SCHEMA-AGNOSTIC ENTITY RETRIEVAL IN HIGHLY HETEROGENEOUS SEMI-STRUCTURED ENVIRONMENTS

Von der Fakultät für Elektrotechnik und Informatik der
Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)
genehmigte Dissertation

von ing. sys. com. dipl. EPF Julien Gaugaz

geboren am 5. Januar 1979
in Morges (Waadt), Schweiz

2015

REFERENT:
Prof. Dr. techn. Dipl.-Ing. Wolfgang Nejdl

KO-REFERENT:
Prof. Dr. Heribert Vollmer

VORSITZ:
Prof. Dr.-Ing. Bernardo Wagner

TAG DER PROMOTION:
22. Juli 2015

ABSTRACT

The web is nowadays a critical information source for many actors of the society. As the amount of information exchanged on the web increased, solutions were developed that help humans accessing and assimilating information better and in a reduced amount of time. Those include standards for structured information representations like XML or RDF, as well as techniques like information retrieval and information extraction. With the advent of the social web, the number of sources of information, and thus also its diversity, further increased such that machine processing faces new challenges.

The works we present in this dissertation tackle three challenges specific to this highly heterogeneous information environment:
1) schema discovery; 2) query relaxation; and 3) propagation of entity identity revisions. All are approaches to cope with information heterogeneity without attempting in vain to impose a unique way of representing data for all information providers on the web. We present a *schema discovery* strategy that gives humans an overview of the structured data available in a given heterogeneous entity collection and that allows them to express queries. With our research in the domain of *query relaxation*, we then show how to leverage duplicate entities to effectively query heterogeneous data in such malleable schemas. Finally, we study the evolution of entities: we explore techniques for tracing and revising identities of evolving entities and analyze how to improve the *propