. - Research Track at the Artificial Intelligence Research Institute of the Spanish National Research Council (IIIA-CSIC). His fields of interest are data privacy, information fusion and approximate reasoning.

He is ECCAI Fellow (2010), Elected Member of ISI (2013). He has published over 200 publications and 4 books. One undergraduate course on artificial intelligence (in Catalan and Spanish), one graduate text (Modeling decisions, Springer, 2007; with Y. Narukawa), a book on the history of computer science (From the Abacus to the digital revolution, RBA, 2010) published in Spanish, Portuguese, Italian, French, English, Polish, Russian, and another one on decisions and elections (The mathematics of elections, RBA, in press).

He founded and is the editor in chief of the journal Transactions on Data Privacy (http://www.tdp.cat/). He is associate editor of Information Sciences (Elsevier) and member of the editorial board of Fuzzy Sets and Systems (2004-), Progress in Artificial Intelligence (2011-), J. of Advanced Computational Intelligence and Intel. Informatics (2007-), Int. J. of Computational Intelligence System (2008-). He founded the annual MDAI conference series in 2004 and is PC co-chair ever since. His research has been funded by national and international agencies.

Privacy-Integrated Graph Clustering Through Differential Privacy

Yvonne Mülle *
University of Zurich, Switzerland
muelle@ifi.uzh.ch

Chris Clifton
Purdue University, USA
clifton@cs.purdue.edu

Klemens Böhm
Karlsruhe Institute of Technology (KIT), Germany
klemens.boehm@kit.edu

ABSTRACT

Data mining tasks like graph clustering can automatically process a large amount of data and retrieve valuable information. However, publishing such graph clustering results also involves privacy risks. In particular, linking the result with available background knowledge can disclose private information of the data set. The strong privacy guarantees of the differential privacy model allow coping with the arbitrarily large background knowledge of a potential adversary. As current definitions of neighboring graphs do not fulfill the needs of graph clustering results, this paper proposes a new one. Furthermore, this paper proposes a graph clustering approach that guarantees 1-edge-differential privacy for its results. Besides giving strong privacy guarantees, our approach is able to calculate usable results. Those guarantees are ensured by perturbing the input graph. We have thoroughly evaluated our approach on synthetic data as well as on real-world graphs.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering; H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms, Security

Keywords

Clustering, Graph Mining, Differential Privacy

1. INTRODUCTION

For many types of data, the proper representation is a graph structure. There exist many approaches which automatically retrieve valuable information from graphs. One such approach is *graph clustering*. Clustering groups similar objects together and assigns dissimilar objects to different groups. In the context of social networks, graph clustering helps to better understand the structure of these networks.

Being able to collect and process such graph data does not only have its opportunities, but also involves privacy risks. The *Gaydar* project [1] of two MIT students illustrates such a risk: by linking the *Facebook* friends of a person with the knowledge of the gender and sexuality of those friends, it has been possible to predict if the person is gay. Let us now assume that an operator of a social network wants to publish information on the community structure (i.e., cluster structure) of the network. The operator decides to publish the result of a graph clustering algorithm, i.e., the community structure of the network, without the actual graph structure. Such a clustering result can still put the users' privacy at risk. Dependent on the background knowledge, an outsider can retrieve actual connections between users from the community structure. In the worst case, it might even be possible to not only identify certain connections but to reconstruct the entire social network. In consequence, it is necessary to publish the community structure and thus the clustering result in a privacy preserving manner. As differential privacy gives strong privacy guarantees, it is desirable to publish a differentially private graph clustering result. For differentially private graph clustering, no approach exists so far. The paper focuses on proposing such an approach.

In order to apply differential privacy to graph clustering, the following challenges have to be solved. 1. It needs to be determined which parts of the graph information that the clustering result exposes are relevant to privacy: a node, that node's edges, or any edges in the graph. 2. A noise-adding mechanism must be developed to protect this information. As a consequence, only the graph parts to be protected should be perturbed. Choosing the appropriate perturbation is challenging as, for instance, minor changes in the edge structure of the graph can significantly alter the clustering result. 3. The ever present trade-off between privacy and usability must be solved so that both are preserved at a reasonable level. This also includes determining which properties a graph clustering algorithm and graphs must have in order to still produce a usable clustering result.

In this paper, we propose solutions to these challenges. An important category of graph clustering approaches calculates their results only based on the edge structure. The nodes are used to represent a cluster. Thus changes in the node set are directly visible in the clustering result. Therefore, we propose m-edge-differential privacy. It relies on a new neighboring graphs definition that guarantees privacy for the edges of a node, but does not require changing the node set. We also investigate if it is necessary to protect all edges of a node or if it is sufficient to only protect some

*Work originated while author was at Purdue University.

of them. Additionally, we propose $\mathcal{PIG}$, an approach that combines the perturbation of the input graph with existing graph clustering approaches. $\mathcal{PIG}$ guarantees 1-edge-differential privacy, a notion that can be generalized to m-edge-differential privacy. The amount of noise created by the perturbation is configurable via a parameter that represents the trade-off between privacy and usability. We develop recommendations on how to set this parameter to achieve specific privacy goals.

2. RELATED WORK

Privacy for data mining tasks can be achieved by releasing sanitized data sets or by developing private data mining algorithms. The approaches not only differ in how data mining is performed, but also in the underlying privacy model.

Privacy-preserving clustering approaches perform clustering on tabular data where the data is distributed among several parties. They are based on the concept of secure multi-party computation. The goal is that each party only learns the final clustering result, but no intermediate values. Privacy preserving k-means clustering [26, 11, 10] and privacy-preserving DBSCAN [15] have been studied. All these approaches have in common that they perform clustering on tabular data and therefore are not able to deal with many instances of graph data. Furthermore, unlike differential privacy they provide no protection against the result inherently revealing individual information. A clustering result contains new, so far probably unknown correlations between entries or nodes and thus has the potential to reveal sensitive information.

Differentially private data analysis is the task of publishing a graph in a differentially private manner. [19] uses a Kronecker graph model in order to publish a differentially private maximum likelihood estimator for graphs. [24] proposes an approach that uses the degree correlations of the original graph to generate a differentially private dK-graph model. Thus, it aims to preserve as much structure of the unmodified graph as possible. Both approaches publish the synthetic graph for a general purpose. However, as data mining tasks often differ in what data they need, the accuracy of the result is expected to increase when the data is sanitized for a specific task.

Differentially private data mining approaches exist for tabular data as well as for graph data. Differentially private data mining techniques have been proposed for frequent itemsets [2, 17] and pattern mining [9, 25], but also for clustering and graph analysis. The challenge how to perform clustering in a differentially private manner has been addressed by [6], [22], and [7]. All approaches focus on k-median and k-means queries on tabular data. [6] realizes the task by publishing private coresets that are representative subsets of a database which preserve some geometric properties. [22] releases private clustering results by perturbing the coordinates of the center points that represent a cluster. [7] proposes an approach for differentially private k-median clustering. The approach uses the exponential mechanism to swap center points with non-center points.

The following approaches release individual graph properties in an edge-differentially private manner. Graph properties like the degree distribution [8], frequent graph patterns [25], counting queries for k-triangles and k-stars [12], and clustering coefficients [27] have been considered. Guaranteeing node-differential privacy has been taken into account for the number of edges in a graph, counting queries like triangles, k-cycles, k-stars and certain estimators for power law graphs [13].

Differential privacy was successfully applied to all those data mining tasks. However, differentially private graph clustering has not yet been addressed. Thus, we focus on how graph clustering results can be released in a differentially private manner.

3. FOUNDATIONS

3.1 Graph Clustering

Clustering groups similar objects in so-called *clusters*. In contrast to clustering on tabular data, graph clustering also – or even exclusively – considers the structural data given by the edges of the graph. The following four criteria specify a graph clustering approach: the graph type, the cluster definition, the clustering realization and the representation of the graph clustering result.

The *graph type* contains the information on what sort of graph the clustering approach can be performed. Definition 1 formalizes the term *graph* as used in this paper.

Definition 1. Graph $G = (V, E)$
A graph $G = (V, E)$ consists of a set of nodes V and a set of edges $E \subseteq V \times V$. The graph is non-attributed and unweighted. The edges between nodes are undirected.

A *cluster definition* contains the properties that a set of nodes must fulfill in order to form a cluster. No universally accepted graph cluster definition exists. We consider cluster definitions that group nodes together based on their connectivity and other structural properties. Such properties are reachability, inter-cluster connectivity, and neighborhood.

The *realization of the cluster definition* consists of two steps: how many clusters a node can belong to and the concrete procedure of calculating the clustering result based on the given cluster definition. We do not restrict the realization of the cluster definition.

A *graph clustering result* $Res = \{C_1, \ldots, C_j\}$ is a set of clusters, each of which consists of a set of node IDs. This general solution is mostly used in state-of-the-art graph clustering algorithms. Furthermore, it contains all essential information and does not reveal additional structural information that could increase the privacy risk of a clustering result.

To sum up, this paper focuses on graph clustering approaches that fulfill the following requirements: (R1) They require an undirected, unweighted and non-attributed graph as input. (R2) Their cluster definition is based on connectivity and other structural properties of the graph. (R3) Their representation of the clustering result consists of node IDs.

3.2 Differential Privacy

Differential privacy [3] is a privacy model that gives strong privacy guarantees. It assumes a powerful adversary. The adversary has a broad, nearly unlimited background knowledge. He even is aware of all entries – except for a single one – of the data set. Despite this strong adversary, differential privacy protects against determining if the unknown individual is even in the data. This is the case as query results on the data set remain indistinguishable, independent of whether a single database entry has participated in the result calculation or not.

An algorithm $\mathcal{A}$ is differentially private if the following holds true: For any possible result, it cannot be determined beyond a specific certainty if it was calculated on a graph G_1 or on its neighboring graph G_2.

Definition 2. ϵ-Differential Privacy
A graph clustering algorithm $\mathcal{GCA}$ is ϵ-differentially private if for all neighboring graphs G_1 and $G_2 \in \mathcal{N}(G_1)$, and for all subsets $\mathcal{S}$ of the set of all possible outputs $\{Res_1, \ldots, Res_i\}$,

$$\Pr[\mathcal{GCA}(G_1) \in \mathcal{S}] \leq \exp(\epsilon) \cdot \Pr[\mathcal{GCA}(G_2) \in \mathcal{S}]$$

$\mathcal{N}(G_1)$ is the set of all neighboring graphs of graph G_1.

The term *neighboring graphs* for graph clustering is discussed in Subsection 4.1. In general, *deterministic algorithms* cannot achieve differential privacy (if the result is dependent on the input data set). But they can be converted into *non-deterministic algorithms* by adding non-deterministic noise. Noise-adding mechanisms for graph clustering approaches are discussed in Subsection 4.2.

4. GRAPH CLUSTERING MEETS DIFFERENTIAL PRIVACY

4.1 Neighboring Graphs

[8] has proposed two adaptations of differential privacy for graph data. *k-edge-differential privacy* assumes that the edges contain the sensitive information and thus up to any k edges in the graph should be protected. In contrast, *node-differential privacy* states that a node and all its adjacent edges contain the sensitive information. In the following, we analyze which requirements a neighboring graphs definition for graph clustering must fulfill. As a result we show that the neighboring graphs definition of node-differential privacy does not meet the needs of graph clustering and k-edge-differential privacy covers more neighboring graphs than required. Limiting the set of neighboring graphs to its sufficient set might simplify achieving differentially private graph clustering. Thus, we propose a new *neighboring graphs* definition based on the requirements we identify in the following.

Requirement: Node Set. A graph clustering result is represented by node IDs. Thus, adding or removing a node can be directly visible in the clustering result. Therefore, it is necessary that neighboring graphs consist of the same node set. This is why the neighboring graph definition of node-differential privacy is not applicable.

Requirement: Edge Set. We consider graph clustering approaches that calculate their result based on the edge structure of the graph. Thus, the clustering result provides information about structural similarities in the graph. Concealing private information encoded in the clustering result directly means concealing the existence of certain edges in the graph. When differential privacy is applied to tabular data, a certain row is both used to calculate the clustering result and at the same time should be concealed in the result. Analogously, the edges of a particular node are what determines the clustering – thus it stands to reason that this is the property that should be concealed in this case. This is why the neighboring graph definition of k-edge-differential privacy is too broad and thus not appropriate here.

Requirement: Number of Protected Edges. Is it sufficient to protect some of the edges of a node or is it necessary to protect all of them? Most real-world graphs are power law distributed, i.e., there exist many nodes with few edges and only few nodes with many edges. For instance, in a social network like Facebook, it is more likely that the highly connected nodes represent companies and public figures. Companies and public figures actively decided to reveal more information about their relationships to other participants than a private person would do. As a consequence, it is sufficient to protect the relationships (and thus edges) of private persons with few edges. Our neighboring graphs definition allows protecting m edges of a node. Setting m to a value that covers the number of edges a private person will have, allows protecting (almost) all edges of some nodes (if they have a low degree) and preserve the privacy of some edges for highly connected nodes.

Neighboring Graphs Definition. Definition 3 formalizes the term *neighboring graphs*. Its flexibility allows preserving all edges of a node or only a single edge in the whole graph. How many edges of a node are preserved is dependent on parameter m.

Definition 3. Neighboring Graphs
Let $m \in \mathbb{N}$ be given. Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ (according to Definition 1) are neighboring graphs iff

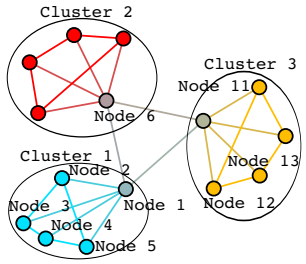
1. $V_2 = V_1$
2. $\exists x \in V_1 : E_2 = (E_1 - E_a(x)) \cup E_b(x)$ with
   $E_a(x) \subseteq \{(u, v) \in E_1 | u = x \vee v = x\}$ and
   $E_b(x) = \{(u, v) \in E_2 | u = x \vee v = x \wedge (u, v) \notin E_1\}$
3. $|E_1 \setminus E_2| + |E_2 \setminus E_1| \leq m$

$\mathcal{N}(G_1)$ is the set of all neighboring graphs of graph G_1.

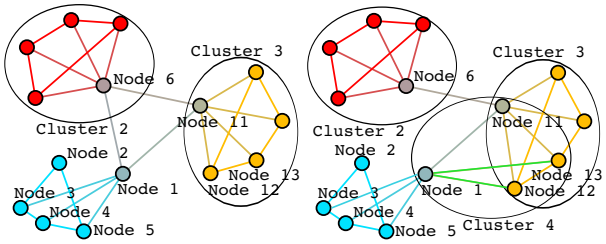
Note that for the case $m = k = 1$, there is no difference between the neighboring graphs definition of k-edge-differential privacy and our definition. For $m = k > 1$ m-edge-differential privacy covers a subset of neighboring graphs compared to k-edge-differential privacy.

Impact of Definition 3 on Clustering Results. Our neighboring graphs definition allows removing edges from and adding edges to a neighboring graph. Thus, comparing the clustering results of two neighboring graphs can result in the following differences: none, only a small number of nodes are clustered differently, or most nodes belong to different clusters, see Example 1.

Example 1. The original graph and two neighboring graphs are shown in Figure 1. Nodes within a circle represent a cluster. A cluster contains at least four nodes which are connected to at least three other nodes in the cluster. In graph G there exist three different clusters. Removing a single edge as in graph G_2 results in the removal of Cluster 1 because its nodes no longer fulfill the cluster definition in G_2. Removing the edge between Node 1 and Node 6 as it is the case for graph G_3 does not have an impact on the clustering result at all because the two nodes are also not clustered in G. However, adding two new edges to G_3 (compared to G) results in a new cluster in the clustering result of graph G_3: Cluster 4.



(a) Original graph G



(b) $G_2 \in \mathcal{N}(G)$ $(m \geq 1)$. (c) $G_3 \in \mathcal{N}(G)$ $(m \geq 4)$.

Figure 1: Illustration of the term *neighboring graphs*.

4.2 Perturbation Mechanisms

We are aware of three different types of perturbation in the literature: sampling, output perturbation and input perturbation. In the following, we discuss if they are suitable for our purposes.

Sampling. Sampling integrates the perturbation into the proceeding of the clustering approach. A possibility would be to only consider a sample of potential edges, which may change the *result set*. For instance, a clique clustering algorithm would have a non-zero probability of creating a cluster out of a set of nodes even though they are not fully connected to each other if it draws a sample of edges that make the nodes fully connected. An advantage of sampling could be that the required amount of noise can be adapted during the execution of the clustering algorithm. At this stage, more information on how to choose the sample could be available. This might reduce the influence of the perturbation on the clustering result. However, sampling has the following two drawbacks: (1) Each of the many existing graph clustering approaches must be separately adapted dependent on the individual cluster definitions and its realizations. (2) Integrating the perturbation into the clustering algorithm makes the analysis of privacy properties complex. In order to prove differential privacy, it is required to analyze the probabilities of clustering results for a specific algorithm. This is particularly challenging as the output space of a graph clustering algorithm is discrete, rewulting in difficulty specifying a probability density function except by (intractable) enumeration. Additionally, finding a closed-form solution for the upper and lower bounds of the probabilities may become very difficult for complex clustering algorithms.

The aforementioned problems can be avoided if the perturbation is a separate step of the algorithm. This allows the perturbation to be analyzed individually.

Output Perturbation. Output perturbation means that the perturbation is performed on the graph clustering result. For instance, the cluster assignment of the nodes can be changed. However, a clustering result only indirectly contains the information that should be protected – the edges of a node. Thus, there exists no intuitive mapping between how the cluster assignment has to be changed and the guarantee of protecting the existence of the edges of a node. The only possibility would be to additionally take the input data into account.

Input Perturbation. Input perturbation adds noise to the input graph. With this method, it is possible to perturb two neighboring graphs in a way that they are indistinguishable with a certain probability for the graph clustering algorithm. As a consequence, the probabilities of a graph clustering result do not need to be calculated; the output of an algorithm on differentially private input is differentially private. In the following, we discuss how the input perturbation must be realized to be appropriate for graph clustering. (1) According to Definition 3, perturbing the graph by changing its number of nodes is not an option. (2) Thus, it is only possible to perturb the graph by adding and removing edges. (2.1) Doing only one of the two (adding or removing edges) is not an option. The reason is that there always exist cases where such a procedure is not possible: If a graph is fully connected, no edges can be added. If a graph does not contain any edges, no edges can be removed. Thus, the clustering result probabilities would also not be affected, and only those results that are calculated for those graphs would be possible. (2.2) Determining the number of edge changes according to the global sensitivity of a clustering approach is not a possibility. The global sensitivity [4] contains the information about the maximum distance between two clustering results of neighboring graphs. There does not exist a correlation between the number of edge changes and the distance between graph clustering results: There is no guarantee that adding or removing l edges will result in a distance increased or decreased by l – yet the distance is what the global sensitivity is based on.

Thus, in order to achieve both differentially private results independent of the algorithm and allow for useful results, we propose an approach that combines edge sampling with edge perturbation in Section 5.

5. $\mathcal{PIG}$

We propose $\mathcal{PIG}$ – **P**rivacy-**I**ntegrated **G**raph clustering approach – as a general approach for guaranteeing 1-edge-differential privacy. It is independent of a concrete graph clustering algorithm as long as the clustering approach fulfills the requirements given in Subsection 3.1. Furthermore, $\mathcal{PIG}$ perturbs the input graph by perturbing the adjacency matrix of the graph dependent on differential privacy's parameter ϵ. As future work, $\mathcal{PIG}$ will be extended to m-edge-differential privacy.

$\mathcal{PIG}$ consists of two steps: the perturbation of the input graph, called $\mathcal{PIG}_{pert}$, and the graph clustering algorithm applied to the perturbed graph. The idea behind $\mathcal{PIG}$ is as follows: If the perturbed versions of neighboring graphs are the same, the graph clustering approach will calculate the same clustering result.

The perturbation step of $\mathcal{PIG}$ is shown in Algorithm 1. The perturbation method is a combination of edge sampling and edge flipping, i.e., edge randomization. It operates on the adjacency matrix A of the input graph. a_{ij} refers to the entry of A in row i and column j and thus contains the information if an edge between node i and node j ex-

Algorithm 1 Graph Perturbation Algorithm $\mathcal{PIG}_{pert}$

```
function perturbGraph(Graph G = (V, E), privacy parameter s)
    construct graph G_pert = (V', E') where V' = V
    construct adjacency matrix A from E of G
    initialize the adjacency matrix A' for E' of G_pert
    for all a_ij ∈ A with i < j do
                                                ▷ Preservation
        if a_ij is chosen with probability 1 − s then
            set a'_ij = a'_ji = a_ij in A' of G_pert
        else
                                                ▷ Randomization
            if 0 is chosen with probability 1/2 then
                set a'_ij = a'_ji = 0 in A' of G_pert
            else
                set a'_ij = a'_ji = 1 in A' of G_pert
            end if
        end if
    end for
    return G_pert
end function
```

ists. The existence of an edge is represented by a value of one, whereas an absence is represented as zero. As the input graph is undirected, the adjacency matrix is symmetric. This property is preserved in the perturbation step. As the definition of self-loops does not make sense in the case of undirected graphs, the corresponding entries a_{ii} are and also remain zero in the perturbed entries a'_{ii}.

The perturbation consists of the following mechanism: For each entry in the adjacency matrix, it is first determined if preservation or randomization should be performed. In order to make this choice, we introduce a privacy parameter s ($s \in (0, 1]$). Preservation is chosen with probability $(1 - s)$, whereas randomization is chosen with probability s. The higher s is, the more entries in the matrix are randomized. In the case of preservation, the original entry of the adjacency matrix of the unperturbed graph is preserved in the perturbed version of the graph. With randomization, the entry in the perturbed version of the graph gets assigned 1 with probability $\frac{1}{2}$ and 0 with the same probability. The assignment and thus the absence or presence of this particular edge in the graph is independent of its existence or absence in the original input graph.

Theorem 1. *Edge-Differentially Private $\mathcal{PIG}$*
$\mathcal{PIG}$ guarantees 1-edge-differential privacy for $\epsilon \geq \ln(\frac{2}{s} - 1)$ ($s \in (0, 1]$).

The idea behind the proof of Theorem 1 is as follows: The proof is based on proving the basic definition of differential privacy (see Definition 2). The probability that $\mathcal{PIG}$ calculates a certain clustering result consists of two terms: the probability that $\mathcal{PIG}_{pert}$ returns a certain perturbed version of the original graph and the probability that the graph clustering algorithm used in $\mathcal{PIG}$ calculates that result on the perturbed graph. In 1-edge-differential privacy neighboring graphs can only differ in one edge. Thus, the ratio of the probabilities that two neighboring graphs are perturbed to an equal graph is only dependent on the ratio of the probabilities that the differing edge gets assigned the same value in $\mathcal{PIG}_{pert}$. According to Algorithm 1, an entry preserves its original value in the perturbed graph with a probability of $1 - \frac{1}{2}s$, and gets assigned its opposite value with a probability of $\frac{1}{2}s$. With this information, the probability ratio that two neighboring graphs are perturbed to the same graph can be expressed and Theorem 1 can be proven. For the proof of Theorem 1 please refer to [20].

Choice of Privacy Parameter s.

Due to its correlation with parameter ϵ in ϵ-differential privacy, $\mathcal{PIG}$'s privacy parameter directly influences the extent of privacy that $\mathcal{PIG}$ is able to guarantee. But what is a sufficient value for s? It is necessary to find a trade-off between the quality of $\mathcal{PIG}$'s clustering result and privacy guarantees in order to choose parameter s.

Ideally, an adversary cannot say for any edge in the perturbed graph whether or not it existed in the original graph with any greater confidence than if they were to flip a coin, i.e., a confidence greater than 50%. The idea to achieve this is to set s to such a value that the expected density of the perturbation result will be twice the original density. This implies that for each edge in the perturbed graph, an adversary can never say with more than 50% confidence that this edge was also present in the original graph. This is particularly relevant for very sparse graphs. For small s it occurs rarely that the perturbation will both be at an entry in the adjacency matrix where the edge is set to exist *and* the perturbation changes that entry.

Theorem 2. *Choice of privacy parameter s*
If privacy parameter s has the following value, it then holds that the expected density in the perturbed graph is twice the original density d ($d \leq 25\%$).

$$(1 - s) \cdot d + \frac{s}{2} \geq 2 \cdot d$$
$$\Leftrightarrow \quad s \geq \frac{2 \cdot d}{1 - 2 \cdot d}$$

The derivation of the expected density is shown in Theorem 4. This way of choosing s is possible up to a density of 25% because the maximum value for s is reached at that point. However, this density limit covers virtually all real-world graphs. At higher densities, a new trade-off between privacy and usability would have to be found.

Influence on Graph Structure.

As $\mathcal{PIG}_{pert}$ changes the edge structure of the input graph, we examplarily analyze the following three important graph properties upon which cluster definitions are based: (1) connectivity that depends on the knowledge of the exact graph structure, and (2) the number of edges, and (3) the density that both are calculated based on the number of edges in the graph. We determine the expected changes of the graph properties in terms of the properties of the original graph. Given a graph $G = (V, E)$, $n = |V|, m = |E|$, we refer to $P = (V_P, E_P)$ as a perturbed version of graph G. $\mathcal{PIG}_{pert}$ preserves an entry in the adjacency matrix with probability $1 - \frac{1}{2} \cdot s$; an entry gets assigned the opposite value in the perturbed graph with probability $\frac{1}{2} \cdot s$.

Connectivity As the perturbation mechanism operates on one edge at a time and does not take the other entries of the adjacency matrix into account, it is possible that the perturbed version of a connected input graph is unconnected.

Number of Edges. The expected number of edges $\mathrm{E}[|E_P|]$ in the perturbed graph P depends on the expected number

of preserved edges and that of added edges (i.e., flipped entries in the adjacency matrix of G). The adjacency matrix of G contains $(\frac{n\cdot(n-1)}{2} - m)$ changeable zero entries which can result in added edges in P. The main diagonal of the matrix contains non-changeable zero entries as self-loops are not allowed in the graph.

THEOREM 3. *Expected Number of Edges in P*

$$\begin{aligned}\mathrm{E}[|E_P|] &= (1 - \frac{1}{2}\cdot s)\cdot m + (\frac{n\cdot(n-1)}{2} - m)\cdot\frac{1}{2}\cdot s \\ &= (1-s)\cdot m + \frac{n\cdot(n-1)}{4}\cdot s\end{aligned}$$

In a sparse graph, there exist only few edges and many changeable zero entries in its adjacency matrix. Thus, perturbing such a graph results in a high increase in the number of edges. The perturbed version of a graph with half of the maximum number of possible edges is expected to have the same edge count as before. If more than half of the possible edges are present, the number of edges decreases in the perturbed graph.

Density. The density of a graph G is defined as the ratio of the number of edges in G and its number of possible edges. The number of possible edges only depends on the number of nodes in the graph and the fact that the graph is undirected. Thus, it is not influenced by the graph perturbation, which only operates on the edge set of a graph.

THEOREM 4. *Expected Density d_P of P*

$$\mathrm{E}[d_P] = \frac{\mathrm{E}[|E_P|]}{\frac{n\cdot(n-1)}{2}} = (1-s)\cdot d + \frac{s}{2}$$

6. EVALUATION

The goal of our empirical evaluation is to analyze the impact of $\mathcal{PIG}$ on cluster quality, i.e., its influence on the clustering result. Furthermore, we perform our evaluation on synthetic graphs that vary in their density. This is interesting as the behavior of $\mathcal{PIG}_{pert}$ changes with increasing density. We use certain graphs and two graph clustering approaches, and from there we generalize our results to properties a graph and clustering approaches must have in order to cope with the perturbation introduced with $\mathcal{PIG}$.

Setup.

Graphs. The synthetic graphs are generated based on [16]: both the community sizes and the number of edges per nodes are power law distributed. As real-world graphs, we use the Disney graph [21] that is a subgraph of the Amazon co-purchasing network with a small number of nodes and the Facebook graph [18] with more than ten times the nodes.

The cluster quality is measured by means of the F1 score. The non-perturbed clustering result is used as ground truth.

Graph Clustering Approaches. We use $\mathcal{PIG}$ with two different graph clustering approaches: *SCAN* [28] and *Graph k-Medoids* [23]. *SCAN* adapts the cluster definition of DBSCAN [5] to graphs. The cluster definition is based on the density of the neighborhood (ε-neighborhood), i.e., with how many neighboring nodes a node shares a certain number of neighboring nodes. If the size of the ε-neighborhood exceeds a threshold μ, the node becomes a core object. The graph perturbation of $\mathcal{PIG}$ directly influences this property. Adding and removing edges in a graph changes which nodes are in the neighborhood of a certain node. As the perturbation is done randomly, similar neighborhoods of neighboring nodes may become dissimilar when common nodes vanish and disjoint new node sets can be added to the neighborhood. *Graph k-Medoids* adapts the cluster definition of k-medoids [14] to graphs. On the one hand, its cluster definition is based on shortest paths which $\mathcal{PIG}_{pert}$ can affect to a great extent. On the other hand, it has the two characteristics that the number of clusters is set to k and that all nodes are assigned to exactly one cluster. Thus, it is interesting whether these characteristics can reduce the influence of $\mathcal{PIG}_{pert}$ on the clustering result.

Cluster Quality.

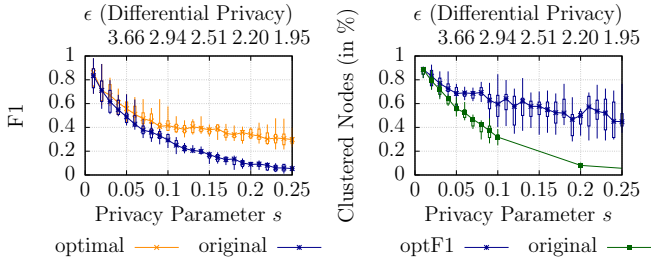
Increasing parameter s means having more changes in the input graph. In Theorem 2, we have presented a heuristic that guarantees a sufficient amount of privacy while minimizing the required amount of perturbation. Thus, we evaluate the impact of the perturbation on cluster quality for a range of s around following minimum values resulting from the heuristic.

	Density	$s \geq$	$\epsilon \leq$
Synthetic graph (506 nodes)	1.29%	0.027	4.292
Disney graph	4.39%	0.096	2.988
Facebook graph	1.08%	0.022	4.499

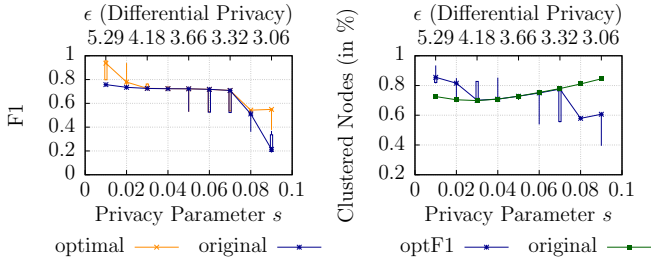
As a general result, increasing privacy parameter s beyond a certain range around the minimum value and thus decreasing ϵ increases the privacy guarantees at the cost of usable clustering results. Due to space limitations, the detailed evaluation of this aspect is in [20].

$\mathcal{PIG}$-SCAN. Figure 2 shows the cluster quality of *$\mathcal{PIG}$-SCAN* for the real-world graphs, compared to the clustering results of *SCAN*. It also contains the corresponding cluster result statistics. The key *original* means that the same parameters as for the clustering on the non-perturbed graph are used. The key *optimal* means that we use those parameters that result in the highest F1 score. Thus, the comparison with the optimal parameter setting shows how close the clustering results on the perturbed and non-perturbed graphs are at best. However, it is difficult or impossible to determine the optimal parametrization without first performing the clustering on the non-perturbed graph and using the result as ground truth. Such a proceeding can result in a privacy risk, as the perturbed clustering result contains additional information on the non-perturbed one. This is not the case if the parametrization is independent of the non-private clustering result.

For the Disney as well as the Facebook graph the cluster qualities are very close to each other for the different parametrizations. First, we consider the *Disney* graph in Figure 2. The decrease in the cluster quality for the original parameter setting is the result of the decrease of the number of clustered nodes. The higher s is, the more random edges are added. This influences the neighborhood structure of the nodes and thus the similarity of the neighborhoods between neighboring nodes. Adding random neighbors implies that adding the same new neighbors to nodes of the former neighborhood is unlikely. As a result, the ε-threshold of *SCAN* can no longer preserve the same set of core objects. If this set changes, the set of nodes which are candidates for a cluster also changes. Thus, the cluster structure itself



(a) Disney data set.



(b) Facebook data set.

Figure 2: Cluster quality of $\mathcal{PIG}$-*SCAN*.

greatly differs from the non-private clustering result. Thus, for $s = 0.1$, the F1 score is only about 0.3.

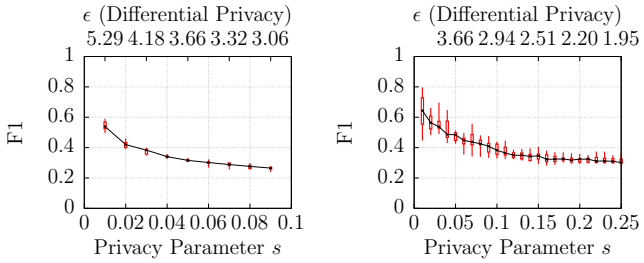
Result 1. Graphs with a high density like the Disney graph (4.39%) require a relatively high s value and thus a lot of perturbation. Additionally, if such a graph does not contain clearly separated clusters in its non-perturbed version – as it is the case for the Disney graph – those changes in the graph structure have an even higher impact. This makes it difficult for density-based approaches like *SCAN* to come up with a useful private clustering result.

For the quality results on the Facebook graph in Figure 2, the F1 score is almost constant for $s \in [0.01, 0.07]$. This behavior can be explained with *SCAN*'s parameter setting together with the structure of the Facebook graph. The Facebook graph is a graph with 4,039 nodes and a density of 1.08%. The non-private clustering result only contains six clusters with almost 90% clustered nodes. Using *SCAN* with $\mu = 160$ requires that core objects must have many neighbors, and this results in a high intra-cluster connectivity. *SCAN*'s ε is the minimum threshold of how similar the neighborhoods of neighboring nodes must be. Thus, setting *SCAN*'s ε= 0.1 reduces the required similarity. This particular parameter setting can cope very well with the additional noise in the graph, resulting in an F1-score of about 0.7 for the minimum required value $s = 0.03$. However, using such an extreme parametrization does not always result in stability against the perturbation. The number of clustered nodes for the original parametrization is about 20% less than the number for the non-private result. The fact that this results in a high F1 score and accuracy means that most of the nodes are true positives wrt. the non-perturbed clustering result.

Result 2. Sparse graphs like the Facebook graph which allows clustering almost all nodes in few, clearly separated clusters can preserve their strong intra-cluster connectivity upon perturbation. Thus, the impact of $\mathcal{PIG}$ on the cluster quality is small and results in useful clustering results.

Result 3. SCAN's thresholds do not strictly bound the neighborhood size, but allow variations in the size that still pass them. Thus, graph clustering approaches which do not require an exact edge structure, but can cope with neighbor changes, and which bound their requirements by thresholds seem to be able to cope with $\mathcal{PIG}$.

$\mathcal{PIG}$-Graph k-Medoids. We evaluate the impact of $\mathcal{PIG}$-*Graph k-Medoids* on the quality of the clustering result on the synthetic graphs with 506 nodes and the *Disney* graph. We use the same parametrization for the clustering on the perturbed graph as used on the non-perturbed one. The cluster quality of $\mathcal{PIG}$-*Graph k-Medoids* is shown in Figure 3. For the synthetic graph, the F1 score is quite low. For the minimum recommended value of $s = 0.03$, there are only quality values under 0.4. Thus, the perturbed clustering result is no longer able to imitate the non-private clustering result very well. The reason is that its cluster definition uses shortest paths that are highly sensitive to edge changes. As a consequence, we have to negate the hypothesis that the two characteristics of *Graph k-Medoids* can reduce the impact of $\mathcal{PIG}$ on cluster quality. An interesting outcome is that $\mathcal{PIG}$ with *Graph k-Medoids* produces clustering results of higher quality on the Disney graph than with *SCAN*, especially as the result of $\mathcal{PIG}$-*SCAN* are the worst on this graph. For the minimum recommended $s = 0.1$, the F1 score is almost 0.4. We hypothesize that the positive outcome is correlated with the special structure of this real-world graph and thus the perturbation can only influence the shortest path structure to a small extent.



(a) Synthetic graph (506 nodes). (b) Disney graph.

Figure 3: Cluster quality of $\mathcal{PIG}$-*Graph k-Medoids*.

Result 4. Clustering algorithms whose cluster definition is based on metrics that are highly sensitive to edge changes do not perform well with $\mathcal{PIG}$. For such approaches, the privacy guarantees dominate the usability.

Graph Density Variation. In the synthetic graphs, the nodes have many edges within their cluster and only few edges to nodes outside of it. Thus, increasing the density for those graphs means strengthening the structure within the communities and then adding more connections to nodes outside the community. With this behavior in mind, the cluster quality results shown in Figure 4 are as expected. For both node sizes, the densest graph has the highest cluster quality. We vary the density in 0.5% steps up to a density of 2.5%. Much higher densities are not possible with the graph generator used and 40 communities. The minimum recommended s values are as follows:

		1.0%	1.5%	2.0%	2.5%
506 nodes	$s \geq$	0.03	0.04	0.05	0.06

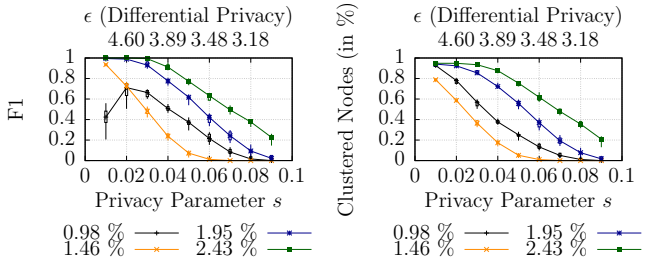


Figure 4: $\mathcal{PIG}$-*SCAN* and density variation on synthetic graphs with 506 nodes.

The F1 score on the graph with density 0.98% is slightly better than the one on the graph with density 2.43%. An interesting outcome is the cluster quality on the graph with density 1.46%. For $s > 0.02$, the cluster quality is the worst compared to the quality on the other graphs. A reason is the percentage of clustered nodes: The higher s is, the fewer nodes are clustered. Compared to the results on the other graphs, it has the lowest clustered-nodes-rate. This has to do with the concrete graph structure and is not dependent on the density. The degree distribution can also not explain why the percentage of clustered nodes is the smallest. The graph with the lowest density has a similar degree distribution.

Result 5. The cluster quality highly depends on the s value chosen. The denser the graph is, the higher s must be. Thus, better cluster quality results on denser graphs are relativized compared to sparser graphs. This is because the denser ones require higher s values.

7. CONCLUSIONS

Publishing usable graph clustering results while giving strong privacy guarantees is an important and challenging task. It is important as a graph clustering result can reveal information that is not directly encoded in the result, e.g., parts of the original graph structure.

In this paper, we have adapted differential privacy to graph clustering. We have developed a neighboring graphs definition suitable for graph clustering results. Based on this definition, we have proposed m-edge-differential privacy. Furthermore, using graph perturbation as noise adding mechanism allows guaranteeing differential privacy for arbitrary graph clustering approaches. Based on those results, we proposed $\mathcal{PIG}$ that guarantees *1-edge-differential privacy*. It independently perturbs each entry of the adjacency matrix. In a thorough evaluation, we have analyzed the impact of $\mathcal{PIG}$ on the graph structure and cluster quality. We have shown that $\mathcal{PIG}$ can lead to good results and can preserve the cluster structure up to a certain extent.

This paper is a first stab at the problem of differentially private graph clustering. We see several topics worthy of further research. First, it might be worthwhile to classify the existing graph clustering approaches according to their ability to cope with a perturbation such as the one of $\mathcal{PIG}$. Then, a user can better choose a clustering algorithm for $\mathcal{PIG}$ and can get a good and at the same time private clustering result. Second, we want to develop adaptation techniques for each category: Slightly adapting the graph clustering algorithm could result in more robustness against the perturbation of $\mathcal{PIG}$.

8. REFERENCES

[1] Gaydar Project at MIT, 2013. last accessed December 30, 2013.

[2] R. Bhaskar et al. Discovering Frequent Patterns in Sensitive Data. In *KDD*, 2010.

[3] C. Dwork. Differential Privacy. In *Automata, Languages and Programming*, volume 4052. 2006.

[4] C. Dwork et al. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*, volume 3876. 2006.

[5] M. Ester et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*, 1996.

[6] D. Feldman et al. Private Coresets. In *STOC*, 2009.

[7] A. Gupta et al. Differentially Private Approximation Algorithms. *CoRR*, abs/0903.4510, 2009.

[8] M. Hay et al. Accurate Estimation of the Degree Distribution of Private Networks. In *ICDM*, 2009.

[9] S.-S. Ho and S. Ruan. Differential Privacy for Location Pattern Mining. In *SPRINGL*, 2011.

[10] G. Jagannathan, K. Pillaipakkamnatt, and R. N. Wright. A New Privacy-Preserving Distributed k-Clustering Algorithm. In *SDM*, 2006.

[11] S. Jha, L. Kruger, and P. McDaniel. Privacy Preserving Clustering. In *Computer Security (ESORICS)*, volume 3679. 2005.

[12] V. Karwa et al. Private Analysis of Graph Structure. *Proceedings of the VLDB Endowment*, 4(11), 2011.

[13] S. Kasiviswanathan et al. Analyzing Graphs with Node Differential Privacy. In A. Sahai, editor, *Theory of Cryptography*, volume 7785. 2013.

[14] L. Kaufman and P. Rousseeuw. *Clustering by Means of Medoids*. Reports of the Faculty of Mathematics and Informatics. Delft University of Technology. 1987.

[15] K. Kumar and C. Rangan. Privacy Preserving DBSCAN Algorithm for Clustering. In *Advanced Data Mining and Applications*, volume 4632. 2007.

[16] A. Lancichinetti and S. Fortunato. Benchmarks for Testing Community Detection Algorithms on Directed and Weighted Graphs with Overlapping Communities. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 80(1), 2009.

[17] N. Li et al. PrivBasis: Frequent Itemset Mining with Differential Privacy. *Proceedings of the VLDB Endowment*, 5(11), July 2012.

[18] J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. In *Advances in Neural Information Processing Systems 25*, 2012.

[19] D. Mir and R. Wright. A Differentially Private Graph Estimator. In *ICDMW*, 2009.

[20] Y. Mülle. Sanitizing Graph Clustering and Community Detection Results Through Differential Privacy. Master's thesis, Karlsruhe Institute of Technology (KIT), 2014.

[21] E. Müller et al. Ranking Outlier Nodes in Subspaces of Attributed Graphs. In *ICDEW*, 2013.

[22] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth Sensitivity and Sampling in Private Data Analysis. In *STOC*, 2007.

[23] M. J. Rattigan, M. Maier, and D. Jensen. Graph clustering with network structure indices. In *ICML*, 2007.

[24] A. Sala et al. Sharing Graphs Using Differentially Private Graph Models. In *IMC*, 2011.

[25] E. Shen and T. Yu. Mining Frequent Graph Patterns with Differential Privacy. In *KDD*, 2013.

[26] J. Vaidya and C. Clifton. Privacy-Preserving k-Means Clustering over Vertically Partitioned Data. In *KDD*, 2003.

[27] Y. Wang et al. On Learning Cluster Coefficient of Private Networks. In *ASONAM*, 2012.

[28] X. Xu et al. SCAN: A Structural Clustering Algorithm for Networks. In *KDD*, 2007.

Big Graph Privacy

Hessam Zakerzadeh
University of Calgary
hzakerza@ucalgary.ca

Charu C. Aggarwal
IBM T.J. Watson Research Center
charu@us.ibm.com

Ken Barker
University of Calgary
kbarker@ucalgary.ca

ABSTRACT

Massive graphs have become pervasive in a wide variety of data domains. However, they are generally more difficult to anonymize because the structural information buried in graph can be leveraged by an attacker to breach sensitive attributes. Furthermore, the increasing sizes of graph data sets present a major challenge to anonynization algorithms. In this paper, we will address the problem of privacy-preserving data mining of massive graph-data sets. We design a *MapReduce* framework to address the problem of attribute disclosure in massive graphs. We leverage the *MapReduce* framework to create a scalable algorithm that can be used for very large graphs. Unlike existing literature in graph privacy, our proposed algorithm focuses on the sensitive content at the nodes rather than on the structure. This is because content-centric perturbation at the nodes is a more effective way to prevent attribute disclosure rather than structural reorganization. One advantage of the approach is that structural queries can be accurately answered on the anonymized graph. We present experimental results illustrating the effectiveness of our method.

1. INTRODUCTION

Network data has become increasingly important in recent years because of the greater importance of various application domains such as the Web, social networks, biological networks, and communication networks. The semantics and the interpretation of the nodes and the links may vary significantly with application domain, *e.g.* in a social network nodes can represent individuals and their connections capture friendship, while in a gene regulatory network nodes are genes and connections refer to their interactions. These graph data carries valuable information and are analyzed in various ways to extract new knowledge. For example, social networks provide significant insight about psychological behavior of individuals or gene regulatory networks are widely studied to elucidate mechanism of diseases.

A major problem with the release of various social networks is that the nodes are often associated with sensitive information about the individual, such as their posts, tweets, interests, hobbies or their political opinions. Individuals might be willing to share such information with their friends, close circles or a particular community but not necessarily with the broader public. An example is the social network published by [18] which captures the sexual contacts, shared drug injections and *HIV* status of individuals. In such cases, a straightforward elimination of only the *directly* identifying information, such as the name or the Social Security Number (SSN) is usually used. However, such an approach, referred to as naive anonymization, is generally not successful in protecting privacy for graph data[5, 12], as the case for multidimensional data.

The structural pieces of information (*e.g.* friendship links) in a social network are usually either far less sensitive than the personal information of a user or are publicly available. An example is a co-authorship network in which links are publicly available, however each author may consider details of his/her ongoing research sensitive. On the other hand, for a social network release to be truly useful, such content-centric information needs to be released along with the social network structure.

In the context of graphs, a major challenge is that the structural information embedded in graphs such as the node degrees are often highly revealing information. Such information can be leveraged by an adversary to launch privacy attacks. In general, attacks on graph data are categorized as either *active* or *passive* [5]. In active attacks, an adversary has the ability to influence the structure of the graph, such as the social network. In such cases, the adversary may construct a highly distinguishable pattern (subgraph), and establish carefully chosen connections with target victim nodes. The adversary can then leverage their actively created subgraph to efficiently determine the identity of the targeted nodes in the released network, even when it is anonymized. In *passive* attacks, the adversary does not have the ability to influence the structure of the graph. The released graph is only "passively" available to the adversary. The adversary may then use all the publicly available background information to learn more about sensitive information belonging to the individual. This can result in privacy violation. Passive attacks are more realistic because attackers can usually access the network modification platform in only a local or limited way, and consequently are not able to significantly alter the overall graph structure.

The privacy of individuals (nodes) in graphs can be breached in three different ways. The first is referred to as *identity disclosure*. This refers to the fact that the identity of individuals associated with graph nodes are disclosed. The second is that of *link disclosure*. In this case, relationships between individuals are considered sensitive and an adversary aims at disclosing the relationship between victims. The identity and link disclosure attacks are generally related because the disclosure of a sensitive link requires the identification of the node identities as an intermediate step. Both forms of privacy are fully related to structural anonymization, and do not even assume that content is released along with the graph. The fi-

PAIS '15 Brussels, Belgium

nal setting is that of *attribute disclosure*. In this case, an attacker aims at finding out sensitive *content attribute(s)* associated with a victim. In this setting, content attributes are always released along with graph nodes, and a user may not wish to disclose this information. The structural information about the graph increases the ease of making content-centric attacks in this case. Even though this kind of disclosure poses a more significant problem, it remains almost completely unaddressed in the literature. This is the main focus of this paper.

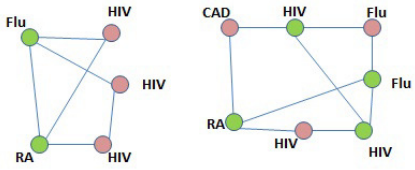
Many privacy models[14, 24, 26, 23, 8] have recently been designed for private release of network data. The models make different assumptions about the background information of the adversary. For example, Liu and Terzi[14] proposed a model, namely k-degree, to prevent *identity disclosure* against an attacker who has knowledge about the *victim node's degree*. The k-neighborhood model[24] prevents *identity disclosure* against an attacker enriched with knowledge about the *immediate neighbors of a target victim*. Algorithms to enforce these privacy models typically suffer from two problems. First, they are expensive and particularly difficult to implement for very large graphs. This is important, because the sizes of graphs have continued to increase significantly in recent years. Secondly, all recent methods focus almost exclusively on *structural* anonymization, with little or no focus on the interplay between content and structure in the context of sensitive *attribute* disclosure of nodes. The latter is usually much more sensitive. Individuals do not wish their views, controversial opinions, and proclivities to be released in the open without their knowledge. Structural anonymization dramatically degrades the graph structure, and provides little *attribute-wise* protection in return for this large loss in utility. It is important to point out that different graph domains have different levels of relative sensitivity of links and attributes. For example, in some domains, the sensitivity of attribute disclosure is much greater than link disclosure. Generally, graph data is sufficiently complex that it is impractical to prevent all forms of disclosure with a single anonymization approach. In such cases, it makes little sense to perturb the structure of the graph. Rather, the use of traditional attribute-centric modification is sufficient as long as the graph structure is taken into account during attribute perturbation. Taking the structure into account during attribute perturbation ensures that important real-world properties of graphs, such as homophily and content-structure locality, are preserved by the anonymization process.

In this work, we devise the first approach to cope with these content-centric issues in the context of very large graphs. Our approach is a simple, yet efficient algorithm to prevent attribute disclosure attacks in the passive scenario, while publishing the full graph structure. Our anonymization technique is best suited to the big-graph scenarios where there are millions or billions of nodes and edges. The existing works which prevent different types of disclosure either fail or suffer from long running time when applied on such graphs. Hence our proposed algorithm, which is a *MapReduce* algorithm, is the first attempt to address the privacy of *big graphs*.

1.1 The Problem

An attacker may obtain different kinds of structural background knowledge about victim nodes. Hay *et al.* [12] systematically captures three prominent types of background knowledge: *vertex* knowledge, *subgraph* knowledge and *hub fingerprint* knowledge. Vertex knowledge refers to node degrees, while subgraph and hub fingerprint knowledge describe (partial) subgraphs around nodes and their distances from hub nodes, respectively. However, some types of background knowledge are more difficult to acquire so the precise kind of knowledge available depends on the problem setting. Some existing proposals [24, 26] consider very powerful adversaries with subgraph or hub fingerprint knowledge. Our work currently considers adversaries with vertex knowledge (*i.e.* victim nodes degrees) in the context of *big graphs*, because this represents a large class of potential attackers and a readily available form of background knowledge. We anticipate considering more sophisticated attack models in future work. We assume that node-degree information is known to the attacker and their goal is to determine *sensitive attribute(s)* values associated with victim node(s). In other words, the adversary is undertaking an *attribute disclosure attack* with node-degree information. Figure 1 illustrates an anonymized graph in which the *identity* of a person is anonymized with respect to the degree. However, an attacker, supplied with nodes degrees, can easily conclude that individuals with node-degree value of 2 are suffering from *HIV*.

Figure 1: Example of two published graphs. Nodes sensitive attributes are shown by each nodes. The graph in (a) does not provide any protection against attribute disclosure attack, while the graph in (b) satisfies 2-diversity.



(a) *Vulnerable graph* (b) *Satisfying 2-diversity*

To prevent this attack, we leverage the privacy models in relational data and mandate that the values of sensitive attributes of nodes with the same degree should be well-represented. Although our proposed *MapReduce* algorithm is capable of adopting all types of diversity (*e.g.*, recursive ℓ-diversity, t-closeness), depending on different requirements on sensitive value distribution, we only consider distinct ℓ-diversity in this work because of its simplicity and fundamental nature. Therefore, we enforce the relative frequency of each sensitive value for the set of nodes with the same degree to be at most $\frac{1}{\ell}$.

The ℓ-diversity concepts in the relational setting can be generalized to graphs as follows. Each record corresponds to an individual in the relational model, while nodes represent individuals in the graph data. The quasi-identifiers are a subset of attributes in relational data and records having the same value for their quasi-identifiers form an equivalence class. On the other hand, the node-degree is the quasi-identifier and nodes with the same degree form an equivalence class in graph data. Figure 1b illustrates a published graph satisfying 2-diversity in which degree-based diversification of sensitive attributes assures a confidence of at least $\frac{1}{2}$.

We consider the problem of attribute disclosure attack in the context of big graph data with the use of the ℓ-diversity model. Preserving the privacy of such a graph can be enabled with the use of distributed frameworks such as *MapReduce*. Therefore, we address the following question in this work:
"How to prevent attribute disclosure attacks using the ℓ-diversity privacy model in big graphs?".

2. RELATED WORK

The problem of privacy preservation was first studied in [3]. This approach was based on noise-based perturbation of the underlying data. Subsequently and starting with Samarati's seminal work[19], a significant amount of research has been done on the problem of privacy preservation and numerous privacy-preserving models (*e.g.* k-anonymity[19], ℓ-diversity[15], t-closeness[13], δ-presence [16], or differential privacy[9, 10]) have been proposed to protect published data against attackers. Each privacy model can prevent par-

ticular types of attacks and makes some assumptions about the attacker's background knowledge. Besides, each type of data, *e.g.* relational data, streaming data or graph data, poses its own unique requirements and challenges and mandates a different privacy model.

Backstorm *et al.* [5] were the first to point out that simply removing identifier information from nodes in a graph does not prevent identity disclosure. The re-identification can occur through structural information that an attacker can obtain through various sources. Hay *et al.* [12] systematically modeled major variants of adversary structural background information.

Starting with the work of [5], graph privacy received increasing attention. The models generally aim to prevent three categories of attacks: *identity disclosure attack*, *link disclosure attack*, and *attribute disclosure attack*. The identity disclosure attack has been studied in works such as [14, 24, 26]. Liu and Terzi[14] considered an adversary armed with node-degree information and proposed an edge addition/deletion technique for anonymization. The solution alters an input graph such that for every node v, there exist at least $k-1$ other nodes with the same degree as v in the published graph. Zhou and Pei[24] assumed that the attacker is aware of subgraph constructed by immediate neighbors (1-neighborhood subgraph) of a victim node. They proposed an algorithm which organizes nodes into groups, according to their neighborhoods, and then anonymizes the neighborhoods of vertices in the same group. In [26], the authors adopted a more general assumption and investigated a scenario where the attacker knows any subgraph around a victim node (d-neighborhood subgraph). The authors then aimed to construct a graph in which there exist at least k isomorphic subgraphs, in the published graph, to each given subgraph in the input graph.

Other works such as [23, 20, 21] focused on the link disclosure attacks. Zheleva and Getoor[23] devised and investigated the effectiveness of several strategies to hide sensitive links within a graph. The work in [20] investigated the interplay between addition/deletion and edge switch techniques and the graph spectrum. The authors have also proposed a spectrum preserving algorithm to prevent link disclosure attacks. Potential disclosure of sensitive links in graph generation algorithms was also examined in [21].

The attribute disclosure attack is explored in a few works[25, 22]. The authors in [25] extended the k-neighborhood privacy such that the ℓ-diversity is preserved over the sensitive attribute of nodes within each group. Yuan *et al.*[22] considered three gradually increasing levels of attacker's background information. Their proposed solution alters the graph structure by adding fake edges and nodes along with generalizing the sensitive values to provide personalized protection. In addition, the graph nodes and their sensitive attributes or a rich social network, in general, can be modeled as a bipartite graph and be anonymized using the techniques proposed by Cormode *et al.*[7]. However, all the proposed works manipulate the graph structure to provide a certain level of privacy. Structural modification dramatically affects the utility that a graph structure provides. In this paper, we put forward an algorithm to prevent the attribute disclosure attack without manipulating the graph structure.

3. PROBLEM DEFINITION

Let the quadruplet $G(V, E, S, f)$ be a simple graph in which V represents a set of nodes (vertices), E is the set of edges, S is the set of sensitive values and $f : V \rightarrow S$ is a mapping function that relates each node to its sensitive value. Table 1 summarizes the list of commonly-used notations in this work. Next we define the notion of ℓ-diversity for a set of nodes.

DEFINITION 1 (ℓ-DIVERSITY PRIVACY CONDITION). *A subset of nodes $V_j \subset V$ in graph $G(V, E, S, f)$ satisfies the ℓ-diversity privacy condition if and only if $\forall v_i \in V_j$ and multiset of sensitive values (S^{V_j}, g) where $S^{V_j} = \{f(v_i) | v_i \in V_j\}$ and $g(\cdot)$ returns the frequency of each sensitive value in V_j, $\forall x \in S^{V_j}$ the inequality $\frac{g(x,V_j)}{\sum_{y \in S^{V_j}} g(y,V_j)} \leq \frac{1}{\ell}$ holds.*

Table 1: List of notations

notation	explanation
v_i	$i^{th} vertex$
$f(v_i)$	function returning the sensitive value of v_i
$deg(v_i)$	degree of vertex v_i
eq^d	set of nodes with degree d (an equivalence class)
N_i	set of immediate neighbors of v_i
S^X	set of sensitive values of set of nodes X
$\|.\|$	size of a set

Next, we generalize the notion of ℓ-diversity for a subset of nodes to the full graph G.

DEFINITION 2 (ℓ-DIVERSIFIED GRAPH). *A graph G with degree set D is called an ℓ-diversified graph if and only if $\forall d \in D$, the set of nodes eq^d with degree d satisfies the ℓ-diversity privacy condition.*

This is the generalization of the distinct ℓ-diversity model[15] to the graph data model. It is possible to define stricter conditions such as recursive or entropy ℓ-diversity. Furthermore, related notions such as t-closeness can be defined.

Our contribution in this work is the proposal of the first *MapReduce* algorithm to create ℓ-diversified big graphs, particularly social networks, so that an attacker supplied by degree information cannot succeed in launching attribute disclosure attacks. The algorithm releases the graph structure intact and in its entirety, thus structural queries can be accurately answered using the published graph. In addition, the algorithm is fully scalable and capable of anonymizing big graphs.

4. MapReduce-BASED PRIVACY ALGORITHMS

As discussed in the social networking literature[6, 11], many big graphs (*e.g.*, social networks, biological networks) are scale-free networks and their degree distribution follow a power-law distribution. This behavior is illustrated in Figure 5. As a rule of thumb, many nodes in a big social network satisfy the privacy condition for practical values of the privacy parameters (*e.g.* ℓ). Therefore, it is more reasonable to first filter the privacy-condition-satisfying nodes out and not involve them in further processing. This dramatically reduces the complexity of privacy preservation process. Algorithm 1 illustrates the steps required to enforce the privacy condition on a big graph. The output of this algorithm is the original graph in which the sensitive values of some nodes are released in group.

In this algorithm, nodes v_i are initially assigned to equivalence classes according to their degrees. In other words, $\mathbb{EQ} = \{eq^d | d \in D\}$ and $eq^d = \{v_i \in V | deg(v_i) = d\}$ where D is the graph degree set. The privacy condition is then checked for each equivalence class eq in $\mathbb{EQ}$, and nodes in equivalence classes not satisfying the ℓ-diversity condition are appended to a list of violating nodes. These privacy-violating nodes are then clustered such that the nodes within each cluster satisfy the ℓ-diversity condition. To cluster nodes, we use an agglomerative clustering in which two clusters are merged in each iteration. However, the merging process must be designed for satisfying the privacy condition. A suitable

Algorithm 1 Big Graph Anonymization Steps

1: $AnonymizationScheme(G)$
2: //G is a simple graph of form (V,E,S,f)
3: $\mathbb{EQ}$= assign nodes with degree d to equivalence class eq^d and form equivalence classes set
4: **foreach** (eq in $\mathbb{EQ}$)
5: **if** (eq does not satisfy the ℓ-*diversity condition)* append nodes $v_i \in eq$ to the violating nodes set VN
6: $\mathbb{C}$=cluster nodes $v_j \in VN$ such that each cluster $c \in \mathbb{C}$ satisfies the ℓ-*diversity condition*
7: define function $f\prime : V \rightarrow S \times \mathbb{N}$ such that
8: **foreach** (v_i in V)
9: **if** (v_i not in VN)
10: $f\prime(v_i) = (f(v_i), 1)$
11: **else**
12: $f\prime(v_i)$ = the multiset of sensitive values of nodes in cluster $c \in \mathbb{C} | v_i \in c$
13: publish $G\prime(V, E, S, f\prime)$

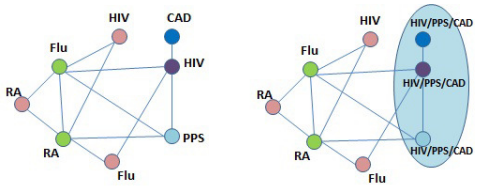
merging criterion is the entropy of sensitive values. Two clusters c_i and c_j are selected when the constituent nodes are connected by at least one edge and cause the maximum change in entropy as a result of the merging. The entropy criterion is stated as follows.

$$\underset{c_i,c_j}{\operatorname{argmax}}\, d(c_i, c_j) = H(c_i \cup c_j) - H(c_i) - H(c_j)$$

where function $H(X)$ denotes the entropy function.

The set of all formed clusters are referred to as $\mathbb{C}$. At the end, the full graph structure is published. However, the sensitive value of node v_j not originally satisfying the privacy condition is replaced by the multiset of sensitive values of nodes in cluster $c \in \mathbb{C}$ that contains v_j ($v_j \in c$). In other words, sensitive values of privacy-violating nodes are released at the cluster level instead of the node level.

Figure 2: Illustration of 2-diversification process of a graph. This process leaves nodes initially satisfying the 2-diversity condition intact and only generalizes sensitive values of 2-diversity-violating nodes.



(a) *A 2-diversity violating graph* (b) *2-divertised version of (a)*

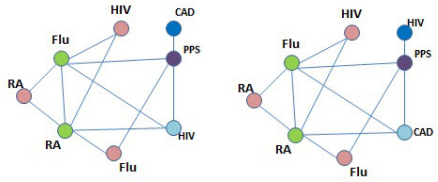
In the clustering step (line 6, Algorithm 1), satisfying the ℓ-diversity condition for each cluster mostly depends on frequencies of sensitive values, which is defined as the function $g(f(v_i), c) : S \rightarrow \mathbb{N}$. This definition of $g()$ results in a *Sensitive Value Frequency aWare* (SVFW) clustering. However, $g()$ can be defined in a more relaxed way $g(f(v_i), c) : S \rightarrow 1$, which basically ignores the frequency of sensitive values within each cluster and turns the process into a *Sensitive Value Frequency aGnostic* (SVFG) clustering. As soon as the number of distinct sensitive values reaches ℓ within a cluster, further merging is no longer performed on that cluster in SVFG, while it may not be case in SVFW. In fact, the SVFG is a relaxed version of SVFW, and apparently an ℓ-diversity-satisfying cluster under SVFW is an ℓ-diversity satisfying cluster under SVFG too, whereas the vice versa may not hold. It is easy to show that SVFG satisfies the ℓ-diversity condition as each node within a given cluster can be related to any of the sensitive values.

Pathological cases might exist in both SVFW and SVFG in which the clustering ends up with a cluster not satisfying the privacy condition. Under such circumstances, sensitive values of nodes belonging to the cluster must be suppressed. Figure 2a demonstrates a graph in which nodes with degrees one, three and four do not satisfy 2-diversity. Applying the anonymization steps in Algorithm 1 results in the 2-diversity-satsifying graph shown in Figure 2b. In this anonymized graph both SVFW and SVFG result in the same anonymized graph as each sensitive value occurs only once in the formed cluster.

A data user, after obtaining the published anonymized graph must instantiate a graph from it. Instantiation means randomly assigning a sensitive value to those nodes whose sensitive values are released as a multiset. A query can be more accurately answered by averaging over multiple instantiations. Figure 3 exemplifies two possible instantiated graphs from the 2-diversity satisfying graph shown in Figure 2b.

Section 4.2 describes how Algorithm 1 can be converted into *MapReduce* jobs to satisfy the ℓ-diversity condition on big graphs. However, before proceeding further, we provide a very brief introduction to the *MapReduce* framework.

Figure 3: Two possible instantiations of 2-diversified graph shown in Fig. 2b



(a) *Instantiation 1* (b) *Instantiation 2*

4.1 MapReduce

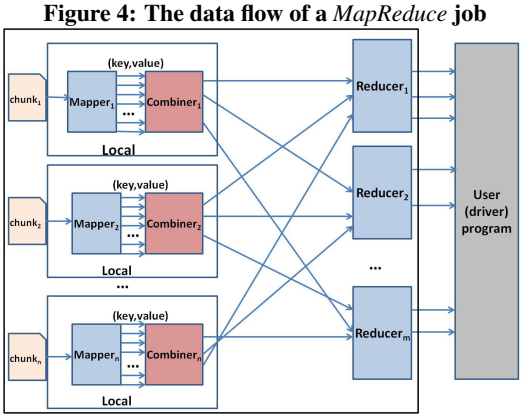
MapReduce is a programming framework proposed by *Google* to enable efficient data processing. As implied by its name, the approach uses distributed *Map* step, followed by a *Reduce* step. These steps must be designed by the application programmer. The *MapReduce* framework splits the data into equal-size chunks, and feeds each chunk to a separate mapper. Each mapper processes its own chunk, and outputs $(key, value)$ pairs. Pairs with the same key are transferred to one reducer by the framework. The set of all reducer outputs are used to construct the final result. An arbitrary function *combiner* can also be defined to reduce the amount of data transfer between mappers and reducers by aggregating the $values$ belonging to each key. There is also a user (driver) program that executes/runs the *MapReduce* program. Figure 4 illustrates the data flow of a *MapReduce* job. We do not define the combiner function in our *MapReduce* jobs as its effect in reducing the data transfer is not significant, considering the mapper output.

4.2 Privacy Algorithm Transformation to MapReduce Jobs

Algorithm 1 consists of two phases, which are referred to as the *pruning phase* and the *clustering phase*, respectively. The pruning phase corresponds to lines 1-5 and the clustering phase to the line 6 in the algorithm. Lines 7-13 involve publishing the resulting anonymized graph, and can also be carried out by a separate job, as will be explained shortly. Here, we demonstrate how each phase can be converted into *MapReduce* jobs.

As a *de facto* standard, we assume edges and attribute informa-

Figure 4: The data flow of a *MapReduce* job



tion of a graph are stored in two separate files: the *relationship* and the *meta-info* file. Each line in the relationship file shows an edge and each line in the meta-info file corresponds to one node and contains the values of different attributes of that node. Minor modification is required for other sorts of graph representations. Besides, for the sake of simplicity, we assume that each node contains only one sensitive attribute, however extension to multi sensitive attributes is straightforward.

4.2.1 Pruning Phase

The pruning phase comprises two *MapReduce* jobs. The first job discovers the immediate neighbors of each node and the second one does the actual filtering (pruning) task based on whether an equivalence class satisfies the ℓ-diversity condition or not.

MapReduce Job for Neighborhood Discovery

This job discovers neighbors of each node. The input to this job is both the relationship and the meta-info files. The mapper and the reducer of this job are as follows:

Mapper: As there are two different input files, mappers are differentiated according to the data chunk they are fed with and their outputs differ depending on whether a mapper gets a chunk of the relationship or meta-info file. If the chunk is coming from the relationship file, then for each record "v_i,v_j" (corresponding to an edge), the mapper outputs two pairs <v_i,v_j> and <v_j,v_i>. However, providing that the mapper is fed by a chunk of meta-info file, it outputs the vertex as the key and its sensitive value as the value.

Algorithm 2 Neighborhood Discovery Job

```
1: Mapper(k,v)
2:   // v can be either of form (v_i,v_j) or (v_i,f(v_i))
3:   if (input chunk belongs to the relationship file)
4:     emit(v_i,v_j)
5:     emit(v_j,v_i)
6:   else
7:     emit(v_i,f(v_i))
```

Reducer: All neighbors of a given vertex (let's say v_i) as well as its sensitive value ($f(v_i)$) are brought together in a reducer. Thus, the reducer receives pair <$v_i, N_i + f(v_i)$> where N_i is the set of all v_i's immediate neighbors. The '+' sign is a simple string concatenation operator. Each reducer then emits the pair <$v_i/f(v_i), N_i$>. The output file of this job is fed into the second *MapReduce* job as input.

MapReduce Job for Filtering

This *MapReduce* job filters out nodes originally satisfying the ℓ-diversity condition and leaves us with only violating nodes. As discussed earlier, it can be modified for other types of ℓ-diversity or even t-closeness.

Algorithm 3 Neighborhood Discovery Job

```
1: Reducer(k,V)
2:   // k is a vertex and V contains all neighbours and the sensitive
       value f(k)
3:   emit(k/f(k),V)
```

Mapper: Each mapper reads a data chunk which has been output from the reducer in the first job. The value of each input record in the chunk is of form $(v_i/f(v_i),N_i)$[1]. The mapper then emits a pair where the key is the degree of v_i, $|N_i|$, and the value is <$v_i/f(v_i), N_i$>. N_i will be used in the clustering phase later.

Algorithm 4 Filtering Job

```
1: Mapper(k,v)
2:   //v is of form (v_i/f(v_i), N_i)
3:   emit(|N_i|,<v_i/f(v_i), N_i>)
```

Reducer: All vertices with the same degree (eq^d) as well as their sensitive values S^d are transferred to the same reducer. The reducer can then simply decide on whether the equivalence class eq^d satisfies the ℓ-diversity or not. If so, all vertices in eq^d are ruled out, otherwise they are output. Therefore, each line in the the reducers' output file contains a privacy-violating node and its sensitive value. The output file(s) will be used as input for the next *MapReduce* job in the clustering phase.

Algorithm 5 Filtering Job

```
1: Reducer(k,V)
2:   //V is a list of (v_i/f(v_i), N_i) where deg(v_i) = d
3:   violation=false
4:   foreach (v_i/f(v_i), N_i) ∈ V
5:     if (freq(f(v_i),S^d)/|S^d| > 1/ℓ)
6:       violation=true
7:       break
8:   if (violation)
9:     foreach (v_i/f(v_i), N_i) ∈ V
10:      emit(v_i/f(v_i), N_i)
```

4.2.2 Clustering Phase

The clustering phase has only one *MapReduce* job. This job groups the privacy violating vertices and forms clusters in which the sensitive values are well-represented. An agglomerative hierarchical clustering algorithm can be used for the clustering. However, merging two clusters must enable ℓ-diversity satisfaction. To achieve this goal, we consider the similarity measure between two neighbor clusters c_i and c_j to be the difference in entropy of sensitive values in the resulting cluster and the original clusters, *i.e.* $H(c_i \cup c_j) - H(c_i) - H(c_j)$[2]. A cluster is removed from further merging as soon as it satisfies the ℓ-diversity condition. The sensitive values of vertices in the remaining cluster may need to be suppressed when they do not satisfy the ℓ-diversity condition.

MapReduce Job for Clustering

The clustering job is in charge of clustering vertices according to sensitive values entropy.

Mapper: Each mapper is fed a chunk consisting of privacy violating vertices (output by the $Filtering$ job). It buffers all the input nodes and clusters them according to entropy increase criterion.

[1]Note that by default the key for each record in the mapper is the record's offset in the input file.

[2]If c_i and c_j are not connected, their similarity will be zero. This can be determined using N_i.

Note that the clustering is influenced by the number of mappers and size of the chunks as violating nodes are split among different mappers. That means a better clustering, in terms of fewer suppressed sensitive values, is expected with fewer mappers or larger chunk size. One mapper operating over all the violating nodes is the ideal case, however it may become a bottleneck in case of having large number of violating nodes.

The output of the *Clustering MapReduce* job is referred to as the Generalized Sensitive Value (GSV) file(s). Each line in the GSV file contains a node name along with the multiset of the cluster sensitive values to which the node belongs. Algorithm 6 shows the pseudocode of this job. Function $multiset(c)$ takes a cluster and returns the multiset of its constituent node sensitive values.

Algorithm 6 Clustering Job

```
1: Mapper(k,v)
2:   // v is of form (v_i/f(v_i), N_i)
3:   append each pair (v_i/f(v_i), N_i) to buffer
4:   if (no more pair)
5:     ℂ = cluster(buffer) //either SVFW or SVFG clustering
6:   for (c in ℂ)
7:     for (v_j in c)
8:       emit(v_j, multiset(c))
```

There is no reducer required for this *MapReduce* job and the mapper's output should be considered as the job's output. To publish the final anonymized graph (lines 7-13, Algorithm 1), the relationship file must be released as original, nonetheless the sensitive values of originally-privacy-violating nodes in the meta-info file must be swapped with their corresponding values in the GSV file. There exist two alternatives to carry out the swap:

1. A *MapReduce* job which caches GSV file and takes the meta-file as input. It then goes through the meta-info file and simply does the swaps. This alternative is suitable when the GSV file is of small size.
2. A *MapReduce* job which joins the meta-info and the GSV files and changes the sensitive values of violating nodes in the meta-info with the corresponding value from the GSV file. This approach is more appropriate for a large GSV file.

The new meta-info file must also be released at the end.

5. DATA TRANSFER ANALYSIS

As shown in Figure 4, each *MapReduce* job involves two data transfers. The first involves data transfer between mappers and combiners. The second involves data transfer between combiners and reducers. Since each mapper and its corresponding combiner run on one node, the first data transfer is local. However, the second data transfer may occur across the network and become a bottleneck. As we have not specified a combiner in this work, we only analyze the amount of data transferred in the second case here.

First MapReduce Job, Pruning Phase

The mapper in the first job doubles the size of the relationships file because it outputs two pairs per input record of the relationships file, but leaves the size of meta-info file unchanged. $|E|$ and $|V|$ show the number of input records from the relationships and the meta-info files, respectively and let b denote the number of bytes required to store a node name (or a sensitive value). So, the mapper's input and output data are of size $2b.|E| + 2b.|V|$ and $4b.|E| + 2b.|V|$.

The reducer then outputs one record per node in which there is the node name, its sensitive value and the list of node's neighbors. The average size of each record is $(2 + \mu).b$ where μ is the average node degree in the input graph. As shown in the social networks literature [4, 6], degree distribution in many social networks follows a power law probability distribution $p(x) \sim x^{-\lambda}$ where $2 < \lambda < 3$. Newman[17] also proved that the average degree in a graph following power law distribution is $\mu = \frac{\lambda-1}{\lambda-2}$ (given $\lambda > 2$). Thus, the reducers' output will be of average size $(2 + \frac{\lambda-1}{\lambda-2}).b.|V|$. In summary, the asymptotical data transfers are:

mapper's input:
$$O(|E| + |V|)$$

mapper's output (reducer's input):
$$O(|E| + |V|)$$

reducer's output:
$$O(\frac{\lambda-1}{\lambda-2}.|V|)$$

Hereafter we only consider the asymptotical analysis of the data transfer.

Second MapReduce Job, Pruning Phase

The second *MapReduce* job is fed by the output of the first job which is of size $O(\frac{\lambda-1}{\lambda-2}.b.|V|)$. It then outputs one pair for each input record in which the key is the node's degree and the value consists of node's name, its sensitive value and neighbors. Thus, the mapper's output has the same magnitude of the input which is

$$O(\frac{\lambda-1}{\lambda-2}.|V|)$$

Afterwards, the reducer prunes out nodes originally satisfying the privacy condition and outputs privacy-condition-violating nodes as well as their sensitive values. Number of output nodes is highly dependant to the sensitive values distribution within different equivalence classes and cannot be easily estimated. However, considering the power law for degree distribution, we can estimate the lower bound of output nodes. The number of nodes for a given degree (let's say x) as follows:

$$\frac{n}{|V|} \sim x^{-\lambda} \rightarrow n \sim |V|.x^{-\lambda}$$

Provided that the number of nodes for a given degree (x) is less than the privacy level (ℓ), nodes with degree x are output by the reducer[3]. To find an estimated lower bound for x we must have:

$$|V|.x^{-\lambda} \lesssim \ell \rightarrow x \gtrsim \sqrt[\lambda]{\frac{|V|}{\ell}}$$

so the estimated size of reducer's output is

$$\Omega(\frac{\lambda-1}{\lambda-2}.|V|.\sum_{x=\sqrt[\lambda]{\frac{|V|}{\ell}}} x^{-\lambda})$$

Third MapReduce Job, Clustering Phase

The input to this *MapReduce* job is the output of the previous job and has the same magnitude. For each input record, which is formed of a node name along with its sensitive value, one pair consisting of the node name and its multiset of sensitive values is output. Therefore, the output size is

$$O(|V|.\sum_{x=\sqrt[\lambda]{\frac{|V|}{\ell}}} x^{-\lambda})$$

6. EXPERIMENTAL RESULTS

In this section, we will study the effectiveness, efficiency, and running time characteristics of the anonymization algorithm. The

[3]Although existence of at least ℓ nodes with degree x is not enough for ℓ-diversity condition to satisfy, it is a necessary condition.

experiments were carried out on Hadoop 1.0.4[4], an open source implementation of the *MapReduce* framework, on the *Hadoop* clusters running on *ACENet*[5]. It has 32 nodes connected through a Gigabit Ethernet connection, each having 16 cores and 64 GB of RAM running Red Hat Enterprise Linux 4.8. Table 2 lists the Hadoop parameters in our experiments[6].

Recall Section 2 stated there is no existing work that prevents attribute disclosure attacks with the fairly limited restrictions associated with our assumptions regarding attacker's background knowledge. For example, Zhou and Pei [25] consider an attacker with 1-neighborhood background knowledge and Yuan *et al.* [22] assumes knowledge about node labels is also available to an attacker. Unfortunately, divergent assumptions about the knowledge held by attackers makes the comparison with our solution impossible. We also note that our goal is to address big graphs while the solutions proposed by other, due to the increased complexity associated with protecting against sophisticated attacks, means these alternative approaches have only been shown to work on comparatively small graphs.

Our anonymization algorithm does not change the graph structure, so all graph properties remain intact and structural queries can be answered accurately using the anonymized graph. However, the sensitive values for some nodes are released as a multiset and in cluster level. This ambiguity introduces some error in answering queries involving the sensitive values, however can be ameliorated by averaging over multiple instantiations. To measure the effect of sensitive values ambiguity, we measure the change of certain queries' results between the original and the anonymized graphs. We considered three types of queries:

- **Pair query** (one hop query): this type of query involves pairs of nodes which are connected. It demonstrates how many nodes from one subpopulation have relation with nodes from another subpopulation in the graph. An example query in this type is "how many users of 19 years old are friends with users of 28 years old?".
- **Trio Query** (two hop query): these queries involve three connected nodes in the graph. it in fact counts triples that satisfy a given condition. A sample query of this type is "how many users of 19 years old are friends with users of 28 years old who are friends with users of 50 years old?"
- **Triangle queries**: this sort of query counts the number of triangles (cliques of size three) that holds a query condition. For instance, "how many users of 19 years old are friends with users of 28 and 70 years old who are friends with each other?".

The utility loss is measured by the average relative query error for each type of query over multiple instantiations. For an anonymized graph and set of n queries $\{q_1, q_2, ..., q_n\}$, the average relative error is calculated as $\frac{1}{n}\sum_{i=1}^{n}\frac{|q_i(a)-q_i(o)|}{q_i(o)}$ where $q_i(a)$ and $q_i(o)$ denote the results of running query q_i on the anonymized and the original graphs, respectively.

The anonymization process only affects a (small) portion of nodes in the big graph. That is, the result of executing these three types of queries on a great portion of the anonymized graph is exactly the same as the original graph. Therefore, to better show the impact of anonymization on each type of query, we extract subgraph which results in different answer for a given type of query between the original and the anonymized graphs and only report the error of executing queries on the subgraph. It is worth noting that the total error (the error on the entire graph) will be much smaller than the reported one because the total error must be calculated on the whole graph in which most nodes remained intact. This subgraph for the pair and triangle queries is the subgraph formed by the initially-privacy-violating nodes and their immediate neighbors (neighbors with distance one). For trio queries, this subgraph is constructed by the initially-privacy-violating nodes and as well as nodes within distance one and two from them. Besides, the running time was measured in terms of the wall-clock time (milliseconds). This provides a good indicator of the overall scalability of the method.

[4] `http://www.hadoop.com`

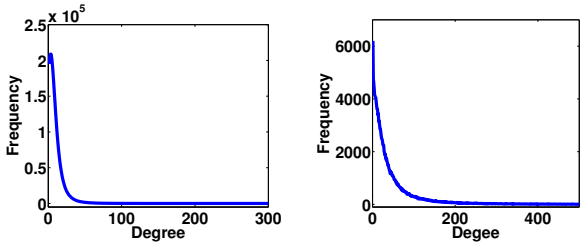
[5] `http://www.ace-net.ca`

[6] The rest of the parameters have the default value.

Table 2: Parameters of Hadoop

Parameter Name	Value
$fs.block.size$	64MB
$io.sort.mb$	1024MB
$io.sort.factor$	50
$dfs.replication$	3

Figure 5: Degree distribution in the $Patent$ and $LastFM$ data sets



(a) *Patent data set* (b) *LastFM data set*

6.1 Data Sets

We used two real big graphs described below:

- **US Patent Citation Graph**: The citation graph includes all citations made by patents granted between 1975 and 1999 and is maintained by National Bureau of Economic Research[2]. This data set contains over 2.9 million nodes and 16.5 million edges. Although it is a directed graph, we consider it as undirected since there is no pair of nodes citing each other. The meta-file also contains the patents' year which is considered as a sensitive attribute.
- **LastFM co-Group Graph**: *Last.fm* is a popular music website which recommends music to users according to their music taste. An anonymous random walk crawl of *Last.fm* is released by Networking group at UC Irvine [1]. The data set contains up to 177K users (nodes) and more than 10M friendship relations (edges) among them. Besides, the meta-file includes users' ages which we consider as the sensitive attribute.

Although these two data sets might fit in the RAM of a commodity PC, they are among the biggest graph data sets publicly available with nodes' attributes. So we utilized them to show the behavior of our proposed solution. As shown in Figure 5, the degree distribution in these graphs follow a power-law distribution. For the sake of clarity, these figures only show the tail and the central part of the degree distribution and the head part is mostly removed.

6.2 Results

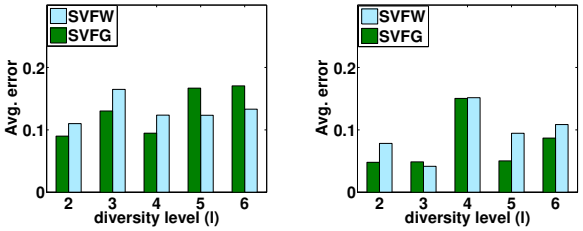
Each anonymized graph was instantiated 30 times and 50 random queries generated by uniformly sampling from the set of sensitive values. The generated queries ran on each instantiated graph and the reported error is averaged over all 1500 queries for each anonymized graph. We set the number of mappers and reducers to 30 and the running time is averaged over three runs. The number of mappers in the clustering phase have also been set to 1 for the

$Patent$ and 10 for the $LastFM$ data set. The selection is due to the number of nodes involved in the clustering. The diversity level (ℓ) also ranges from 2 to 6 in the experiments.

The average relative errors in answering different types of queries on the $Patent$ and $LastFM$ data sets versus the diversity level are shown in Figure 6 through 8. These figures reveal that typically the SVFG technique results in smaller error in answering different types of queries than SVFW technique. We conjecture that this phenomenon is mainly due to fine grain clusters formed in the $SVFG$ technique which can largely eliminates the randomness in the instantiation process. Besides, these figures show that no relation exists between the diversity level and average error.

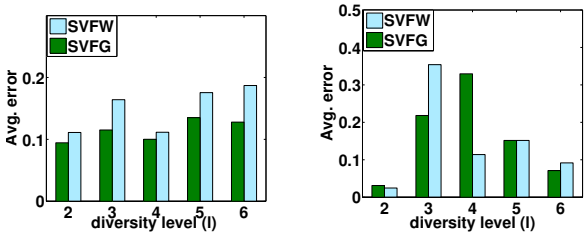
As Figure 9 confirms, the running time and the number of nodes involved in the diversification process have a direct relationship. The increase in the running time is mainly due to the clustering part and can be alleviated by executing more mappers in the clustering phase.

Figure 6: Average relative error of one-hop queries vs. ℓ



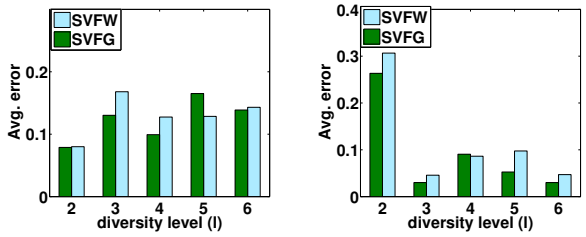
(a) *Patent data set* (b) *LastFM data set*

Figure 7: Average relative error of two-hop queries vs. ℓ



(a) *Patent data set* (b) *LastFM data set*

Figure 8: Average relative error of triangle queries vs. ℓ

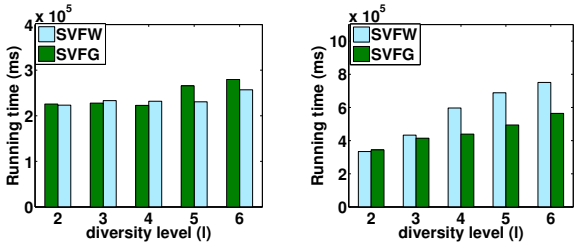


(a) *Patent data set* (b) *LastFM data set*

7. CONCLUSION

Even though publishing social networks between individuals or entities provides various benefits, it poses serious privacy concerns for the underlying individuals or entities. Therefore, the social network must undergo a non-trivial anonymization process before it is released. Anonymization techniques typically alter the graph structure to protect privacy, however the structure manipulation may dramatically degrade the utility of the published graph and turns to be counter intuitive. In addition, the existing anonymization solutions do not scale up and are not practically applicable on real big social network graphs. This work is the first attempt to protect the privacy of individuals in big graphs with no structure manipulation by taking advantage of *MapReduce* paradigm. This approach publishes a

Figure 9: Running time vs. ℓ



(a) *Patent data set* (b) *LastFM data set*

graph in which sensitive attributes are protected and also is capable of answering structural queries as accurate as the original graph. As our future work, we plan to consider more powerful attackers and also leverage *MapReduce* to design fully scalable anonymization techniques to protect other sorts of sensitive information within big social networks such as sensitive links.

8. REFERENCES

[1] Lastfm data set. http://odysseas.calit2.uci.edu/doku.php. Accessed: Feb 2014.

[2] Patent data set. http://www.nber.org/patents/. Accessed: Feb 2014.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. *ACM Sigmod Record*, 29(2):439–450, 2000.

[4] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[5] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, 2007.

[6] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[7] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. Class-based graph anonymization for social network data. 2009.

[8] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. *VLDB*, 2008.

[9] C. Dwork. Differential privacy. In *ICALP*, 2006.

[10] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TTC*. 2006.

[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM Computer Communication Review*, 1999.

[12] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis. Resisting structural re-identification in anonymized social networks. In *VLDB*, 2008.

[13] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, 2007.

[14] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD*, 2008.

[15] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[16] M. E. Nergiz, M. Atzori, and C. Clifton. Hiding the presence of individuals from shared databases. In *SIGMOD*, 2007.

[17] M. E. Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[18] J. Potterat, L. Phillips-Plummer, S. Muth, R. Rothenberg, D. Woodhouse, T. Maldonado-Long, H. Zimmerman, and J. Muth. Risk network structure in the early epidemic phase of hiv transmission in colorado springs. *Sexually transmitted infections*, 2002.

[19] P. Samarati. Protecting respondents identities in microdata release. *IEEE TKDE*, 13(6):1010–1027, 2001.

[20] X. Ying and X. Wu. Randomizing social networks: a spectrum preserving approach. In *SDM*, 2008.

[21] X. Ying and X. Wu. Graph generation with prescribed feature constraints. In *SDM*, 2009.

[22] M. Yuan, L. Chen, and P. S. Yu. Personalized privacy protection in social networks. *VLDB*, 2010.

[23] E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *Privacy, security, and trust in KDD*. 2008.

[24] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *ICDE*, 2008.

[25] B. Zhou and J. Pei. The k-anonymity and l-diversity approaches for privacy preservation in social networks against neighborhood attacks. *Knowledge and Information Systems*, 28(1):47–77, 2011.

[26] L. Zou, L. Chen, and M. T. Özsu. K-automorphism: A general framework for privacy preserving network publication. *VLDB*, 2009.

Opening up government data while maintaining data privacy

Caroline Tudor
Office for National Statistics
Segensworth Road
Titchfield, PO15 5RR, UK
+441329444730
caroline.tudor@ons.gov.uk

Philip Lowthian
Office for National Statistics
Segensworth Road
Titchfield, PO15 5RR, UK
+442075928640
philip.lowthian@ons.gov.uk

Keith Spicer
Office for National Statistics
Segensworth Road
Titchfield, PO15 5RR, UK
+441329444983
keith.spicer@ons.gov.uk

ABSTRACT
In this paper, we describe a UK approach to opening up microdata collected by government with examples of actual use-cases of anonymising datasets. We describe briefly the reasoning behind the Open Data movement and the challenges faced in trying to release data openly in practice. Several case studies are provided including that of the Department of Energy and Climate Change (DECC) public use file, and the microdata teaching file from the 2011 UK Census. The anonymisation approach mainly involves detecting quasi-identifier attributes in the data and then modifying the dataset to ensure relative anonymity based on those attributes. This approach is aligned with the principles of k-anonymity. It also involves intruder testing to simulate linking attacks, whereby friendly intruders attempt to attack the dataset and find vulnerabilities to further inform disclosure risk assessment.

Categories and Subject Descriptors
J.1 [**Computer Applications**]: Administrative Data Processing - Government , K.4 [**Computers and Society**]: Public Policy Issues – Privacy.

General Terms
Security

Keywords
Disclosure risk, open data, government data, intruder testing, k-anonymity, linking attack.

1. INTRODUCTION
This paper examines one approach the UK Office for National Statistics (ONS) recommends for opening up government record level data while maintaining data anonymity. Section 2 provides some background to the *Open Data* movement and how this has largely been enabled by technological advances allowing data to be processed and shared far more easily. We also describe what is meant by open data. Section 3 examines the impact of privacy attacks within an open data framework with reference to the *jigsaw (mosaic) effect* perhaps known more widely in the privacy literature as a *linking attack*. In section 4 we put this into the context of government data and set out the value that open datasets have as well as their limitations.

We usefully summarize a general approach to opening up micro-datasets. The approach broadly operates on the principle of *k-anonymity* such that information for any person is hidden amongst k-1 other individuals. Sampling as well as suppression and recoding of variable categories are used to achieve this. We describe *intruder testing* which is used to help inform risk assessment and appropriate selection of *quasi-identifiers*. Our overall goal is to try to achieve k-anonymity with a small k value (e.g. k = 2) to at least remove all uniques, and a weak k-anonymity with a larger k value (e.g. k = 3 or more) for subsets of records which are more vulnerable to attack. This procedure works well to satisfy real-world requirements for a balance of risk-utility. Section 5 describes application of our anonymisation approach to examples of open government datasets and the steps taken to minimize disclosure risk. Section 6 concludes with a short discussion including a summary of the challenges ONS have faced in creating useful open data.

2. OPEN DATA CONTEXT
Web-based technology has allowed increasing numbers of people to share and link data. Information disclosure can now be in digital form: downloadable from the internet and easily processible by computer. In 2008, the Open Data movement undertook to make more data public and accessible, particularly data collected by government, with the argument that this information collected on our behalf should be made freely available to hold government to account. Open data as a concept simply encompasses data that are made available by organizations, businesses and individuals for anyone to access, use and share[1] no matter where they are and what they want to do with the data. Advocates of the open data movement espouse innovative combining of datasets leading to improved citizen engagement and empowerment, being a driver of economic growth and leading to better delivery and efficiency of services. However the UK government, along with other participating countries, faces a number of challenges in order to transition towards open data, one of which is to reconcile the right to information with the right to privacy. While open data must be data that do not relate to an identified or identifiable data subject, achieving this in practice is difficult due to the linking attacks, also known as the jigsaw effect of comparing multiple datasets to eventually reveal disclosive information about one or more identifiable individuals. Two or more datasets each posing negligible disclosure risk in isolation, present an increased risk when the information from these datasets is pieced together in some way.

[1] http://theodi.org/guides/what-open-data

Traditionally UK government release many sensitive datasets at record level under licence; either End User Licence (EUL), Special Licence (SL) or within a safe setting. These have conditions attached which include signing up to a set of conditions on use of the data (EUL) to registration of the user and detailing the purpose of use (SL). There has more recently been a greater emphasis on releasing data with minimum restrictions for the user. As part of the government's commitment to the open data agenda, the UK National Archives developed the Open Government Licence (OGL) which enables and encourages free use of government information. The user is allowed to publish, adapt and combine with other data as long as the information is not *personal data*. Personal data means data relating to a living individual who is or can be identified either from the data or from the data in conjunction with other information that is in, or is likely to come into, the possession of the data controller (UK Data Protection Act, 1988). It is therefore a difficult balancing act between producing open data which are of some use and protected to a reasonable level even when combined with other data sources.

In this paper we discuss some real-world examples of how some government datasets have been made open datasets and thus made available publicly, taking account of privacy considerations, and the resulting limitations on such datasets. We discuss publicly released datasets from the department of Business, Innovation and Skills (BIS), Department for Energy and Climate Change (DECC), the 2011 Census microdata, and also take a look at the licensed survey datasets within the Office for National Statistics and how they were assessed for potential open release.

3. ASSESSING RISK OF LINKING ATTACKS ON OPEN DATA

In order for data to be released under OGL, it is first necessary to reduce the risk of identification. All explicit identifiers such as name and date of birth should be removed from the dataset. Once data have been de-identified so that it is no longer possible to establish links to particular individuals, data may be considered for release openly and used for a wide range of purposes. However there may be a small residual risk that identifiable data could be revealed. Sets of attributes may still be linked with external data to uniquely identify individuals in the population and are called *quasi-identifiers* as defined in [1].

The risk of a linking attack becomes more likely when many similar data are available, to a large number of people. As set out in [2] the risk of jigsaw identification/linking attack with the inclusion of anonymised databases in a transparency programme increases due to three reasons (adapted here in summary):

1. The very concept of open data precludes the possibility of withdrawing access to data if need be.
2. The amount of data on the web grows annually thanks to information on social networking sites and local press coverage.
3. Jigsaw identification is computationally complex. However dramatic increases in computer power have made this easier and complete future-proofing against such disclosure is almost impossible.

Crucially, the responsibility for preventing linking attacks lies with the releasing agency. According to [2], this depends on the nature of the information, the availability of other information, and the technology in place that could facilitate the process of identification. Determining the level of acceptable risk in open data according to these factors is therefore complex.

Privacy risk is regulated at the European level by an EU Directive[2] which states that to determine whether a person is identifiable, account should be taken of all the means likely to be reasonably used either by the controller or by any person to identify the said person. In the UK, the Information Commissioner's Office (ICO) - which is responsible for enforcement of the Data Protection Act (DPA) - released in 2012 its "Anonymisation: managing data practice protection risk code of practice"[3] online. This details how to release anonymised data with the caution that publication under an open government licence is a release to the wider world and carries more risk. The stance from the ICO Anonymisation Code of Practice[4] is for data providers to assess whether it is reasonably likely that an individual can be identified from the data and to consider what other data are available and how and why the data could be linked. It suggests that data providers should establish an auditable process for ensuring an adequate level of anonymisation. One particular assessment that the Code of Practice advocates is a test of whether an intruder might be able to achieve re-identification. This would be done by way of a 'motivated intruder test' as part of a risk assessment. In section 5 we describe how such a practical test provides useful additional information to support risk assessments of datasets, particularly in reference to linking attacks.

4. AN ANONYMISATION APPROACH TO OPENING UP GOVERNMENT DATA

In a government context, a primary purpose of open datasets is for teaching or as training datasets. These allow code to be tested and checked before using it on a more complete dataset released under end-user or special licence. These datasets permit researchers to get a feel for what the data may be like and to allow preliminary hypotheses to be formed. At present, most open government datasets are generally not suitable for research projects other than making initial speculations and for some simple tabulations.

Our anonymisation approach essentially implements variations of the k-anonymity principle as a way of cutting down the detail into a much reduced open dataset. The concept of k-anonymity was first introduced by [3] as a way of preserving privacy. In essence a release of data is said to have the k-anonymity property if the information for each person contained in the release is hidden amongst k-1 other individuals. In practice this means that any quasi-identifier in the released table must appear in at least k records. So if the quasi-identifiers are age and sex, then it will ensure that there are at least k records with 30-year old females for example. This property can be achieved by generalization and suppression. Generalization refers to publishing more general values which can be done by recategorizing age into bands for

[2] http://ec.europa.eu/justice/policies/privacy/docs/wpdocs/2007/wp136_en.pdf

[3] https://ico.org.uk/for_organisations/guidance_index/~/media/documents/library/Data_Protection/Practical_application/anonymisation-codev2.pdf

[4] https://ico.org.uk/for_organisations/data_protection/topic_guides/anonymisation

example. Suppression can be done by removing outliers or by providing only a sample of records. Sampling has the benefit that an intruder can never be sure whether a person is contained within the dataset.

In the context of real-world data, using algorithms to find optimal anonymous tables can be unpractical and even for optimal solutions, the distortion of the data can be too high leading to un-useful tables ([4]). The balance of risk-utility is absolutely crucial in practice for government data releases. [4] discusses an alternative known as *weak k-anonymity* which requires it to be enforced in just a subset of the records. This of course means that there is a possibility that those variables which are not controlled via k-anonymity can be used to identify someone. However this risk is generally small when interest in these variables is low and so is typically a more practical option. Our approach aims for k-anonymity enforced on the entire dataset with a small k value e.g. k=2 to remove all uniques as a minimum; and a *weak* k-anonymity (on a subset of records) with a larger k e.g. k = 3 or more, depending on the dataset and its particular vulnerabilities.

K-anonymity involves consideration of sets of attributes that can be linked with external information to re-identify the respondents to whom the information refers. As in [5] a data release is said to satisfy k-anonymity if every combination of values of quasi-identifiers can be indistinctly matched to at least k individuals. This information can be known only by linking the released data with externally available data. We make use of intruder testing to help refine the appropriate set of quasi-identifiers.

The general procedure that ONS advise here for creating open datasets is outlined as follows:

i. Assess dataset background
ii. Choose the key variables that might be used for identification (key variables described in [6])
iii. Consider how a dataset might be disclosive under intruder scenarios (see [7])
iv. Analyse variable combinations / quasi-identifiers ['uniques' analysis or similar]
v. Carry out a formal intruder test and refine steps above
vi. Generalize (recode variables) and/or suppress (reduce dataset/sample) to make suitable for public release to achieve properties of k-anonymity

(i) Dataset Background

Disclosure risk assessment should commence by talking to potential users of the data to understand the types of research that the data will be used for. This step is sometimes ignored, to the detriment of the final dataset. Main considerations might be the variables in which they are most interested and the level of detail that is needed, and particularly the required level of geography. It is also important to understand whether the original dataset is a survey sample, an administrative dataset or a census. General consideration must be afforded to any existing protection in the dataset due to an intruder's uncertainty as to whether an individual is actually present in the data.

(ii) Assess disclosure under intruder scenarios

Having done this preparatory knowledge gathering, the next stage is to consider how a dataset could be shown to be disclosive by looking at a number of intruder scenarios. An intruder (or attacker) is somebody who attempts to discover personal information about an individual, household or business in the dataset. This is most likely to occur if the intruder has some initial knowledge about a particular member of the dataset with respect to a number of variables known as key. For example an individual in the data could be a relative, neighbour or work colleague These intruder scenarios include combining the dataset with other data sources.

(iii) Analyse variable combinations

Based on the intruder scenarios that fit with a particular dataset, the procedure is then to select a set of key variables (five or six for each dataset) to form an *identification key* and tabulate combinations (also referred to as quasi-identifiers in the privacy literature) to create a series of two, three and four dimension tables. These combinations should be plausible i.e. likely to be similar to tables required by researchers.

In general, knowledge of the data should lead to a suitable range of combinations being selected. It should be noted that creating tables with a large number of variables will be counterproductive as patterns may emerge that would not be noticed by a researcher. Most records are unique if a large number of variables are combined. Instead, we should consider just a limited set of variables, the values of which are what an intruder is likely to know. These are the aforementioned identification key, the variables likely to be used by an intruder to identify the individual and then discover the rest of the information in the remaining variables.

Variable combinations which are rare or unique will indicate potential disclosure issues and variables will need to be recoded or excluded if required. This approach of looking for rare combinations of variables is similar to the more formal k-anonymity method described in [3] and [8].

(iv) Carry out a formal intruder test

This involves using "friendly intruders" to try and see if they can re-identify anyone in the dataset. These friendly intruders should have some background knowledge of the data (as a data user might) but should not be specialist hackers using advanced techniques. This is in consideration of the phrase "means likely and reasonably" as referenced in the EU directive and UK Data Protection Act. The intruder motives would not be malicious. They would not release their findings into the public domain but would feed back their finding to aid in the publication of a secure dataset. One of the main purposes of such a test is to try and capture what other information may be linked to the dataset by the intruder to attempt disclosure. Thus appropriate selection of intruders in terms of awareness of similar data sources and good penetration skills (able to search and analyze the data) are important to get accurate results. The information resulting from the intruder test may be used to refine the previous steps, particularly with regards to which variable combinations are considered to be quasi-identifiers and therefore utilized in the identification key. For a more detailed discussion of intruder testing, please see [9].

(v) Generalize and Suppress

The last stage of the process involves taking steps to minimize overall risk in the dataset. Our approach is to ensure that k-anonymity is achieved to a low k value, i.e. at least k =2 to ensure no uniques (or k = 3 to eliminate pairs) in the data. This is usually achieved by sampling. The next stage is to recode variable categories to reduce detail (generalization) so that weak k-

anonymity is achieved to a higher k value for a subset of the data where there are particular vulnerabilities.

5. EXAMPLES OF MAKING DATA OPEN

In this section we consider how the general procedure outlined in section 4 is applied in practice to three examples of open datasets produced by the UK government. One of these is a sample of microdata from the UK 2011 Census collected by the Office for National Statistics (ONS) while the second is a sample from an administrative dataset produced by the Department for Energy and Climate Change (DECC) on domestic gas and electricity consumption. The steps followed in order to produce these datasets are shown below. There are many similarities in producing these datasets but some important differences. There were fewer variables that could easily be recoded in the DECC data leading to less flexibility in reshaping the data. The DECC data is typical of a lot of datasets which contain a lot of information but not a lot of variables which can be recoded in a straightforward way. Most variables are dataset specific and any recategorization would reduce the utility significantly. An ongoing project to publish education data held by the Department for Business, Innovation and Skills (BIS) is also discussed briefly as our third example.

In all three cases, statistical disclosure control has to be applied to ensure that sufficient protection is given to avoid an individual, household, business or other statistical unit being re-identified. As detailed in section 4, data might be recoded and/or only a limited number of variables released. For a more general discussion of statistical disclosure control techniques that might be applied during this process, the reader is referred to [10].

5.1 2011 UK Census microdata

The 2011 UK Census is a rich data source with many published tabular outputs available from the ONS website. In addition to these tables, record level data are being made available to researchers; a teaching dataset at individual level was published in 2014. The data can be accessed through the link below. Note that more detailed datasets will shortly be available under more prohibitive licensing and access conditions. This dataset is a random 1% sample of records for England and Wales published to encourage a wider use of census data and as an introduction to these more detailed datasets.

http://www.ons.gov.uk/ons/rel/census/2011-census/2011-census-teaching-file/index.html

The broad approach was to use both sampling and suppression of variables to remove uniques/pairs and achieve k-anonymity to a k of at least 2 in the published database, and then recoding to further achieve weak k-anonymity to a larger k based on the most identifying variables which were age, ethnic group, industry, economic activity and religion.

Producing the Census Microdata teaching file

A small sample size is used in order to aid protection (among other factors such as imputation for non-response), since a potential identification might be uncertain because the intruder will have doubt as to who is in the sample (and who is not). An intruder may find a record which corresponds to an individual for whom they are searching, possibly somebody unique in the sample with respect to specific visible characteristics. However they cannot be certain as to whether this sample unique is the person they are attempting to find because of the small sample size. The individual who is unique in the sample will not necessarily be the person they are looking for and there is no certainty that they would be unique in the population.

Starting from a large dataset – containing most variables and some further derived variables, with all the standard categories – it would clearly be possible to identify an individual, either directly or indirectly from these data. Hence some work was necessary to create a dataset suitable for 'open' data and public release:

- Remove all direct personal identifiers such as Name, Address and Date of Birth. The released file will have to contain no information allowing identification of an individual or household so this is the initial step in producing the data
- Decide on the variables to include in the data. Only a subset of census variables should be present in this teaching dataset, including basic demographic information and those variables used in the most popular tables. The level of Geography is to be Region (9 Regions for England plus Wales). Other variables include Sex, Ethnic group, Country of Birth, Industry, Marital Status and Household composition.
- Identify the key variables. These are the variables (usually in combination) which are most likely to assist an intruder to identify an individual in the data. These variables are usually those that are in the public domain such as Sex, Age, Ethnic group or those which a friend, relative or work colleague might know such as Occupation, Marital Status, Hours worked/week along with more sensitive variables such as Health.
- Create tables from the 1% sample using combinations of the key variables. Any low counts could lead to an individual in the data being identified. Note that this is a sample so there will be considerable doubt if a unique combination of variables in the sample is equivalent to a unique combination in the population.
- Create the same tables from the population data. Look for unique or rare combinations
- The most identifying variables were found to be
  - Geography
  - Sex
  - Age
  - Ethnic group
  - Industry
  - Economic Activity
  - Religion
  - Country of Birth
- Recode some of these variables to protect the data.
  - Recode Age into 8 Categories
  - Recode Ethnic group from 16 to 5 categories
  - Recode Industry from 17 to 12 categories
  - Recode Economic Activity from 13 to 9 categories
  - Recode Religion from 10 to 8 categories
- Recreate the tables from earlier using the recoded variables. The results show many combinations with sample uniques but very few with population uniques.
- Swap a small number of records (include these population uniques along with other records) between Region.

- Intruder testing was used as confirmation that risk was reduced to an acceptable level based on the number of correct identifications (if any).
- Publish the Data as an open data microdata file

5.2 DECC – ENERGY DATA

DECC has published two datasets from the National Energy Efficiency Data Framework (NEED) One of these is a Public Use File (open data) to be discussed here (49,815 records). The other is a file released under End User Licence (4,086,448 records). Both datasets are based on samples of properties which have been assessed for an energy performance certificate (EPC). Variables relating to the property are included along with gas and electricity consumption values.

The same methodology was applied in producing these datasets. A link to the Public Use File is shown here.

https://www.gov.uk/government/publications/national-energy-efficiency-data-framework-need-anonymised-data-2014

In this example the broad approach was again to use sampling and suppression of variables to achieve k-anonymity across the entire published dataset with a low k value. Intruder testing is used to confirm which variables might be used for identification and the "Year of EPC Assessment" variable subsequently removed. Recoding of variables was also applied to achieve weak k-anonymity with a higher k, for a subset of the most identifying variables.

The process was as follows for the production of the open data microdata file.

- A consultation period with potential users of the data was set up. This gave an indication of the level of detail users expected in the output data.
- Direct identifiers and detailed geographical indicators were removed from the data.
- The most visible variables were selected as key variables. These are the variables most likely to be used by intruders in attempts to identify a property. These variables are shown below along with plausible intruder scenarios.
    - Property Type (for example detached or end terrace). This would be obvious to anybody walking past the property in many cases, although there could be some doubt. For example is a house a single property or has it been divided into flats?
    - Property Age. An estimate of this can be made, although it may not be correct. Specialist property knowledge could be required for an accurate estimate. If the exact date of construction was known and the variable published at this level of detail it would provide an ideal starting point for an intruder.
    - Floor area. The floor area band would not be easy to estimate from outside. A visitor to the property would have a much better idea of this value, although even then a correct estimate may not be easy.
    - Geography. At a lower level of geography there will be fewer properties thus making a correct identification more likely. This is to be taken into consideration when deciding whether to release the data at National, Region or Local Authority levels.
- Look at distributions of the visible variables both individually and in combination. Are there low counts at National, Region and LA levels? If a property can be identified as belonging to a particular combination, much additional detail including the approximate gas and/or electricity consumption could be determined. If combinations of these variables produce low counts then certain variables may require recoding. The response variable of major interest is gas / electricity consumption. Low counts in the bands would give some information about the property but possibly not too much. Look out for values at the top and bottom of the range which are highlighted in the consumption data. In combination with the visible variables they could require protection.
- As a result of this analysis both Property Age and Floor area size are recoded into a smaller number of categories to reduce the number of low cell counts. It was also decided that the data would be released at Region level and not Local Authority level.
- The actual gas and electricity consumption values are given additional protection by being rounded to the nearest multiple of 5. This ensures that the actual value is not released in the dataset.
- A small number of records were swapped between Regions.
- Intruder testing was carried out by post graduate students at Southampton University. A cash prize was offered for a correct identification. There were no correct identifications but as the 'year of the energy performance certificate' was considered to be of particular use by the 'intruders' this variable was removed from the published open dataset, although it remains in the End User Licence data.
- Publish the Data as an open data microdata file.

5.3 BIS – FURTHER EDUCATION DATA

The Department of Business, Innovation and Skills (BIS) is planning to publish an open dataset of Further Education Learning aims, Providers and outcomes. This is a large dataset with many millions of records. It was hoped that a number of 'essential' variables would be included in the published data. These include variables relating to the type of course and an outcome grade variable.

In this example we achieve k-anonymity for a low k by suppressing variables non-essential to the user, as well as weak k-anonymity for a higher k by removing entire records within certain regions.

The process was as follows:

- From the list of essential variables decide on which are most visible and therefore key variables.
  Age group (3 categories)
  Sex

Learning aim (equivalent to a detailed course description)
Delivery Provider (a college or a company)

- The Region in which the learning took place was used as a geography variable.
- Tables of combinations of the key variables resulted in many unique combinations. The data could not be published in the current form. There was a requirement that the learning aim variable was retained and the following approach was followed.
- Age group to be recoded into 2 groups. Sex was dropped from the dataset.
- Records with a Learning Aim with fewer than a pre-defined number of enrolments within a Region were excluded from the data.
- Records which were unique with respect to Age group, Provider and Learning Aim within a Region were removed from the data.
- Data are currently in the process of being distributed for intruder testing before final release as an open dataset.

5.4 Should all licensed data (EUL) instead be released under OGL?

Currently many outputs from the UK Data Service[5] are released under a more restrictive End User Licence (EUL). The EUL is a 'light touch' licence with users promising not to attempt disclosure and to ensure that any outputs passed on do not compromise the confidentiality of individuals. Users of the EUL should keep the data confidential and not attempt to identify organizations, individuals or households in the data. In practice these datasets are designed so that the possibility of disclosure is remote. On this basis, the ONS recently conducted an intruder testing exercise to see whether the EUL was too conservative and whether these data could potentially be released under OGL. The Labour Force Survey and Living Costs and Food Survey microdata were used as two example datasets for assessment (see [11]). These were interesting cases as the intruder testing assumed the intruder had *response knowledge* of who was in the sample.

The disclosure scenario of response knowledge was considered a reasonable possibility under an OGL since the data would then be available to a much wider audience who would not be signing to a set of agreed conditions, unlike with the EUL. It was subsequently found from the intruder testing that re-identification was possible for certain individuals. Response knowledge meant that intruders would have much wider knowledge of individual attributes beyond the limited set of quasi-identifiers used to achieve k-anonymity under traditional EUL intruder scenarios. The conclusion from this was therefore that the conditions of an OGL mean extra precaution should be taken with releasing government data and to make careful assessments on a case-by-case basis. Significantly reducing detail by limiting the number of variables and their categorical breakdown is paramount to reducing the additional risk that comes with releasing open data.

[5] http://ukdataservice.ac.uk/

6. SUMMARY AND DISCUSSION

This paper has discussed the approach the ONS has taken towards opening up government data. Broadly speaking, suppression of variables and sampling (in two of the examples) are used to guarantee k-anonymity for a low k value (generally to remove uniques as a minimum). As a second stage, recoding/recategorisation of variables is used to generalize the dataset so that weak k-anonymity is achieved for a higher k value for a subset of records that are more likely to be attacked. Intruder testing is used to help inform the process in consideration of external information that might be linked to the dataset. The three examples discussed demonstrate the limited amount of information that can be made available openly. The purpose of these open datasets is usually only for use as teaching or training datasets. We briefly discussed how more detailed datasets available under End User Licence were not suitable for open release in their current form.

We have shown with our examples the difficult balancing act between producing Open Data which are of some use and protected to a reasonable level so that they remain non-personal data. Increases in technology in the past ten to fifteen years have changed the data environment beyond recognition. The consideration of other publicly available data sources is virtually impossible with the continual addition of data on the web. Intruder testing goes some way towards testing this in a practical way but is dependent on using knowledgeable and skilled intruders. As mentioned in the ICO anonymisation code of practice, this should be carried out periodically as the risk of re-identification may change with time bearing in mind likely increases in computing power and as the public availability of data increases. Feedback on intruder testing has been mostly positive and some of the benefits of this approach are outlined in [9]. Benefits include learning which variables and which types of individuals might be vulnerable to attack, and perceptions of disclosure. These provide a practical feel for data controllers of the level of risk. Further work would be helpful in developing expertise further in undertaking intruder testing, including working towards establishing reasonable standards and guidance. These would include the methodology employed, the length of time reasonable for an intruder to attempt disclosures, the level of 'uncertainty' that is reasonable, and better advice on the use of external information. However the importance of theoretical and sound practical risk measures should not be forgotten since the use of intruder testing is very much a snapshot of risk specific to each intruder and the parts of the data they are given (e.g. intruders might only assess particular geographies local to them and which they are knowledgeable about).

It also follows from this that there is a need to establish how much effort is "reasonable", (as mentioned in the ICO code) and where to set the line of acceptable risk. What values of 'k' are acceptable in open data? Can we measure units of anonymity to help data controllers make a decision? There is a clear link here to the concept of differential privacy – how much extra we can learn from an individual being included in a database as opposed to not including them. The purpose of a statistical office is both to collect and disseminate statistical information that will aid policy and research and, generally, be for the 'public good'. Hence it is an unreasonable, and usually unattainable goal to aim for only releasing datasets that are zero risk or in these examples to obligate strong k-anonymity. Ultimately, there is a legal interpretation – what risks would it be reasonable to protect against, so that the data publisher has a defence.

The future of anonymisation is unclear given the ever increasing amount of information being made publicly available. Open datasets only add to the disclosure risk. Currently these data generally have poor utility for answering complex research questions. An alternative we mention very briefly here is the potential use of synthetic or modelled data in an attempt to move towards a much richer set of data retaining at least all primary properties of interest to the researcher. One may argue that synthetic data have little or no risk as they do not represent the original data. However the creation of synthetic datasets that are truly representative of the population is very much an art. There is a related cost-benefit argument to whether the idea of open data is sustainable given the amount of effort government agencies need to produce such datasets. It is also important to remember that due to lack of research value, many open government data are still released alongside other licensed datasets (as is the case with the DECC data for example) Work for the future is not only about how to make open data as useful as possible but must also address all associated wider issues: data privacy but also technical, legal, economic and policy issues.

7. ACKNOWLEDGMENTS

The authors would like to thank Mary Gregory from DECC and Johanna Hutchinson from BIS for allowing us to reference their datasets. The authors are also grateful for the reviewers' comments which helped to improve this paper.

8. REFERENCES

[1] Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M., *L*-diversity: Privacy beyond *k*-anonymity, ACM Transactions on Knowledge Discovery from Data (TKDD), v.1 n.1, p.3-es, March 2007

[2] O'Hara, K, Whitley, E and Whittall, P (2011) Avoiding the Jigsaw Effect: Experiences With Ministry of Justice Reoffending Data.(unpublished briefing paper) http://eprints.lse.ac.uk/45214/

[3] Sweeney, A.L.. K-anonymity: a model for protecting privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10 (5), 2002; 557-570

[4] Atzori, D.M.. Weak k-Anonymity: A Low-Distortion Model for Protecting Privacy. Information Security Lecture Notes in Computer Science Volume 4176, 2006, pp 60-71

[5] Samarati P., (2001) Protecting Respodents' Identities in Microdata Release, "IEEE Trans. Knowl. Data Eng., no 6, 1010-1027.

[6] Elliot, M.J., and Dale, A. Disclosure risk for microdata: Workpackage DM1.1 What is a key variable? *Report to the European Union ESP/204 62/DG III , 1998*

[7] Elliot, M. J., and Dale, A. Scenarios of attack: The data intruder's perspective on statistical disclosure risk. *Netherlands Official Statistics. Vol 14, Spring 1999, 6-10.*

[8] Sweeney, B. L.. Achieving k-anonymity privacy protection using generalization and suppression. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10 (5), 2002; 571-588

[9] Tudor, C, Cornish, G, and Spicer, K. Intruder Testing on the 2011 UK Census: Providing Practical Evidence for Disclosure Protection. *Journal of Privacy and Confidentiality*: Vol. 5: Iss. 2, Article 3, 2014
Available at: http://repository.cmu.edu/jpc/vol5/iss2/3

[10] Hundepool A., Domingo-Ferrer J., Franconi L., Giessing S., Schulte Nordholt E., Spicer K., de Wolf P., Statistical Disclosure Control; Wiley (2012)

[11] Elliot,M. Mackey, E., O'Shea, S., Tudor, C., Spicer K., EUL to OGD: A Simulated Attack on Two Social Survey Datasets. *CD – only proceedings of Privacy in Statistical Databases, International Conference*, Ibiza, Spain, September 17-19, 2014.

Private Computation of the Longest Increasing Subsequence in Data Streams

Luca Bonomi
Dept. of Mathematics and Computer Science
Emory University
Atlanta, GA
lbonomi@emory.edu

Li Xiong
Dept. of Mathematics and Computer Science
Emory University
Atlanta, GA
lxiong@emory.edu

ABSTRACT

In this paper, we study the problem of privately computing ordered statistics with the goal of monitoring sequential data streams. Despite the broad series of techniques for time-series monitoring, only few works provide provable privacy guarantees employing the formal notion of differential privacy. While these solutions are well established, their focus is mostly limited to count based statistics (e.g. number of distinct elements, heavy hitters). In this paper, we consider a more general problem of privately computing the length of the longest increasing subsequence (LIS) in the data stream model. This important statistic can be used to detect trends in time-series data (e.g. finance) and perform approximate string matching in computational biology domains. Our proposed approaches employ the differential privacy notion which provides strong and provable privacy guarantees. Our solutions estimate the length of the LIS using block decomposition and local approximation techniques. We provide a rigorous analysis to bound the approximation error of our algorithms in terms of privacy level and length of the stream.

1. INTRODUCTION

Sequential data are central in a broad range of domains and applications, such as biomedical, financial and health-care setting where data are continuously collected for monitoring purpose or for mining behavioral patterns. For example, individual household power consumption data may be collected by smart meters to provide billing information or for monitoring purpose. Despite the importance of these tasks, the release of the real data value may disclose sensitive user information. Therefore privacy preserving solutions are employed to compute the required statistics while providing privacy for users. Among them, the popular notion of *differential privacy* [6] is used to construct privacy preserving algorithms. The privacy is achieved by bounding the adversary inference ability in determining the presence of any event in the data stream [7, 10, 9, 4]. Despite the strength of such a privacy model, the current solutions are limited to count based statistics.

In this paper, we study the problem of privately computing ordered statistics with the goal of enabling applications to monitor sequential data streams. Consider for example, a user who may wish to be advised in his/her financial decisions without incurring the risk of disclosing his/her financial information. In such a setting, it is crucial to design effective solutions that enable third-party to detect user's financial trends while preserving the sensitive information. To address this problem, we propose to study the privacy preserving computation of *longest increasing subsequence* (LIS) in the stream model.

The computation of the LIS provides useful information about the sortedness of the data stream and it can be used to detect trends in time-series data. In general, the task of computing the sortedness of a data stream is receiving considerable attention from the computer science community [16, 12, 1, 13, 5, 19, 11]. The sortedness of data stream has important implications from both practical and theoretical perspectives. Many applications rely on ranked data and the massive amount of information dynamically generated cannot be processed in an off-line fashion requiring solutions to have small update time and memory requirements.

The computation of the LIS in the data stream model rises new challenges compared to the traditional privacy setting. First of all, privacy requirements in protecting sensitive information for this ordered statistic have a greater impact on the final utility. Count based statistic over a stream are typically computed by decomposition which leads to a considerable reduction of perturbation noise required by the privacy mechanism. However, ordered statistics are generally not easy to approximate via decomposition since they require a global view of the entire stream. Second, the LIS has higher memory requirements compared to standard counting based statistics (e.g. counts, heavy hitters). In fact, it has been shown in [12] that there exists a space lower bound of $\Omega(T)$ for any randomized algorithm that computes the LIS exactly over a stream of length T. This strong separation between count based functions and LIS impacts both efficiency and utility of the solutions for this problem.

To address these challenges, we propose a series of solutions for privately computing the LIS while minimizing the error introduced by the perturbation and approximation. The detailed contributions are reported below.

Our Contributions. In this paper, we study the problem of privately computing the LIS in a time-bounded stream of length T. 1) Our proposed solutions compute the length of the LIS providing strong and provable privacy guarantees based on the notion of differential privacy. 2) We propose a decomposition framework for approximating the length of the LIS using local information in the stream. This technique allows us to reduce the error due to perturbation noise from the privacy mechanism. Using the Patience Sorting algorithm [15] as a black box for locally computing the exact length of the LIS, we provide an error bound for our framework. 3) We conduct an output-sensitive utility analysis for two cases based on the length of the output LIS. In particular, we as-

Algorithm 1 Patience Sorting

1: **procedure** PATIENCE SORTING(σ)
   Input: event stream σ
   Output: $LIS(\sigma)$ length of the longest increasing subsequence
2: $P(j) \leftarrow \emptyset$ for $j = 0, 1, \ldots, m-1$
3: **for** (any new element $\sigma(i)$) **do**
4: Find the largest $P(j)$ such that $P(j) \leq \sigma(i)$
5: $P(j+1) = \sigma(i)$
6: **Output** the largest j such that $P(j) \neq \emptyset$
7: **end for**
8: **end procedure**

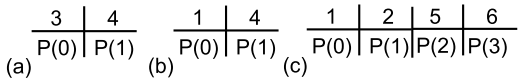


Figure 1: Running example of the Patience Sorting algorithm over the stream $\sigma = 3, 4, 1, 2, 5, 7, 6$.

sume $LIS(\sigma) = \sqrt{T}/\beta$, where T is the length of the input stream σ, and β is a parameter in the range $[1/\sqrt{T}, \sqrt{T}]$. For each solution, we bound the approximation error in the case of long and short LIS respectively depending on the value of β. 4) We propose a new streaming approach which computes the LIS using a hierarchy structure of the stream. Our algorithm achieves a $(1 - \frac{T-b}{T+b})$-approximation to the length of the LIS in the worst case, where the parameter b controls both the perturbation noise to achieve the desired level of privacy and the accuracy. 5) We provide a discussion about possible extensions of our solutions to address time-series stream monitoring and string matching problems. To the best of our knowledge, we are the first to investigate the problem of privately computing the longest increasing subsequence.

The rest of the paper is organized as follows. Section 2 provides the problem definition and presents the privacy model. Section 3 illustrates our decomposition schema and Section 4 describes our hierarchy solution. In Section 5, we provide a summary of our results and also describe some possible extensions. Finally, Section 6 concludes the paper.

2. PRELIMINARIES

Given a sequence σ of elements $\sigma(i) = a_i$ defined over a finite alphabet $\Sigma = \{0, 1, \ldots, N-1\}$, an increasing sequence of length k in σ is a subsequence $\{i_0, i_1, \ldots, i_{k-1}\}$ such that $i_0 < i_1 < \cdots < i_{k-1}$ and $a_{i_0} \leq a_{i_1} \leq \cdots \leq a_{i_{k-1}}$. Furthermore, let $\sigma[i, j]$ denote the contiguous sequence $\sigma(i)\sigma(i+1)\ldots\sigma(j)$ in the stream σ, and let $LIS(\sigma)$ be the length of the longest increasing subsequence in σ.

The problem of computing the LIS has received much attention in the streaming setting (see [2] for a survey of results), where the sequence σ is given an element at a time. In such model, data arrive continuously and at every time i algorithmic solutions are required to report $LIS(\sigma[0, i])$ by using a small amount of memory and performing only few passes over the stream. In the rest of the section, we briefly summarize the non-private techniques present in literature by categorizing them as *exact* and *approximate* solutions.

Exact Solution. The study of LIS in the streaming setting was initiated by Liben-Nowell et al. in [16], where the authors developed an exact one pass algorithm that requires $O(k)$ space for deciding if the length of longest increasing subsequence is at least k. In addition to this technique, the classical algorithm for computing the LIS is based on the Patience Sorting procedure [15]. This approach can be interpreted as a one pass streaming algorithm for computing the exact LIS in $O(T)$ space and it requires $O(\log LIS(\sigma))$ update time. Since we use this approach to build our solutions, we briefly describe this algorithm here.

In the Patience Sorting procedure, the length of the longest increasing subsequence is computed using a set of sorted *piles* $P(0) < P(1) < \cdots < P(m)$ each storing an element of the stream σ. For any new element $\sigma(i)$ that appears in the stream, the algorithm places $\sigma(i)$ in the leftmost pile $P(j)$ such that $P(j) > \sigma(i)$. The number of non empty piles represents the length of the LIS at any time point. An overview of the Patience Sorting algorithm is illustrated in Algorithm 1. Below, we describe a running example of this algorithm.

EXAMPLE 1. *Let $\sigma = 3, 4, 1, 2, 5, 7, 6$ be a stream in input. The algorithm starts with a set of empty piles $P(j)$ for $j = 0, \ldots, m-1$. When the first element arrives in the stream it is placed in the first pile $P(0)$. After the arrival of the second element, the situation in the piles is illustrated in Figure 1 (a). The number of piles denotes the length of the longest increasing subsequence at each time. Therefore, in this case the length of the LIS is two. When the third element $\sigma(2) = 1$ arrives in the stream, the algorithm places this element in $P(0)$, as shown in Figure 1 (b). Following the steps of the algorithm, the final set of piles is reported in Figure 1 (c). At the end of the stream the length of the longest increasing subsequence is four.*

Despite the simplicity of this procedure, the Patience Sorting algorithm is optimal from the space complexity perspective. In fact, Gopalan et al. [12] showed a space lower bound of $\Omega(n)$ for any randomized algorithm that computes the LIS exactly.

Approximate Solution. In [12] the authors proposed a $(1+\epsilon)$-approximation for the LIS computation using $O(\sqrt{T/\epsilon})$ space. A series of works have been developed to estimate the length of the LIS using the number of inverted elements in the stream. In this direction, Ajtai et al. [1] proposed a $(1+\epsilon)$-approximation which requires $O(\frac{1}{\epsilon}\log\log T)$ space to estimate the number of inverted pairs. Later this result has been improved by Gupta and Zane [13]. Cormode et al. [5] proposed a series of algorithmic solutions based on distance preserving embeddings. Recently in [19], the authors investigated the problem of computing the LIS in asymmetric edit distance setting.

2.1 Differential Privacy

Differential privacy [6] is a recent notion of privacy that aims to protect the disclosure of information when data statistics are released. In the streaming setting, due to the dynamics of the data, the classical differential privacy notion has been redefined such that the privacy is guaranteed at *event-level* [7, 10, 9, 4]. In other words, the privacy goal is to protect the presence or absence of any single event in the stream. The formal definition of the differential privacy notion adopted in our work is reported below.

DEFINITION 1 (DIFFERENTIAL PRIVACY [4, 9]). *Two streams σ and σ' of the same length are neighboring streams if they differ exactly in one element at time t. A privacy mechanism M gives α-differential privacy if for any two neighboring streams σ, σ', and for any set of outcomes $S \subseteq Range(M)$,*

$$Pr[M(\sigma) \in S] \leq e^{\alpha} \times Pr[M(\sigma') \in S] \quad (1)$$

The parameter α is called the *privacy parameter* and it defines the privacy level of the mechanism. Higher values of α lead to lower level of privacy, while smaller values pose a stronger privacy guarantee. Intuitively, a mechanism is differentially private if an adversary is unable to determine whether an event of interest took place or not by observing the output of the mechanism over the stream.

Our goal consists in designing a mechanism that, at any time t in the stream, reports the length of the longest increasing subsequence while achieving differential privacy. In addition, we would like the mechanism to be useful, that is, its output well approximates the real length of the LIS. To evaluate our solutions, we introduce the following utility notion.

DEFINITION 2 ((ϵ, δ)-USEFUL). *A streaming algorithm $\mathcal{A}$ is (ϵ, δ)-useful, if for any input stream σ and query q, with probability at least $1-\delta$, the relative distance between the approximate answer from $\mathcal{A}$ and the real answer of q is within ϵ, formally* $P\left[\frac{\|\mathcal{A}(\sigma)-q(\sigma)\|}{q(\sigma)} < \epsilon\right] \geq 1-\delta$

To achieve differential privacy, one well established technique is the *Laplace Mechanism* [8]. Dwork et al. [8] showed that to obtain a α-differentially private solution it is sufficient to perturb the real output of the function by adding a random variable (noise) from a Laplace distribution with probability density function $pdf(x|\lambda) = \frac{1}{2\lambda}e^{-|x|/\lambda}$, where the parameter λ is determined by α and the *sensitivity* of the function to compute. The sensitivity measures the contribution of any single element on the final output. We will use the Laplace mechanism and some other statistical tools to design our privacy preserving solutions.

Statistical tools. In our approaches, we make use of the *Laplace Mechanism* to achieve differential privacy and *sequential composition* property of differential privacy. Furthermore, in our construction the noise may not come from a single Laplace distribution, but rather is composed by a sum of independent Laplace distributions. Therefore, here we state two useful results for sum of independent Laplace distributions.

THEOREM 1 (LAPLACE MECHANISM [8]). *For a function $f : D^T \rightarrow \mathbb{R}^d$, let $GS(f)$ be the sensitivity of f defined as*

$$GS(f) = \max_{D,D'} \|f(D) - f(D')\|_1 \quad (2)$$

where D' and D are neighbouring, then the algorithm that outputs: $\tilde{f}(D) = f(D) + Lap(GS(f)/\alpha)^d$ satisfies α-differential privacy.

THEOREM 2 (SEQUENTIAL COMPOSITION [17]). *Let M_i be a non-interactive privacy mechanism which provides α_i-differential privacy. Then, a sequence of $M_i(D)$ over the database D provides $(\sum_i \alpha_i)$-differential privacy.*

LEMMA 1 (SUM OF LAPLACE DISTRIBUTIONS [4]). *Let $Y = \sum_{i=1}^{n} l_i$ be the sum of $l_1, \ldots, l_n$ independent Laplace random variables with zero mean and parameter b_i for $i = 1, \ldots, n$, and $b_{max} = \max\{b_i\}$. Let $\nu \geq \sqrt{\sum_{i=1}^{n} b_i^2}$, and $0 < \lambda < \frac{2\nu^2}{b_{max}}$. Then $Pr[Y > \lambda] \leq exp\{-\frac{\lambda^2}{8\nu^2}\}$*

COROLLARY 1 (MEASURE CONCENTRATION [4]). *Let Y, $\{b_i\}_i$, λ and b_{max} defined as in Lemma 1. Suppose $0 < \delta < 1$ and $\nu > \max\{\sqrt{\sum_i b_i^2}, b_{max}\sqrt{2\ln\frac{2}{\delta}}\}$. Then $Pr[|Y| > \nu\sqrt{8\ln\frac{2}{\delta}}] \leq \delta$*

2.2 Differentially Private Computation of the LIS - A Baseline Approach

In the rest of the paper, we present our solutions for computing the length of the LIS in the stream. Our approaches require the stream to be *time-bounded*, we assume in fact that the length of the stream is T and it is given a priori.

Here we consider a baseline approach that solves the problem of privately computing the length LIS by perturbing directly its real value at every time point. In particular, for every new element $\sigma(i) = a_i$ in the stream, the algorithm first computes the real $LIS(\sigma[0,i])$ (e.g. using any non-private solution, Patience Sorting in this case) and then it adds a perturbation noise η_i. Given the privacy parameter α, due to the composition property of differential privacy, to obtain an overall mechanism of α-differential privacy, the baseline approach applies the Laplace mechanism at each time point with parameter $\alpha' = \alpha/T$. For each new incoming element, it samples a Laplace variable $\eta_i \sim Lap(1/\alpha')$ which will be used to perturb the real value of $LIS(\sigma[0,i])$. Therefore, at every time i, the algorithm will answer the LIS query by returning $\tilde{l}(\sigma[0,i]) = LIS(\sigma[0,i]) + \eta_i$. We can observe that the sensitivity for the LIS function is 1, since replacing an element from the stream may change the length of the longest increasing subsequence by at most 1. Therefore, perturbing the real value of $LIS(\sigma[0,i])$ with η_i is sufficient to achieve privacy. The utility of this approach is reported in the following theorem.

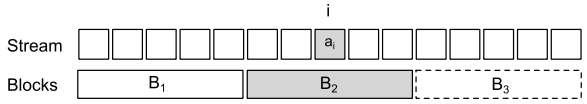


Figure 2: Block Decomposition example at time i: expired blocks (solid lines), active blocks (gray) and the future blocks (dashed lines).

THEOREM 3 (BASELINE UTILITY). *The baseline algorithm is $(\frac{\beta\sqrt{T}}{\alpha}\ln\frac{1}{\delta}, \delta)$-useful for computing the longest increasing subsequence.*

PROOF. *The released length of the LIS at each time i is obtained by perturbing the real length of the LIS with Laplace noise. Therefore, at every time step in the stream we have that the additive error from the noise can be bounded as follows:*

$$Pr[|\eta_i| > \gamma] \leq 2\int_{\gamma}^{\infty} \frac{\alpha}{2T} e^{-x\alpha/T} dx = e^{-\gamma\alpha/T} \quad (3)$$

Hence, with probability at most δ the additive error is at least $\frac{T}{\alpha}\ln\frac{1}{\delta}$. The final result follows by normalizing the error by the $LIS(\sigma) = \sqrt{T}/\beta$. $\square$

Space and Time Analysis. The memory and time complexity for this approach are the same as the non-private algorithm used to compute the real length of the LIS. Therefore, using the Patience Sorting algorithm for example, the space and update time required are $O(LIS(\sigma))$ and $O(\log LIS(\sigma))$ respectively.

3. DECOMPOSITION FRAMEWORK

The baseline approach introduces an additive error that grows linearly with the length of the stream. Therefore, for small LIS this error could dramatically degenerate the utility of this solution. The reason for this large perturbation noise is due to the fact that each individual element in the stream could affect all the possible outputs of the algorithm over the entire stream. This phenomenon could also occur for more sophisticated streaming algorithms that compute the LIS by using a small sketch of stream ([12, 19] for example). Although such solutions could reduce the space requirements, the use of a sketch does not directly reduce the error due to the perturbation noise since an element of the stream could still affect a large number of outputs (e.g. linear with the length of stream).

To overcome this problem, we decompose the computation of the LIS over segments of the stream. This intuition follows the

idea proposed by Chan et al. [4] where a linear and binary decomposition frameworks are employed to privately compute the number of non-zero elements in a binary stream. Despite the similarity in these decompositions, the computation of the longest increasing subsequence is harder to achieve than the simple count function. For this reason, we study the utility loss in approximating the LIS inflicted by using the local information of the stream. Due to space limitation, we consider only an extension of the binary decomposition since it provides better utility with respect to the linear decomposition proposed in the original paper [4].

In our work, we investigate the implications of decomposing the LIS computation over blocks (i.e. stream segments) both from the utility and complexity perspective. It is important to note that the nature of the decomposition should be data-independent to avoid additional privacy cost. In principle, any algorithm $\mathcal{A}_{LIS}$ that computes the LIS (either exact or approximate way) can be used as a building block to compute the LIS on each stream segment so that the perturbation noise required by the privacy mechanism can be reduced with respect to the direct use of $\mathcal{A}_{LIS}$. On the other hand, by limiting our computation on segments we introduce an approximation error.

In the rest of the section, we use the Patience Sorting algorithm [15] as a simple building block. We prove the reduction in the perturbation noise and the approximation error of our solution. We focus on this particular algorithm because it allows us to have an internal procedure that computes the exact length of the LIS over segments of the stream. In this way, we can directly measure how our decomposition impacts the exact solution. Since the original Patience Sorting algorithm computes not only the length of the LIS but also the elements forming the sequence, we use a modified version that only keeps the top element of the piles in the data structure as illustrated in the Algorithm1. In this way, we can compute the length of the LIS but using only $O(LIS(\sigma))$ space.

Before presenting our technique, we illustrate some concepts that will be useful in explaining our algorithm. A block $B = \sigma[j, j + b - 1]$ of size b represents a continuous segment of b symbols in the stream σ. Due to the dynamics of the data in the stream, a block assumes three different states over stream depending on the current time. At time i, the block B can be in one of the following states: **expired** hence the new arrival does not affect the block B (i.e. $j + b - 1 < i$), **active** when the new arrival is contained in the block B (i.e. $j \leq i \leq j + b - 1$) and **future** hence B contains only upcoming elements (i.e. $j > i$). An example of block decomposition of the stream is illustrated in Figure 2. The life cycle of a block B consists of starting as a future block, becoming active, and finally the block expiration.

3.1 Binary Decomposition

We start observing that in general the decomposition of the LIS over blocks may incur large approximation error. In fact, by simply dividing the stream into blocks and combining the length of their LIS as a answer could lead to an approximation error that is proportional to the number of blocks used in the decomposition. To reduce this error, we develop a decomposition using variable length blocks, where the number of blocks in the stream decomposition is $O(\log T)$. We organize the blocks in a binary tree where at time i the tree has $\log i$ levels. Each level $l = 0, \ldots, \log i$ in the tree partitions the stream into disjoint blocks of length $i/2^l$. Figure 3 illustrates an example of binary decomposition of the stream.

Using this representation, each node k in the tree is associated with a block B_k and it stores the perturbed value of the $LIS(B_k)$. At any time i the algorithm updates the noisy LIS of the active blocks in the binary tree, and it answers the query $LIS(\sigma[0, i])$ as

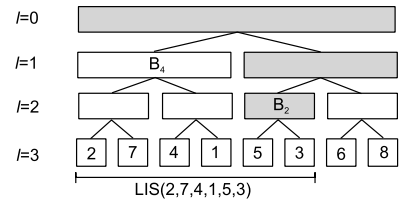


Figure 3: Binary Decomposition example. At time 5 (six symbols), the algorithm updates the active blocks (in gray). It computes the answer to the LIS query by summing the contributions of B_2 and B_4 containing the 2 and 4 most recent symbols respectively.

Algorithm 2 Binary Decomposition

1: **procedure** BINARY DECOMPOSITION(T, α, σ)
Input: upper bound on the stream length T; privacy parameter α; event stream σ
Output: $\tilde{l}(\sigma)$ released longest increasing subsequence
2: **for** $(i = 0, 1, \ldots, T - 1)$ **do**
3: **for** (every active B at time i) **do**
4: UPDATE PILES($B, \sigma(i)$)
5: **end for**
6: **for** (every block B that will expire at time $i + 1$) **do**
7: $LIS(B) \leftarrow$ number of piles for the block B
8: $\tilde{l}(B) \leftarrow LIS(B) + Lap(2 \log T/\alpha)$
9: **end for**
10: Let $i_1 < i_2 < \cdots < i_m$ be the positions of non-zeros in the binary representation of $i + 1$
11: $\tilde{l}(\sigma) \leftarrow 0$
12: $k \leftarrow i$
13: **for** $(j = i_1, i_2, \ldots, i_m)$ **do**
14: $B \leftarrow \sigma(k - 2^j + 1) \cdots \sigma(k)$ ▷ Retrieve the block to reconstruct the LIS
15: $k \leftarrow k - 2^j$
16: $\tilde{l}(\sigma) \leftarrow \tilde{l}(\sigma) + \tilde{l}(B)$ ▷ Sum the noisy contributions of the expired block B
17: **end for**
18: **Output** $\tilde{l}(\sigma)$
19: **end for**
20: **end procedure**

illustrated in Algorithm 2.

Algorithm Description. In the loop at lines 3-5, the algorithm updates the piles for the active blocks associated with the time i. In particular, the procedure `Update Piles` implements the Patience Sorting algorithm as in Algorithm 1, where in this case the update is performed independently on each active block B for any new coming element $\sigma(i)$. At lines 6-9, the noisy length of the LIS for each block that will expire is computed. At line 10, we compute the binary representation of $i+1$ and let $i_1 < i_2 < \cdots < i_m$ be the positions of non-zeros in such representation. Then the answer for $LIS(\sigma[0, i])$ is computed by summing up the length of the LIS for the blocks containing the most recent $2^{i_1}, 2^{i_2}, \ldots, 2^{i_m}$ elements respectively. Therefore at each time i, the output result is obtained by adding the contributions of at most $\Theta(\log i)$ blocks in the loop at lines 13-17.

Privacy Analysis. We can observe that each element affects at most $\log T$ blocks; therefore, perturbing the LIS of each block with a random variable from $Lap(\log T/\alpha)$ is sufficient to satisfy the privacy requirement.

THEOREM 4 (BINARY DECOMPOSITION PRIVACY). *The Binary Decomposition achieves α-differential privacy.*

PROOF. *In this decomposition, each element $\sigma(i)$ participates in the LIS of at most $\log T$ active blocks. Therefore, for any two neighboring streams the difference in L_1-norm of their outputs can*

be bounded by $\log T$. *Therefore using Theorem 1, it is sufficient to add to each LIS of each block a random variable from a Laplace distribution with parameter* $\log T/\alpha$ *to satisfy the privacy requirement.* □

Utility Analysis. This decomposition with variable length blocks allows us to reduce the perturbation error due to the privacy mechanism. However, in this way we introduce an approximation error that depends on the number of blocks. We can observe that at most $O(\log T)$ blocks of variable length are needed to answer a LIS query. The utility results for this decomposition are reported below.

LEMMA 2 (BINARY BLOCK ERROR BOUND). *Let* σ *be a stream of* T *symbols, and let* $LIS(\sigma) = \frac{\sqrt{T}}{\beta}$, *where* β *is positive. Without loss of generality we assume* $T = 2^t - 1$, *and we consider a partition of the stream* σ *into* $B_0, B_1, \ldots, B_{t-1}$ *non-overlapping blocks, where each block* B_k *is of size* 2^k. *Then in reporting the sum of the longest increasing subsequence in each block,* $lis(\sigma) = \sum_{k=0}^{t-1} LIS(B_k)$, *we incur the following approximation error.*

$$LIS(\sigma) \leq lis(\sigma) \leq \begin{cases} \log T \cdot LIS(\sigma) & \beta \geq 1 \\ (1 + \log \beta\sqrt{T}) \cdot LIS(\sigma) & \beta \in [1/\sqrt{T}, 1) \end{cases} \tag{4}$$

PROOF. *First, we start noticing the following lower-bound* $lis(\sigma) \geq LIS(\sigma)$. *In fact, the part of the real longest increasing subsequence which is contained in each block is at most the length of the longest increasing subsequence in the stream segment represented by the block. Second, we prove the two cases separately. For short value of* $LIS(\sigma)$ ($\beta \geq 1$), *we consider the case where each segment in each block is monotonic but none of them can be concatenated to form an increasing sequence in the entire stream. Then, we have that* $LIS(\sigma) \geq LIS(B_k)$, *for* $k = 0, \ldots, t-1$, *which leads to* $\log T \cdot LIS(\sigma) \geq \sum_{k=0}^{t-1} LIS(B_k) = lis(\sigma)$. *For the case of long value of LIS (*$\beta \in [1/\sqrt{T}, 1)$*), we proceed as follows. Let* j *be a positive integer such that* $2^{j-1} < \sqrt{T}/\beta \leq 2^j$. *Therefore, for all the blocks* B_k *with* $k \geq j$ *we have that* $LIS(B_k) \leq \sqrt{T}/\beta$, *otherwise there exists a monotonic sequence which is longer than the longest increasing subsequence, hence we have a contradiction. Furthermore, due to the binary tree decomposition the sum of the length of the LIS for the blocks* B_k *with* $k = 0, \ldots, j-1$ *can be bounded as follows.*

$$\sum_{k=0}^{j-1} LIS(B_k) \leq \sum_{k=0}^{j-1} 2^k = 2^j - 1 \leq 2\sqrt{T}/\beta \tag{5}$$

Therefore, the reported $lis(\sigma)$ *can be upper bounded with the value below.*

$$\begin{aligned} lis(\sigma) = \sum_{k=0}^{t-1} LIS(B_k) &\leq \sum_{k=0}^{j-1} LIS(B_k) + \sum_{k=j}^{t-1} LIS(B_k) \\ &\leq 2\sqrt{T}/\beta + (t-j)\sqrt{T}/\beta \\ &\approx \sqrt{T}/\beta(1 + \log \beta\sqrt{T}) \end{aligned} \tag{6}$$

This concludes the proof of the Lemma. □

THEOREM 5 (BINARY DECOMPOSITION UTILITY). *The binary decomposition algorithm for computing the length of the longest increasing subsequence achieves the following utility results.*

$$\begin{cases} ((\log T - 1) + \frac{\beta \log^{3/2} T}{\alpha\sqrt{T}} \ln \frac{1}{\delta}, \delta)\text{-useful} & \beta \geq 1 \\ (\log \beta\sqrt{T} + \frac{\beta \log^{3/2} T}{\alpha\sqrt{T}} \ln \frac{1}{\delta}, \delta)\text{-useful} & \beta \in [1/\sqrt{T}, 1) \end{cases}$$

PROOF. *This decomposition has the advantage that the number of blocks combined in estimating the length of the LIS is only logarithmic which leads to an approximation error as shown in Lemma 2. This decomposition introduces a perturbation noise which is a sum of at most* $O(\log T)$ *i.i.d. Laplace random variables with parameter* $O(\log T/\alpha)$. *Let* $\xi = \sum_k \eta_k$ *denote the error due to the sum of the Laplace random variables, we can use the result in Corollary 1 to bound this quantity. Choosing* $\nu = \sqrt{\sum_k \frac{\log T}{\alpha}}\sqrt{2 \ln \frac{2}{\delta}}$ *with probability at least* $1-\delta$, *the quantity* ξ *is at most* $O(\frac{\log^{3/2} T}{\alpha} \ln \frac{2}{\delta})$. *Therefore, the final utility follows using the results from Lemma 2 and by normalizing this value by the* $LIS(\sigma)$. □

Space Analysis. The space requirement for this approach is related to the number of active blocks that need to be updated and to the space complexity of the internal procedure. Due to the nature of the binary decomposition at any time i there are $\Theta(\log T)$ blocks that are active. Using a similar argument as in Theorem 5, we can show that the space complexity is $O(LIS(\sigma) \ln(\beta^2 LIS(\sigma)))$.

THEOREM 6 (BINARY DECOMPOSITION SPACE COMP.). *Let* $LIS(\sigma)$ *be the length of the longest increasing subsequence in the stream* σ, *then the Binary decomposition framework has space complexity* $O(LIS(\sigma) \ln(\beta^2 LIS(\sigma)))$.

PROOF. *We begin by recalling that the internal procedure for computing the length of the* LIS *is the Patience Sort algorithm, where we keep only the top of the piles. At any time* i *in the stream,* $\log T$ *blocks are active, one in each level of the tree structure. Furthermore, let* j *be a positive integer such that* $2^{j-1} < LIS(\sigma) \leq 2^j$. *Therefore for the blocks in any level* $i > j$ *in the tree, we can upper bound their space requirements with* $LIS(\sigma)(\log T - j + 1)$, *since* $LIS(\sigma)$ *is the current length of the longest increasing subsequence. On the other hand, due to the nature of the binary tree the space required by the blocks below the level* i *is* $2^j - 1$. *Therefore the space complexity for this approach is* $O(LIS(\sigma)(\log T - j + 1))$. *Using the notion that* $LIS(\sigma) = \sqrt{T}/\beta$ *and* $j = \log(LIS(\sigma))$, *the previous requirements can be rewritten as* $O(LIS(\sigma) \ln(\beta^2 LIS(\sigma)))$. □

Time Analysis. The total update time for this solution is related to the updates of the active blocks. Since at every time i there are $\Theta(\log T)$ active blocks, the update time is $O(\log T \log LIS(\sigma))$ using Patience Sorting algorithm.

4. HIERARCHY MECHANISM

In the previous section, we showed that the binary decomposition considerably reduces the perturbation noise in the final output compared to the baseline approach. However, such technique suffers from the fact that the computation of the LIS is generally hard to be decomposed in blocks leading in some cases to a large approximation error. To overcome this problem, we propose a new algorithm which computes the LIS over the stream by simulating the behavior of the Patience Sorting algorithm. In contrast to our previous approaches, this solution computes the length of the LIS by smoothing the impact of each element with the purpose of reducing the perturbation noise while achieving a good approximation ratio.

The main idea is to reduce the impact of those elements that stay too long in the LIS so that the total noise required by the privacy mechanism is decreased. Given an integer $b > 0$, we construct a series of $m = \Theta(\ln \frac{T}{b})$ layers $l_0, l_1, \ldots, l_{m-1}$ with b buckets each, where at layer i each bucket contains 2^i elements. Given the series of elements with index $\{1, 2, \ldots, T\}$ in the stream, each layer

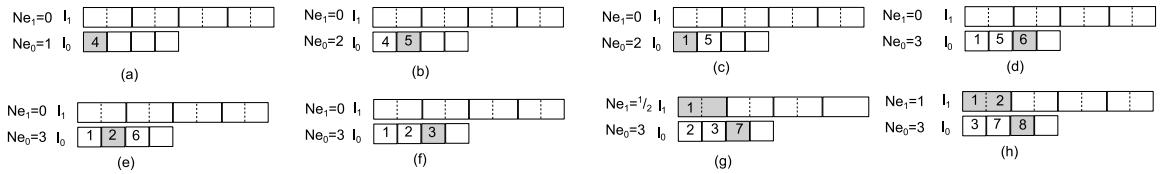


Figure 4: Running example of the Hierarchy mechanism on the input stream $4, 5, 1, 6, 2, 3, 7, 8$, $b = 4$ and $m = 2$.

simulates the behavior of the Patience Sorting algorithm where in this case the original piles are replaced with buckets that can contain multiple elements. In fact, at layer i the elements in the range $[(j-1)2^i + 1, j2^i]$ can be placed into the same bucket j. Intuitively, each layer has a different granularity, in fact l_0 keeps the exact top elements in the most recent b piles in the Patience Sorting algorithm, while l_1 keeps an approximation of the next $2b$ piles and so forth for the other layers. As the original algorithm, our procedure computes the length of the LIS by counting the number on non-empty buckets. In our case multiple elements may fall in the same bucket; therefore, we use a scaling factor equal to the length of the bucket to compute the contribution of each layer. Furthermore, in addition to insertion and replacement moves allowed in the Patience Sorting algorithm, we introduce an expiration move that forces elements that stay in a bucket at layer l_i for more than $2^i b$ iterations to be moved up to layer l_{i+1}. The algorithm computes the length of the LIS in the stream by adding the contribution at each layer. The code for this procedure is reported in Algorithm 3.

Algorithm Description. The algorithm starts initializing a set of m layers containing b buckets each, at lines 2-3. Within a layer i, each bucket is denoted with $P_i(j)$, for $j = 1, \ldots, b$ and it has size $2^i b$. In the main loop, lines 4-21, each new element coming in the stream is inserted in the first layer using the the same rule as the Patience Sorting algorithm, lines 5-7. In the inner loop at lines 9-13, the algorithm checks layer by layer to find the expired elements. When an expired element p in a pile $P_i(j)$ is found, the algorithm removes p and inserts it in the next layer. At line 14, the number of non-empty buckets for each layer is computed by normalizing the number of elements within each bucket with the corresponding bucket's size. In the loop at lines 16-19, the perturbation noise is applied to each count and finally the length of the LIS is returned.

We illustrate our hierarchy mechanism in the example below.

EXAMPLE 2. *Consider the situation in Figure 4. When the first element arrives in the stream it is placed in the first bucket at l_0 as shown in (a). The second element that arrives is 5, since it is larger than 4 it is placed in the next bucket (b). The third element in the stream is 1. Since the insertion of the elements in the buckets follows the same rules as the Patience Sorting algorithm, we find the bucket that contains the smallest element larger than 1 and insert this element in that bucket. Therefore, in our case, 1 overwrites 4 in the first bucket (c). At this point the length of the LIS is 2, as represented by the number of non empty buckets in l_0. The algorithm proceeds in a similar manner of the next three incoming elements (d),(e) and (f). After these new elements, the element 1 in l_0 is moved up to l_1 since it has been present in l_0 for more than b steps and the new incoming element 7 is inserted in l_0 (g). In the next step, the element 2 is moved up, and it is inserted in the same bucket with the element 1. At the same time the new element 8 is inserted in l_0 (e). The reported length of the LIS is obtained by summing the contribution of each layer. Layer l_0 contributes with $Ne_0 = 3$ and l_1 contributes with $Ne_1 = 1$. Hence the algorithm reports a length of the LIS of 4 while the exact length is 5.*

Algorithm 3 Hierarchy Mechanism

1: **procedure** HIERARCHY MECHANISM(T, α, σ, b)
   Input: upper bound on the stream length T; privacy parameter α; event stream σ; accuracy parameter b
   Output: $\tilde{l}(\sigma)$ released longest increasing subsequence
2: $m = \Theta(\ln \frac{T}{b})$
3: Initialize each layer $l_i = [P_i(1), \ldots, P_i(b)]$ $i = 0, \ldots, m-1$ with b empty buckets
4: **for** $(i = 0, 1, \ldots, T-1)$ **do**
5: Insert $\sigma(i)$ in l_1
6: Find the largest $P_1(j)$ such that $P_1(j) \leq \sigma(i)$
7: $P_1(j+1) = \sigma(i)$
8: **for** $(i = 0, \ldots, m-1)$ **do**
9: Let p be the element that expires at l_i
10: Remove p from l_i and insert it in l_{i+1}
11: Find the largest element in $P_{i+1}(j)$ such that $P_{i+1}(j) \leq p$
12: $P_{i+1}(j+1) = p$
13: **end for**
14: Let Ne_i be the number of non-empty buckets at layer l_i
15: $\tilde{l}(\sigma) \leftarrow 0$
16: **for** $(i = 0, 1, \ldots, m-1)$ **do**
17: $\widehat{Ne_i} \leftarrow Ne_i + Lap(mb/\alpha)$
18: $\tilde{l}(\sigma) \leftarrow \tilde{l}(\sigma) + \widehat{Ne_i}$ ▷ Sum the noisy contribution of each layer
19: **end for**
20: **Output** $\tilde{l}(\sigma)$
21: **end for**
22: **end procedure**

Privacy Analysis. In this algorithm the contribution of each element on the LIS is progressively decreased according to the layer in which the element appears. The privacy result for our hierarchy mechanism is reported in the following theorem.

THEOREM 7 (HIERARCHY MECHANISM PRIVACY). *The Hierarchy Mechanism achieves α-differential privacy.*

PROOF. *Given any two neighboring streams, we can observe that each element can affect at most m layers over the entire stream. In particular, at l_0 an element contributes to the LIS with a factor 1 for b times, at l_1 contributes with factor 1/2 for $2b$ and at the general level l_i contributes with factor $1/2^i$ for $2^i b$ times. Let $\mathbf{Ne}$ be the vector of contributions for each layer for the input stream $\sigma = a_1, \ldots, a_i \ldots, a_T$. Then, $\forall i \in [1, T]$ and $\sigma' = a_1, \ldots, a'_i \ldots, a_T$ we have that*

$$\|\mathbf{Ne}(\sigma) - \mathbf{Ne}(\sigma')\| \leq mb \tag{7}$$

Then adding a random Laplace noise with parameter mb/α to the contribution of each layer i, is sufficient to satisfies α-differential privacy. Furthermore, using Corollary 1 we can see that the additive error introduced by noise is only $O(\frac{b}{\alpha} \log^{3/2}(\frac{T}{b}) \log(\frac{2}{\delta}))$. □

Approximation Error. Our algorithm smooths the contribution of each element in the stream according to its layer leading to an underestimated value for the length of the LIS. The following Theorem summarizes the approximation ratio in the worst case.

THEOREM 8 (HIERARCHY APPROXIMATION ERROR). *Let σ be a stream of length T, and b be the number of buckets in each*

Table 1: Summary of results for LIS query over entire stream.

Method	Error	Memory	Update Time
Baseline	$O(\frac{\beta\sqrt{T}}{\alpha}\ln\frac{1}{\delta})$	$O(LIS(\sigma))$	$O(\log\frac{\sqrt{T}}{\beta})$
Binary	$O((\log T-1)+\frac{\beta\log^{3/2}T}{\alpha\sqrt{T}}\ln\frac{1}{\delta})$ where $\beta\geq 1$ $O(\log\beta\sqrt{T}+\frac{\beta\log^{3/2}T}{\alpha\sqrt{T}}\ln\frac{1}{\delta})$ where $\beta\in[1/\sqrt{T},1)$	$O(LIS(\sigma)\ln(\beta^2 LIS(\sigma)))$	$O(\log T\log\frac{\sqrt{T}}{\beta})$
Hierarchy	$O((1-\frac{T-b}{T+b})+\frac{b\beta}{\sqrt{T}\alpha}\log^{3/2}(\frac{T}{b})\log(\frac{2}{\delta}))$	$O(LIS(\sigma))$	$O(\log b\log\frac{T}{b})$

layer of our algorithm. Then, the hierarchy mechanism returns a $(1-\frac{T-b}{T+b})$*-approximation of the length of LIS.*

PROOF. *Let* k *be the length of the LIS over the entire stream. We begin by showing that this algorithm never overestimates the length of the LIS and then proceed by showing the error in the underestimate. To understand why this algorithm always reports a length of the LIS less or equal to the real length we consider the following case. Let us assume that there exists an element* $\sigma(j)$ *in a bucket at level* $i>0$ *in our algorithm that differs from the Patience Sort. Since this element is extra in our algorithm it means that there is an element* $\sigma(j')$*,* $j'>j$ *that replaces* $\sigma(j)$ *in the exact Patience Sort. Since* $\sigma(j')<\sigma(j)$*, we have that in our structure* $\sigma(j')$ *has replaced another element* $\sigma(j'')$*. Due to the nature of our algorithm this operation could only occur in a layer* $i'<i$*, hence in replacing* $\sigma(j'')$ *with* $\sigma(j')$ *in our algorithm we have a larger loss of contribution than replacing* $\sigma(j)$*. Therefore we cannot have an overestimate length of the LIS.*

Now, we examine the error in underestimating the length of the LIS. Consider a worst case scenario where only the first k *symbols in* σ *contribute to the LIS, while the rest of the stream does not increase the length of the LIS. In this situation, as the stream proceeds the elements of the LIS that initially are in layer* 0 *are progressively moved up introducing a small additive error. Below, we quantify this error. Let* $m=\log(\frac{T}{b}+1)-1$ *be the number of layers in our structure, then the maximum additive errors on the LIS is achieved when all the elements forming the LIS are in layer* m*. This quantity is computed as follows.*

$$\sum_{i=1}^{m}\frac{k}{2^i}=k\left(\frac{T-b}{T+b}\right) \tag{8}$$

Hence the returned value from our algorithm is lower bounded by $LIS(\sigma)(1-\frac{T-b}{T+b})$*. This shows that our returned length* $\tilde{l}(\sigma)$ *satisfies the following inequality.*

$$LIS(\sigma)\left(1-\frac{T-b}{T+b}\right)\leq\tilde{l}(\sigma)\leq LIS(\sigma) \tag{9}$$

Therefore, our algorithm provides a $(1-\frac{T-b}{T+b})$*-approximation of the length of LIS.* □

Space Analysis. Since this algorithm simulates the Patience Sorting algorithm by keeping only the top of the piles forming the LIS, it follows that the space complexity is linear with the length of the LIS in the stream $O(LIS(\sigma))$.

Time Analysis. For any new incoming element in the stream, the total running time is given by the cost required for updating each pile. There are at most $m-1$ buckets, one for each level, that need update, where each operation requires $O(\log b)$ time. Since $m=\Theta(\log\frac{T}{b})$, the update time is $O(\log b\log\frac{T}{b})$.

This solution points out a strong connection between the approximation ratio and the noise required to achieve privacy. We can see that increasing b has a beneficial effect on the approximation ratio but on the other hand increases the privacy cost. In fact, as an extreme case using $b=T$ the algorithm returns the exact length of the LIS but incurs a large perturbation noise. Compared with our decomposition framework, this algorithm provides the user with a way to balance the approximation ratio and the noise due to the privacy mechanism.

5. SUMMARY OF RESULTS

Table 1 summarizes the utility results of our proposed solutions. We can see that both our strategies outperform the baseline approach in many perspectives. We notice that the baseline approach incurs a large perturbation error which could dramatically compromise the utility. Specifically, the additive error in the baseline strategy grows linearly with the length of the stream. For the binary decomposition instead, we provide output-sensitive utility results showing the benefits of this technique for different lengths of LIS. Due to the use of disjoint blocks, this approach incurs a considerably smaller perturbation error with respect to the baseline solution. In fact, the dependency of the error with respect to the perturbation noise is only polylogarithmic in this case. Furthermore, we can observe that the decomposition framework has small space requirements and update time. In principle, the space and time complexity of this solution could be further improved by using more sophisticated algorithms (e.g. [12, 19]) as internal procedure instead of relying on the Patience Sorting. For count based statistic the binary decomposition has been shown very effective; however due to the nature of the LIS, this strategy incurs an approximation error. Our hierarchy approach specifically addresses the LIS problem by directly simulating the Patience Sorting algorithm. This procedure incurs a smaller computational time and it has small memory requirements. Comparing the worst case performance of this technique with the binary decomposition, we can observe that the decomposition framework is still superior leading to a smaller additive error with the same approximation ratio. This result is due to the fact that the hierarchy strategy suffers when the LIS constitutes the initial part of the stream. In fact, as the execution proceeds the elements in the sketch are moved in higher level increasing the approximation error over the stream. However, we can notice that in real scenarios such situation is unlikely to occur because in many applications we can assume that the stream presents trends over time.

5.1 Extensions

In this section, we describe how to employ our developed techniques to solve real world problems.

Detecting trends in time-series data. Our proposed techniques can be extended to effectively detect trends in time-series data by restricting the computation of the LIS over windows in the stream. In fact, in monitoring applications, recent data is more important than distant data; therefore, using a sliding window W, we limit the computation of the LIS on the most W recent data. For example, a sudden increase of price in financial data will lead to an increment

in the length of the LIS in the current window. Constraining the computation of the LIS on a sliding window of length W is beneficial both from the utility and complexity perspective. In fact, it has been shown in [3] that for the binary mechanism the use of a sliding window reduces the impact of the privacy to a factor that is independent from the length of the stream but it is only related to the size of the window W. A similar result can be also derived for the hierarchy mechanism, where in this case, the number of layers in the data structure depends only on the length of W rather than the entire stream.

Approximate String Matching. The problem of computing the LIS is a classical string matching problem that has been extensively studied in computational biology [14]. However, only few solutions have been proposed to privately match biological sequences. Generally, these approaches provide privacy and security in matching strings by applying cryptographic techniques [18]. However, due to their high complexity these approaches may not be effective in real scenarios. In this setting, we believe that our solutions can be very promising by providing formal privacy guarantee and incurring a small computational overhead. Since the problem structure of the LIS is similar to other popular problems for computing string similarity measures (e.g. edit distance), we believe that our hierarchy approach could be a first step toward the design of efficient privacy preserving algorithms for matching strings.

6. CONCLUSIONS

In this paper, we considered the problem of privately detecting trends in stream data. Specifically, we addressed the problem of computing the length of the LIS while protecting the presence of single event in the stream. We developed two different solutions that provide formal guarantee of privacy. The first approach approximates the length of the LIS by assembling local information computed on segments of the stream. The second approach constructs a small sketch of the stream by exploiting the structure of the problem. Using a rigorous analysis, we showed that these strategies provided significant benefits over the baseline approach.

For the future, we consider to investigate two possible research directions. First, we plan to further develop our extensions and turn our theoretical results into concrete algorithms to be applied to solve time-series monitoring and string matching problems. Second, our proposed solutions provide important insights about the privacy implications for computing complex ordered statistics. Therefore, we plan to better understand what kind of privacy sketching algorithms can benefit in this setting.

7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1117763.

8. REFERENCES

[1] Miklós Ajtai, T. S. Jayram, Ravi Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 370–379. ACM, 2002.

[2] David Aldous and Persi Diaconis. Longest increasing subsequences: From patience sorting to the baik-deift-johansson theorem. *Bull. Amer. Math. Soc*, 36:413–432, 1999.

[3] Jean Bolot, Nadia Fawaz, S. Muthukrishnan, Aleksandar Nikolov, and Nina Taft. Private decayed predicate sums on streams. In *Proceedings of the 16th International Conference on Database Theory*, ICDT '13, pages 284–295, New York, NY, USA, 2013. ACM.

[4] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3), November 2011.

[5] Graham Cormode, S. Muthukrishnan, and Süleyman Cenk Sahinalp. Permutation editing and matching via embeddings. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*, ICALP '01, pages 481–492, 2001.

[6] Cynthia Dwork. Differential privacy. In *ICALP*, 2006.

[7] Cynthia Dwork. Differential privacy in new settings. In *SODA*, pages 174–183, 2010.

[8] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC 2006*.

[9] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 715–724, New York, NY, USA, 2010. ACM.

[10] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010.

[11] Funda Ergun and Hossein Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 730–736, 2008.

[12] Parikshit Gopalan, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Estimating the sortedness of a data stream. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 318–327, 2007.

[13] Anupam Gupta and Francis X. Zane. Counting inversions in lists. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '03, pages 253–254, 2003.

[14] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.

[15] J. M. Hammersley. A few seedlings of research. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability*, pages 345–394, Berkeley, Calif., 1972. University of California Press.

[16] David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. In *Proceedings of the 11th annual international conference on Computing and Combinatorics*, COCOON'05, pages 263–272, 2005.

[17] Frank D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD '09*.

[18] S. Rane and Wei Sun. Privacy preserving string comparisons based on levenshtein distance. In *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pages 1–6, Dec 2010.

[19] Michael Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *SODA*, pages 1698–1709, 2013.

Efficient Sanitization of Unsafe Data Correlations

Bechara AL Bouna
Department of Computer Science and Engineering
Qatar University & Antonine University
Doha, Qatar - Baabda, Lebanon
bechara.albouna@upa.edu.lb

Chris Clifton
Department of Computer Science
Purdue University
West Lafayette, Indiana - USA
clifton@cs.purdue.edu

Qutaibah Malluhi
Department of Computer Science and Engineering
Qatar University
Doha, Qatar
qmalluhi@qu.edu.qa

ABSTRACT

In this paper, we present a study to counter privacy violation due to unsafe data correlation. We propose a safe correlation requirement to keep correlated values bounded by l-diversity and evaluate the trade-off to be made for the sake of a strong privacy guarantee. Finally, we present a correlation sanitization algorithm that enforces our safety constraint and demonstrates its efficiency.

1. INTRODUCTION

Preserving privacy in outsourced databases has received considerable attention in the last decade. Several privacy constraints [23, 22, 16, 11] have been defined on datasets to prevent disclosure of sensitive information related to individuals. These constraints are based on generalizations that transform quasi-identifiers values into a general form and create quasi-identifier groups to eliminate possible linking attacks. A second approach is *table decomposition*: Quasi-identifiers and sensitive values are placed in separate tables, and tuples are divided into groups that are linked in a way that provides sufficient uncertainty in the join criteria to meet privacy constraints. This approach has been alternately termed anatomy [29], fragmentation [4] and slicing [14]; we will use the term anatomy to refer to this class of approaches, as it does not have other meanings in the database community.

Anatomy has the advantage that exact data values are maintained, allowing data and actions on individual data values to be outsourced. Only the link between identifying and sensitive values is generalized. We envision this work being used in the context of [20], where the actual links (and in our case, some data values) are encrypted to ensure the server cannot violate privacy, while still enabling some server-side use of the data.

As an example, Figure 1a shows prescription history, where the attribute *DrugName* is sensitive. Figure 1b represents an anatomized version of Table Prescription with attributes separated into $Prescription_{QIT}$ and $Prescription_{SNT}$. The anonymized table satisfies the 2-diversity privacy constraint[16]; given the 2-diverse table, an adversary can at best link a patient to a drug with a probability equal to 1/2.

Despite anatomy's efficiency in preserving privacy and data fidelity, it and other generalization based techniques defects when some types of correlation exist in the data.

The most obvious problem is when values in identifiers (or quasi-identifiers) are directly correlated with sensitive values as discussed in [27] and [8]. Based on knowledge of such correlation (possibly learned from the data), an adversary may increase the probability that a given individual is linked to a given sensitive value with probability greater than the $1/l$ enforced by the anatomization groups. The inter quasi-identifying group correlation between *United States* and *Retinoic Acid* given in Figure 1 shows that with respect to knowledge mined from the anonymized data, an adversary is able to assert knowledge regarding the global distribution of countries and drugs. Such global distribution increases the probability of linking individuals to sensitive values on the basis of their countries. The authors of [27, 8] demonstrated how correlation can be used to violate privacy constraints. They argued that the (sometimes implicit) assumptions of an i.i.d. model and random worlds model, when tuple independence does not hold in the actual data, allows adversaries to learn and use cases where these assumptions do not hold to violate privacy. In this paper, we shed more light on the threat of data correlation that could be found after a naïve anonymization of a table, and give methods to control that risk. While we use the anatomy model [29] in our examples, this work also applies to other bucketization techniques such as fragmentation [4] and slicing [14].

1.1 Contributions

We present a study to counter privacy violations and at the same time preserve data utility. Our contributions can be summarized as follows:

- We propose a safe correlation requirement to reduce the threat of exposed correlations between quasi-identifier and sensitive attributes. We show that under this requirement, correlations can be bounded by a trade-off between utility and privacy.
- We provide a sanitization algorithm to ensure safety from correlations by solving a linear programming problem in a post-anonymization process.

The key idea is that we do not completely hide correlations

Country	Manufacturer	Drug Name
United States	Envie De Neuf	Mild Exfoliation
Columbia	Gep-Tek	Azelaic acid
United States	Raphe Healthcare	Retinoic Acid
United States	Envie De Neuf	Mild Exfoliation
France	Raphe Healthcare	Azelaic acid
United States	Raphe Healthcare	Retinoic Acid
Columbia	Jai Radhe	Cytarabine
United States	Raphe Healthcare	Azelaic acid
Columbia	Raphe Healthcare	Retinoic Acid
France	Jai Radhe	Cytarabine
United States	Raphe Healthcare	Azelaic acid
United States	Raphe Healthcare	Retinoic Acid
Columbia	PQ Corp.	Epsom. Magnesium
United States	Envie De Neuf	Mild Exfoliation
United States	Jai Radhe	Adapalene

(a) Original Prescription table

Country	Manufacturer	GID
United States	Envie De Neuf	1
Columbia	Gep-Tek	1
United States	Raphe Healthcare	1
United States	Envie De Neuf	2
France	Raphe Healthcare	2
United States	Raphe Healthcare	2
Columbia	Jai Radhe	3
United States	Raphe Healthcare	3
Columbia	Raphe Healthcare	3
France	Jai Radhe	4
United States	Raphe Healthcare	4
United States	Raphe Healthcare	4
Columbia	PQ Corp.	5
United States	Envie De Neuf	5
United States	Jai Radhe	5

GID	Drug Name
1	Mild Exfoliation
1	Azelaic acid
1	Retinoic Acid
2	Mild Exfoliation
2	Azelaic acid
2	Retinoic Acid
3	Cytarabine
3	Azelaic acid
3	Retinoic Acid
4	Cytarabine
4	Azelaic acid
4	Retinoic Acid
5	Epsom. Magnesium
5	Mild Exfoliation
5	Adapalene

(b) Anonymized Prescription table $Prescription_{QIT}$ and $Prescription_{SNT}$

Figure 1: Example scenario

(we want to support learning from the data), this follows the spirit of t-closeness [11], but building on anatomy allows us greater grouping flexibility without the utility loss from over-generalizing data values.

2. ADVERSARY MODEL

We assume that both the adversary and defender have knowledge of correlations in the data; in the case of the adversary, his/her knowledge is mainly based on what can be learned from the anonymized data. As for the defender, it can include any correlations that can be learned from the original data. We also assume that an adversary has outside information enabling it to link (quasi)-identifying information with individuals. Thus all quasi-identifiers and identifiers are considered individually identifiable.

We assume that the adversary *does not* have prior knowledge of sensitive values for specific individuals. For example, if an adversary knew the prescriptions being taken by all of the individuals in Figure 1b except for a specific individual, then it is clearly possible for the adversary to determine his/her prescriptions. While there are methods to deal with data analysis under such a scenario up to a point (e.g., [5]), they violate our goal of storing and disclosing actual data values. Full protection against other kind of background knowledge is impossible while still maintaining data utility [5].

3. RELATED WORK

The anatomy [29], fragmentation [4] and slicing [14] models have been proposed to provide a technique that ensures privacy and preserves data granularity lost using generalization - based approaches such as k-anonymity [22, 23], l-diversity [16], (α, k)-anonymity [28], and t-closeness [11].

Unfortunately, these models fail to provide the promised privacy because of the dependencies that might exit in the data. In [14] the authors provide a technique that combines both generalization and bucketization to protect datasets against membership disclosure. Despite its originality, this approach remains vulnerable to negative correlations even while grouping attributes that are highly correlated. In [25] [15], disassociation is applied in a way to preserve both, the original terms to leverage utility and the k^m-anonymity privacy constraint. A privacy breach can still occur due to the lack of diversity. Particularly, when ensuring k^m-anonymity without using generalization which makes the technique vulnerable to homogeneity attacks.

An adversary discovering correlations in the data can use these correlations to discover information about individuals [27] [8]. In [27], the authors consider correlations as foreground knowledge that can be mined from anonymized data. They use the possible worlds model to compute the probability of associating an individual with a sensitive value based on a global distribution. In [8], a Naïve Bayesian model is used to compute association probability. They use exchangeability [2] and DeFinetti's theorem [21] to model and compute patterns from the anonymized data.

There are two components to each of these papers. The first is a relatively simple idea - that we can use correlations to link identifying information to sensitive values. A much deeper aspect is that they show *how* an adversary can find such correlations in the anonymized data. Our work addresses the first component directly: We ensure that even given knowledge of the true correlations present in the data, the probability that a particular sensitive value can be linked to a particular individual is below a threshold (e.g., $1/l$ for l-diversity, or α for (α, k)-anonymity.) This ensures that our method prevents not only the attacks in [27, 8], but any other correlation-based attacks that may be developed. Furthermore, we try to preserve and expose correlations where possible, increasing utility of the data. In [13], the authors deal with background knowledge that can be mined from the data. In their paper, they focus mainly on what is known as negative correlations limiting by that the ability to handle positive and exposed correlations.

[5] defines the notion of differential privacy to handle private data publishing efficiently. The technique gained much popularity among computer scientists providing strong assumptions on the way that data should be released. In essence, differential privacy guarantees privacy without making any assumption on the adversary's background knowledge. More accurately, it shows robustness when a certain number of tuples in the dataset are known by the adversary. Despite its originality, differential privacy tends to be less efficient when correlation among the tuples is high [9]. In addition, the appropriate value of ϵ to achieve the needed

real-world privacy is unclear [10].

While there are approaches that bridge differential privacy and generalization for data release [17, 12], they are not applicable in our environment. For example, [17] releases noisy group sizes; if applied in our model, the server would likely be able to use query history to distinguish true vs. fake tuples and thus reduce this noise, violating ϵ-differential privacy. Alternatively, [12] uses sampling to show that at some point k-anonymization techniques can achieve a relaxation of ϵ-differential privacy with a small error probability δ. This, however, significantly decreases the utility of the data which already suffers from constraints imposed by generalization.

4. FORMALIZATION

We first define basic concepts and notations used in the paper (see also Table 1).

Given a table T with a set of attributes $\{A_1, ..., A_b\}$, $t[A_i]$ refers to the value of attribute A_i for the tuple t. Attributes of a table are divided as follows:

- *Quasi-identifiers* A^{qi} represent attributes that can be used (possibly with external information available to the adversary) to identify the individual associated with a tuple in a table. *Name, Gender, Age and Zipcode* are examples of quasi-identifiers.
- *Sensitive attributes* A^s contain sensitive information that must not be linkable to an individual. In our example (Table 1), *DrugName* is considered sensitive and should not be linked to an individual.

Definition 1 (Equivalence class / QI-group)**.** *[22] A quasi-identifier group (*QI-group*) is defined as a subset of tuples of* $T = \bigcup_{j=1}^{m} QI_j$ *such that, for any* $1 \leq j_1 \neq j_2 \leq m$, $QI_{j1} \cap QI_{j2} = \phi$.

Table Prescription shown in Figure 1a is composed of 6 different quasi-identifier groups identified by their GID attribute's values.

Definition 2 (l-diversity)**.** *[16] a table* T *is said to be* l*-diverse if each of the QI-groups* $QI_j (1 \leq j \leq m)$ *is* l*-diverse; i.e.,* QI_j *satisfies the condition* $c_j(v_s)/|QI_j| \leq 1/l$ *where*

- m *is the total number of QI-groups in* T
- v_s *is the most frequent value of* A^s *in* QI_j
- $c_j(v_s)$ *is the number of tuples of* v_s *in* QI_j
- $|QI_j|$ *is the size (number of tuples) of* QI_j

For instance, quasi-identifier group QI_1 in Figure 1a is 3-diverse containing 3 distinct sensitive values.

Definition 3 (Anatomy)**.** *Given a table* T*, we say that* T *is anatomized if it is separated into a quasi-identifier table (*T_{QIT}*) and a sensitive table (*T_{SNT}*) as follows:*

- T_{QIT} *has a schema* $(A_1, ..., A_d, GID)$ *where* A_i *(*$1 \leq i \leq d$*) is either a nonsensitive or quasi-identifier attribute and GID is the group id of the QI-group.*
- T_{SNT} *has a schema* (GID, A^s_{d+1}) *where* A^s_{d+1} *is the sensitive attribute in* T.

Table 1: Notations

T	a table containing individuals related tuples
t_i	a tuple of T
u	an individual described in T
A	an attribute of T
A^{qi}	a quasi-identifier attribute of T
A^s	a sensitive attribute of T
QI_j	a quasi-identifier group
T^*	Anonymized version of table T
$\mathcal{CD}$	a set of correlation dependencies
$cd : A^{qi} \dashrightarrow A^s$	a correlation dependency between attribute A^{qi} and the sensitive attribute A^s

Figure 1b is an anatomized version of Table Prescription in Figure 1a in which only the links between individuals and their sensitive values are generalized.

To express correlations between attributes of an anonymized table T^*, we use the term correlation dependencies $\mathcal{CD}$ formally defined as follows:

Definition 4 (Correlation Dependency)**.** *Let* A^{qi} *be an attribute of* T^**, and* A^s *be the sensitive attribute of* T^**. A correlation dependency (*$cd^{qi} \in \mathcal{CD}$*) of the form of* $cd^{qi} : A^{qi} \dashrightarrow A^s \in \mathcal{CD}$ *exists over* T^* *if* $\exists v_s \in A^s$ *and* $v_{qi} \in A^{qi}$ *s.t.* $P(v_s|v_{qi}) >> P(v_s)$.

We assume that dealing with correlation dependencies is not a straightforward process in which we can assume that every correlation is unsafe. Such assumption contradicts the basic utility of data outsourcing and causes dramatic damage to the utility of aggregate analysis. It is important to specify to what extent correlation is unsafe and define its legitimate boundaries during the anonymization process. For completeness, we define the significance of a sensitive value v_s w.r.t. a quasi-identifier value v_{qi} based on a confidence and support measures to be discussed below.

Definition 5 (Significant Sensitive Value)**.** *Given a correlation dependency of the form* $cd^{qi} : A^{qi} \dashrightarrow A^s$ *over a table* T*, we say that a sensitive value* v_s *is significantly related to* v_{qi} *iff*

- $conf(v_{qi}, v_s) = Pr(A^s = v_s, A^{qi} = v_{qi})/Pr(A^{qi} = v_{qi})$ *is less than or equal to minConf threshold (*$conf(v_{qi}, v_s) \leq minConf$*) or greater than or equal to a maxConf threshold (*$conf(v_{qi}, v_s) \geq maxConf$*) and,*
- $sup(v_{qi}, v_s) \geq minSup$ *where minSup is defined to capture sensitive values that are frequently correlated with the quasi-identifier values.*

We use *confidence* (*conf*), easily mined from the data during anonymization, to determine the strength of a correlation dependency and limit the number of **significantly** related sensitive values. Specifically, a sensitive value related to a quasi-identifier value by a correlation dependency is significant if its *confidence* is at least equal to a maximum confidence (*maxConf*) threshold or at the most equal to a minimum confidence (*minConf*) threshold, and it has a *support* greater than a minimum support (*minSup*) threshold. *minConf*, *maxConf* and *minSup* are set to satisfy safety requirements as shown in the next section.

5. CORRELATION-BASED PRIVACY VIOLATION

High correlation would allow us to use the values of one attribute to predict the values of other attributes. While this is valuable knowledge, it can also violate the privacy constraints. The problems detailed in [27, 8] lie with the ability of an adversary to extract patterns (correlations) from an anonymized table that can be used to violate privacy. Summarizing, we define here the privacy problem as follows:

Definition 6 (Privacy Problem)**.** *A privacy violation occurs if for a given individual u, $Pr(u_s = v_s|T^*) > 1/l$, where v_s is a sensitive value of A^s, and T^* is an l-diverse anonymized version of T.*

Definition 6 provides a general perspective of the privacy breach but yet we cannot assume that every correlation is unsafe. As mentioned earlier, such an assumption contradicts the basic utility of data outsourcing. For this reason, we consider that for a given an anonymized table T^*, if an adversary is able to associate a *significant* sensitive value v_s to an individual u with a probability greater than $1/l$ based on the assumed adversary knowledge, we say that the privacy principle has been violated.

It is essential to enforce proper safety requirements during the anonymization process to keep *significant* correlations bounded and eliminate by that any possible breach of privacy.

We present in the following our safe correlation safety constraint to bound correlation dependencies of the form $cd^{qi} : A^{qi} \dashrightarrow A^s$.

Safety Constraint (Safe Correlation)**.** *Given a correlation dependency of the form ($cd^{qi} : A^{qi} \dashrightarrow A^s$) over T. Let v_{qi} be a value of quasi-identifier attribute A^{qi} and $v_s \in A^s$ be a sensitive value significantly related to v_{qi}. We say a safe correlation constraint is satisfied for T^* iff*

1. ***significant** sensitive values are uniformly distributed such that $Pr(A^s = v_{s_i}, A^{qi} = v_{qi}|T^*) = 1/\lambda_{v_{qi}}$ for $(1 \leq i \leq |\mathcal{S}(v_{qi})|)$ and,*
2. *there are at least l distinct significant sensitive values for v_{qi}, $|S(v_{qi})| \geq l$*

where

- *$\mathcal{S}(v_{qi})$ is the set of sensitive values **significantly** related to v_{qi} and,*
- *$\lambda_{v_{qi}} \geq l$ is the correlation constant.*

Using this safe correlation requirement we provide boundaries to correlation while making sure that the most frequent correlated value does not appear too frequently, and that the low correlation values do not appear too rarely in T^*. We note that the correlation constant $\lambda_{v_{qi}}$ depends on the actual correlation between a quasi-identifying value v_{qi} and the significant sensitive values. $\lambda_{v_{qi}}$ is determined in a post-anonymization process explained in the next section.

Theorem. *An adversary cannot use his/her previous knowledge of some of the significant correlations to link individuals to sensitive values in the anonymized dataset.*

Proof. Given that $Pr(A^s = v_s, u|T^*)$ can be written as $Pr(A^s = v_s, t_{v_{qi}}|T^*)$ where $t_{v_{qi}}$ is individual u's tuple and $t[A^{qi}] = v_{qi}$. Assuming that v_s is significantly related to v_{qi} meaning that $v_s \in S(v_{qi})$ and thus $Pr(A^s = v_s, A^{qi} = v_{qi}|T^*)$ is equal to $1/\lambda_{qi}$. If a privacy violation occurs as such $Pr(A^s = v_s, t_{v_{qi}}|T^*) > 1/l$, the correlation itself must violate our assumptions. According to the safe correlation constraint, significant correlations between sensitive and quasi-identifying values are bounded by l-diversity. In other terms, there are $l-1$ other sensitive values such that $Pr(A^s = v_{s_i}, A^{qi} = v_{qi}|T^*) = 1/\lambda_{v_{qi}}$ for $(1 \leq i \leq l-1)$. □

Figure 2 shows how we can achieve this safety constraint using the correlation sanitization algorithm defined in Section 6. As we can see, several values have been suppressed to make sure that both probabilities remain equal after the anonymization process.

Country	Manufacturer	GID
United States	Envie De Neuf	1
Columbia	Gep-Tek	1
United States	Raphe Healthcare	1
United States	Envie De Neuf	2
France	Raphe Healthcare	2
United States	Raphe Healthcare	2
Columbia	Jai Radhe	3
*	Raphe Healthcare	3
Columbia	Raphe Healthcare	3
France	Jai Radhe	4
*	Raphe Healthcare	4
*	Raphe Healthcare	4
Columbia	PQ Corp.	5
United States	Envie De Neuf	5
United States	Jai Radhe	5

GID	Drug Name
1	Mild Exfoliation
1	Azelaic acid
1	Retinoic Acid
2	Mild Exfoliation
2	Azelaic acid
2	Retinoic Acid
3	Cytarabine
3	Azelaic acid
3	Retinoic Acid
4	Cytarabine
4	Azelaic acid
4	Retinoic Acid
5	Epsom. Magnesium
5	Mild Exfoliation
5	Adapalene

Figure 2: Safe correlation: a post-anonymization safety constraint.

One subtle remaining issue is multi-dimensional correlations, where several combined attribute values can correlate with a sensitive attribute. Formally, we define a p-dimensional correlation dependency as follows:

Definition 7 (p-Dimensional Correlation Dependency)**.** *Let A^{qi} be quasi-identifying attribute of table T, we say a correlation dependency of the form $cd_p : (A_1^{qi}, ..., A_p^{qi}) \dashrightarrow A^s$ is p-dimensional where A^s is a sensitive attribute of T iff $\exists$ p values $v_1 \in A_1^{qi}, ..., v_p \in A_p^{qi}$ such that for a given $v_s \in A^s$, v_s is significantly related to $(v_{qi_1}, ..., v_{qi_p})$.*

Typically, dealing with p-dimensional correlation dependencies cannot be done while assuming a straightforward extension of the safety constraint. It is essential to consider parameters related to data utility with respect to safety. While this is left for a future work, we assume that safety is guaranteed if and only if any subset of possible attribute combinations of the p-dimensional correlation dependency antecedent is 'safe'.

6. PRIVACY ENFORCEMENT

We now provide the correlation sanitization algorithm, a mechanism to enforce the safe correlation requirement.

6.1 Correlation Sanitization: a Linear Programming Problem

Given an anonymized table T^*, the correlation between a significant sensitive value v_s and a quasi-identifying value v_{qi} can be referred to as $Pr(A^s = v_s, A^{qi} = v_{qi}|T^*)$ and

determined as follows:

$$Pr(A^s = v_s, A^{qi} = v_{qi}|T^*) = \frac{\sum_{QI_j \in \mathcal{QI}(v_{qi})} c_j(v_{qi}) \times Pr(A^s = v_s, t_i|QI_j)}{\sum_{QI_j \in \mathcal{QI}(v_{qi})} c_j(v_{qi})} \quad (1)$$

To achieve the safe correlation constraint, we solve the linear programming (LP) problem subject to maximizing the sum of count of QI-values in each QI-group in T^* such that, $\forall v_s \in S(v_{qi})$, $\sum_{j=1}^{m} p_{j,k}x_{i,j} = \frac{1}{\lambda_{v_{qi}}} \times c(v_{qi})$, where

- $x_{i,j}$ is a variable that represents the count of v_{qi} in QI-group QI_j denoted by $c_j(v_{qi_i})$,
- $p_{j,k}$ represents the probability of associating a tuple t_i with the sensitive value v_{s_k} in QI-group QI_j denoted by $Pr(A^s = v_{s_k}, t_i|QI_j)$, and
- $c(v_{qi_i})$ is the total number of tuples with v_{qi_i} in T^*.

The problem can be viewed as an anonymization problem in which we determine the number of QI-values that should be suppressed in each QI-group in order to guarantee an appropriate correlation constant $(1/\lambda_{v_{qi}})$. To summarize, the linear programming problem can be expressed as follows:

$$\begin{aligned} \max \quad & \textstyle\sum_{i,j} x_{i,j} \\ \text{s.t.} \quad & 0 \le \textstyle\sum_j p_{j,k}x_{i,j} - x_i \le \epsilon, \text{ if } v_{s_k} \in S(v_{qi_i}) \\ & 0 \le \textstyle\sum_j p_{j,k}x_{i,j} \le c(v_{qi_i} : v_{s_k}), \text{ if } v_{s_k} \notin S(v_{qi_i}) \\ & 0 \le x_{i,j} \le c_j(v_{qi_i}) \\ & 0 \le x_i \le c(v_{qi_i}) \times \tfrac{1}{l}. \end{aligned}$$

where,

- x_i is a variable that expresses the correlation constant te be determined during the anonymization process. We note that x_i is equal to $\frac{1}{\lambda_{v_{qi_i}}} \times c(v_{qi_i})$ such that $x_i \le \frac{1}{l} \times c(v_{qi_i})$. Figure 3 shows the set of constraints of the LP problem including variables x_i.
- ϵ is a user defined error bound.
- $c(v_{qi_i} : v_{s_k})$ is the actual correlation of v_{qi_i} and v_{s_k} determined from the anonymized table T^*

The constraints coefficients matrix is computed based on the set of constraints expressed in Figure 3.

	QI_1	...	QI_m
v_{qi_1}	$x_{1,1}$	...	$x_{1,m}$
⋮	...		...
v_{qi_d}	$x_{d,1}$	...	$x_{d,m}$

×

	v_{s_1}	...	v_{s_n}
QI_1	$p_{1,1}$	...	$p_{1,n}$
⋮	⋮		⋮
QI_m	$p_{m,1}$	...	$p_{m,n}$

(v_{qi_1}, v_{s_1})	$c_1 = p_{1,1}x_{1,1} + \ldots + p_{m,1}x_{1,m} - b_1x_1$
(v_{qi_1}, v_{s_2})	$c_2 = p_{1,2}x_{1,1} + \ldots + p_{m,2}x_{1,m} - b_1x_1$
⋮	⋮
$(v_{qi_d}, v_{s_{n-1}})$	$c_{(d\times n)-1} = p_{1,n-1}x_{d,1} + \ldots + p_{m,n-1}x_{d,m} - b_dx_d$
(v_{qi_d}, v_{s_n})	$c_{d\times n} = p_{1,n}x_{d,1} + \ldots + p_{m,n}x_{d,m} - b_dx_d$

Figure 3: Constraints for LP problem formed based on T^*

Algorithm 1 Correlation Sanitization

Require: a table T, a correlation dependency ($cd^{qi} : A^{qi} \dashrightarrow A^s$), a minimum and maximum confidence thresholds ($minConf$, $maxConf$), a minimum support threshold $minSup$, l the privacy constant and ϵ the error bound
Ensure: safe correlation for T^*
/**Pre-anonymization: determine significant sensitive values */
for each distinct v_{qi} in A^{qi} **do**
  $S(v_{qi})$={v_s | v_s is a sensitive value significantly related to v_{qi} w.r.t $minConf$, $maxConf$ and $minSup$}
  if $|S(v_{qi})| < l$ **then**
    for each v_s in $S(v_{qi})$ **do**
      Suppress $(c(v_{qi}, v_s))$ tuples with $t[A^{qi}] = v_{qi}$ and $t[A^s] = v_s$ in T
    end for
  end if
end for
$T^* = Anonymize(T, l)$
/**Post-anonymization: formalizing an LP problem */ /** 1 - Determine structural variables X for objective function $z = \sum_{i,j} x_{i,j}$ from T^* */
$cl = 0$;
for each distinct v_{qi_i} in A^{qi} **do**
  for each QI in T^* **do**
    $X[cl] \leftarrow x_{i,j}$
    Set $0 \le x_{i,j} \le c_j(v_{qi_i})$
    $cl = cl + 1$;
  end for
end for
for each distinct v_{qi_i} in A^{qi} **do**
  if $S(v_{qi})$ is not empty **then**
    $X[cl] \leftarrow x_i$
    Set $0 \le x_i \le \frac{1}{l} \times c(v_{qi_i})$
    $cl = cl + 1$;
  end if
end for
/** 2- Determine constraints coefficients matrix from T^* */
$cI = 0, r = 0, C[][] = 0$;
for each distinct v_{qi_i} in A^{qi} **do**
  $cI = i * m$;
  for each distinct v_{s_k} in A^s **do**
    $cl = cI$;
    for each QI_j in T^* **do**
      $C[r][cl] = p_{j,k}$;
      $cl = cl + 1$;
    end for
    if $v_{s_k} \in S(v_{qi_i})$ **then**
      $cl = getColFor(v_{qi_i})$;
      $C[r][cl] = -1$;
      $B[r] = \epsilon$;
    else
      $B[r] = getCorrelation(v_{qi_i}, v_{s_k})$;
    end if
    $r = r + 1$;
  end for
end for
Solve LP problem {max. $z|CX \le B$}
/**Anonymize QI-Values*/
for each QI_j in T^* **do**
  Suppress $c_j(v_{qi_i}) - x_{i,j}$ values of v_{qi_i} in QI
end for

Now that we have shown how we can guarantee the safe correlation safety constraint, we present our correlation sanitizer algorithm that ensures that the most frequent correlated values do not appear too frequently, and that the less frequent correlated values do not appear too rarely in T^*. The algorithm takes a table T, a quasi-identifier correlation dependency cd^{qi}, minimum and maximum confidence thresholds ($minConf$, $maxConf$), the minimum support

threshold $minSup$ and the error bound ϵ. It ensures the safe correlation requirement for T.

The algorithm is composed of two main tasks, pre- anonymization and post-anonymization. In pre-anonymization, from Step 1 to 8, the algorithm retrieves the set of significant sensitive values $S(v_{qi})$ for each distinct value v_{qi} in the quasi-identifier attribute A^{qi}, based on $minConf$, $maxConf$ and $minSup$. Hence, a privacy breach could occur at this level when an adversary is able to determine possible associations with sensitive values based on the size of $S(v_{qi})$. That is why the algorithm from Step 3 to 7 suppresses the tuples related to v_{qi} and v_s if $|\mathcal{S}(v_{qi})|$ is less than l.

In post-anonymization, we ensure that the probability of associating v_{qi} with any of its significantly related sensitive values $v_s \in S(v_{qi})$ is equal to $1/\lambda_{v_{qi}}$ which is achieved by solving the linear programming problem discussed in the previous section. It first retrieves the structural variables from T^* (Step 10 to 24). Each variable $x_{i,j}$ representing the count of v_{qi_i} in QI_j is bounded by $c_j(v_{qi_i})$, variable x_i expressing $\frac{1}{\lambda_{v_{qi_i}}} \times c(v_{qi})$ is determined based on the LP solution. Note that x_i is bounded by $\frac{1}{l} \times c(v_{qi})$.

In the second block of post-anonymization from Step 25 to 43, the algorithm determines the constraints coefficients matrix. In Step 31, we store $p_{j,k}$ corresponding to $Pr(t_i, A^s = v_{s_k}|QI_j)$ and associated with variable $x_{i,j}$ of column cl in the constraint coefficient matrix C. In order to guarantee safe correlation, the algorithm verifies if $v_{s_k} \in S(v_{qi_i})$ where $Pr(A^s = v_{s_k}, A^{qi} = v_{qi_i}|T^*)$ should be equal to $1/\lambda_{v_{qi}}$. In this case, the algorithm stores a -1 coefficient for variable $x_i \leq a$ corresponding to v_{qi_i} for column cl in C and the error bound ϵ in B. On the other hand, if $v_{s_k} \notin S(v_{qi_i})$, the auxiliary variable in this case is bounded by the actual correlation of v_{s_k} and v_{qi_i} as shown in Step 39.

The LP problem is solved in Step 44 such that for each QI-group QI_j, a number of $c_j(v_{qi_i}) - x_{i,j}$ is suppressed from Step 45 to 47.

Framing this as an optimization problem raises concerns of a minimality attack [26]. The safety constraint addresses this: Because of the requirement that all exposed values have equal number, the optimal suppression will always remove the more numerous values. A minimality attack will thus assume that the suppressed values are only the more common values. This would be the (presumably known) correlations; the probability of any given value being suppressed is based on its probability given correlations. In other words, the optimality of the suppression tells us that what we can estimate from the data is exactly what we would expect from just knowing the correlation.

There is still an issue of minimality attacks on the underlying anonymization method. This can be addressed through using a non-deterministic approach in Step 9. This protects against minimality attacks, as described in [3].

Let $|A^{qi}|, |A^s|$ be the number of distinct quasi-identifying and sensitive values in attributes A^{qi} and A^s respectively, the time complexity of the sanitization algorithm can be estimated by $O(m \cdot |A^{qi}| \cdot |A^s|)$ where m is the number of QI-groups in T^*.

In addition, based on the linear programming problem defined in 6.1, we can say that the sanitization algorithm scales. In fact, $\forall i, j$, if $x_{i,j}$ and x_i are equal to zero, we can easily verify that all constraints are satisfied.

7. EXPERIMENTAL EVALUATION

We now present a set of experiments to evaluate the efficacy of our approach. We implemented the correlation sanitization code in Java based on the Anonymization Toolbox [7], running on an Intel XEON 2.4GHz PC with 2GB RAM.

7.1 Evaluation Dataset

In keeping with much work on anonymization, we use the Adult Dataset from UCI Machine Learning Repository [6]. We treat *Occupation* as a sensitive attribute; other attributes are presumed to be (quasi- or actual-) identifiers.

We used cd^{qi} : *Education* $\dashrightarrow$ *Occupation* as a correlation dependency for the adult dataset containing 32561 tuples. We note that using such correlation dependency, an adversary is able to identify the occupation of an individual in the dataset according to education.[1]

In the next section, we present and discuss results obtained from running our algorithm.

7.2 Evaluation Results

We conducted a set of measurements to evaluate the efficiency of our correlation sanitization algorithm. These measurements can be summarized as follows:

- Evaluating the correlation threat after anonymization,
- Determining anonymization cost represented by the loss metric to capture the fraction of tuples that must be (partially or totally) generalized, suppressed, or encrypted in order to satisfy the safety constraints, and
- Comparing anonymization cost in two different datasets w.r.t several minimum and maximum confidence values (minConf and maxConf),

7.2.1 Correlation Evaluation

We evaluate here the remaining correlation in the dataset after a naïve anonymization using the correlation sanitization algorithm. In fact, we compare the outcome of anonymization techniques, more precisely anatomy and correlation sanitization, using a java-based implementation of Wong's approach [27]. We use in this test $l = 3, 4$ and 5 for several significant sensitive values as shown in Figure 4.

We note that in order to calculate $Pr(A^s = v_s, t_i|QI_j)$ defined in the correlation sanitization algorithm, we used the possible world model with actual correlations as shown in the example of Section 5 for the following significant sensitive values; *Handlers_cleaner*, *Craft_repair*, *Exec_managerial* and *Adm_clerical*.

As expected, the correlation sanitization algorithm bounds the correlations with confidence greater than 0.9 and lower than 0.1 while others eventually remain representing the y-axis in the Figures 4b, 4c and 4d expressing residual non-violating correlations related to non-significant sensitive values that could not be exposed.

7.2.2 Anonymization Cost Evaluation

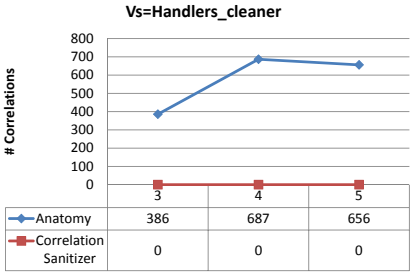
We evaluate our proposed correlation sanitization algorithm to determine the number of tuples and values that are suppressed to achieve the safety constraint. We use the following loss metric to quantify such loss of data fidelity.

[1]We invite the reader to check out [27] for more details on how to compute the global distribution and the privacy breach value for each attribute value.

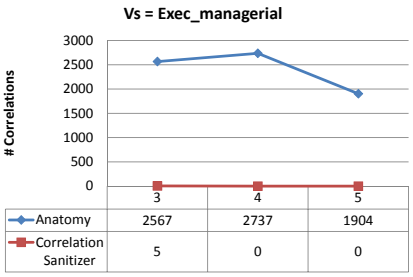
Definition 8 (Loss Metric ($\mathcal{LM}$)). *Let $g(T^*, v)$ be a function that returns the number of tuples where the value v is suppressed in the anonymization T^* of T. The loss metric ($\mathcal{LM}$) for table T^* and value v is*

$$\mathcal{LM}(T, v) = \frac{g(T^*, v)}{|T|} \quad (2)$$

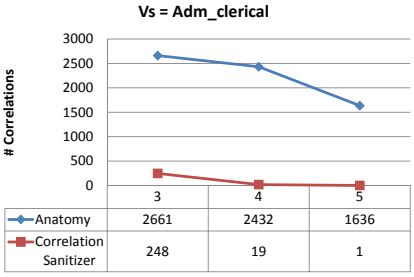
Figure 2 shows an anonymized version of table prescription where the grouping is safe. The loss metric for this anonymization has a loss metric equal to $\mathcal{LM}(Prescription, UnitedStates) = 1/3$.



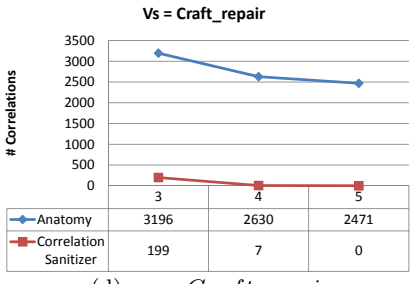
(a) $v_s = Handlers\_cleaner$



(b) $v_s = Exec\_managerial$



(c) $v_s = Adm\_clerical$



(d) $v_s = Craft\_repair$

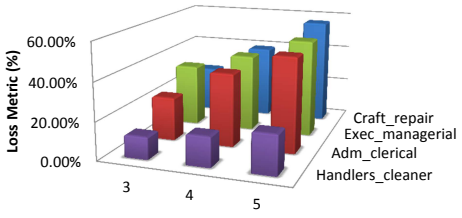
Figure 4: Correlation Sanitization for $l = 3, 4$ and 5

We applied the algorithm on table T to ensure safe correlation for values for cd^{q_i} : $Education \dashrightarrow Occupation$ with $minConf$, $maxConf$ and $minSup$ equal to 0.1, 0.9 and 0.2. Results in Figure 5 show explicitly the trade-off between privacy and utility such that for the sensitive value $Craft\_repair$ and $l = 5$, $\mathcal{LM}$ reaches 56%. At some point, we can see that the result can be dwarfed by the loss of utility. We have not identified any inherent reason why this must hold. Further research into more effective anonymization algorithms may produce techniques that meet privacy requirements while increasing the ability to learn from the data.

7.2.3 Cost Evaluation in Different Datasets

We also compared the anonymization costs computed when applying the correlation sanitization algorithm to the Adult dataset and the Bank Marketing Dataset used in [19]. In the latter, we treat *Balance* as a sensitive attribute while the remaining attributes are presumed to be (quasi- or actual-) identifiers. For computational reasons, we generalize the values of attribute *Balance* to 21 intervals to reduce the total number of distinct sensitive values. The results are shown in Figure 6 for $l = 2, 3$ and 4 with 5 different values for minConf and maxConf respectively represented in the X-axis.

Not surprisingly, the results are similar for both datasets showing that the cost increases when anonymizing the correlations. This is only to confirm as in [12] that there is a trade-off to be made at the stake of utility in order to meet strong privacy requirements. While this could be limiting to generalization techniques, it remains debatable in our approach where exact data values are maintained[2].



	3	4	5
Handlers_cleaner	11.78%	15.84%	20.73%
Adm_clerical	23.20%	39.00%	50.26%
Exec_managerial	33.12%	41.36%	51.92%
Craft_repair	24.64%	39.27%	56.67%

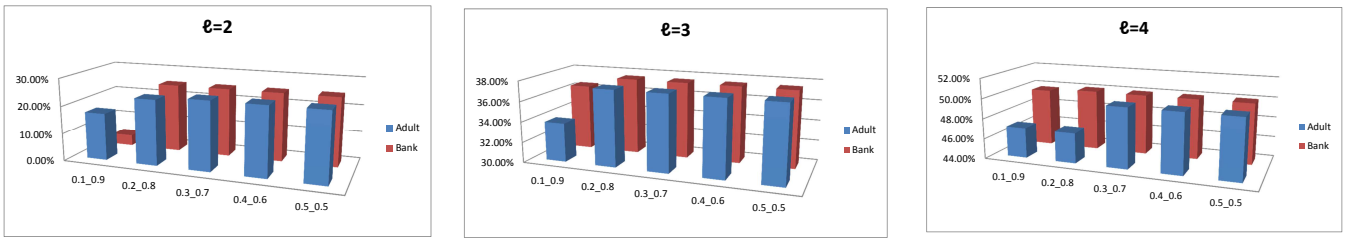
Figure 5: Evaluating loss for Correlation Sanitization

8. CONCLUSION

In this paper, we presented new methods to cope with defects of anonymization techniques resulting from unsafe data correlation. We defined a new safety constraint to deal with correlation between quasi-identifier and sensitive attributes. We provided a sanitization algorithm to ensure the safe correlation in a post-anonymization process. Finally, we showed, using a set of experiments, that there is a trade-off to be made between privacy and utility. This trade-off is quantified based on the number of tuples and values to be anonymized using anonymization algorithms.

A related problem is coping with correlations in transactional datasets where multiple tuples could be related to an

[2]Note that while suppression prevents privacy violations, it does not necessarily prevent discovery of correlations. Preliminary results on a decision tree learning approach customized to anatomized data show comparable classification accuracy to decision trees learned on the original data.



(a) Bank vs. Adult dataset with l=2 (b) Bank vs. Adult dataset with l=3 (c) Bank vs. Adult dataset with l=4

Figure 6: Correlation Sanitization for $l = 2, 3$ and 4

individual [1]. Under such assumption, a straightforward extension of safety constraint could not be achieved leading eventually to more sophisticated privacy violation detection and elimination methods. Achieving sufficient utility in such environments may also need to consider alternative privacy models such as LKC-privacy [18] or (k, m)-anonymity [24].

9. ACKNOWLEDGEMENTS

This publication was made possible by an NPRP grant 09-256-1-046 from the Qatar National Research Fund. The statements made herein are solely the responsibility of the author[s].

10. REFERENCES

[1] B. al Bouna, C. Clifton, and Q. M. Malluhi. Using safety constraint for transactional dataset anonymization. In *DBSec*, pages 164–178, 2013.

[2] D. J. Aldous. Exchangeability and related topics. In *École d'été de probabilités de Saint-Flour, XIII—1983*, volume 1117 of *Lecture Notes in Math.*, pages 1–198. Springer, Berlin, 1985.

[3] G. Cormode, N. Li, T. Li, and D. Srivastava. Minimizing minimality and maximizing utility: Analyzing method-based attacks on anonymized data. In *Proceedings of the VLDB Endowment*, volume 3, pages 1045–1056, 2010.

[4] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. Extending loose associations to multiple fragments. In *Proceedings of the 27th International Conference on Data and Applications Security and Privacy XXVII*, DBSec'13, pages 1–16, Berlin, Heidelberg, 2013. Springer-Verlag.

[5] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, pages 265–284, Berlin, Heidelberg, 2006. Springer-Verlag.

[6] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[7] M. Kantarcioglu, A. Inan, and M. Kuzu. Anonymization toolbox, 2010.

[8] D. Kifer. Attacks on privacy and definetti's theorem. In *SIGMOD Conference*, pages 127–138, 2009.

[9] D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *SIGMOD Conference*, pages 193–204, 2011.

[10] J. Lee and C. Clifton. Differential identifiability. In *The 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1041–1049, Beijing, China, Aug. 12-16 2012.

[11] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115, 2007.

[12] N. Li, W. Qardaji, and D. Su. On sampling, anonymization, and differential privacy or, k-anonymization meets differential privacy. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 32–33, New York, NY, USA, 2012. ACM.

[13] T. Li and N. Li. Injector: Mining background knowledge for data anonymization. In *ICDE*, pages 446–455, 2008.

[14] T. Li, N. Li, J. Zhang, and I. Molloy. Slicing: A new approach for privacy preserving data publishing. *IEEE Trans. Knowl. Data Eng.*, 24(3):561–574, 2012.

[15] G. Loukides, J. Liagouris, A. Gkoulalas-Divanis, and M. Terrovitis. Disassociation for electronic health record privacy. *Journal of Biomedical Informatics*, 50(0):46 – 61, 2014. Special Issue on Informatics Methods in Medical Privacy.

[16] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. *l*-diversity: Privacy beyond *k*-anonymity. *ACMTransactions on Knowledge Discovery from Data (TKDD)*, 1(1), Mar. 2007.

[17] N. Mohammed, R. Chen, B. C. Fung, and P. S. Yu. Differentially private data release for data mining. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 493–501, New York, NY, USA, 2011. ACM.

[18] N. Mohammed, B. C. M. Fung, P. C. K. Hung, and C. kwong Lee. Anonymizing healthcare data: a case study on the blood transfusion service. In *KDD*, pages 1285–1294, 2009.

[19] S. Moro, R. Laureano, and P. Cortez. Using data mining for bank direct marketing: An application of the crisp-dm methodology. In P. N. et al., editor, *Proceedings of the European Simulation and Modelling Conference - ESM'2011*, pages 117–121, Guimaraes, Portugal, Oct. 2011. EUROSIS.

[20] A. E. Nergiz and C. Clifton. Query processing in private data outsourcing using anonymization. In *The 25th IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSEC-11)*, Richmond, Virginia, July 11-13 2011.

[21] P. Ressel. De Finetti-type theorems: an analytical approach. *Ann. Probab.*, 13(3):898–922, 1985.

[22] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. Knowl. Data Eng.*, 13(6):1010–1027, 2001.

[23] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[24] M. Terrovitis, N. Mamoulis, and P. Kalnis. Privacy-preserving anonymization of set-valued data. *Proceedings of the VLDB Endowment*, 1(1):115–125, Aug. 2008.

[25] M. Terrovitis, N. Mamoulis, J. Liagouris, and S. Skiadopoulos. Privacy preservation by disassociation. *Proc. VLDB Endow.*, 5(10):944–955, June 2012.

[26] R. C.-W. Wong, A. W.-C. Fu, K. Wang, and J. Pei. Minimality attack in privacy preserving data publishing. In *VLDB*, pages 543–554, 2007.

[27] R. C.-W. Wong, A. W.-C. Fu, K. Wang, P. S. Yu, and J. Pei. Can the utility of anonymized data be used for privacy breaches? *ACM Trans. Knowl. Discov. Data*, 5(3):16:1–16:24, Aug. 2011.

[28] R. C.-W. Wong, J. Li, A. W.-C. Fu, and K. Wang. (alpha, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing. In *KDD*, pages 754–759, 2006.

[29] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *Proceedings of 32nd International Conference on Very Large Data Bases (VLDB 2006)*, Seoul, Korea, Sept. 12-15 2006.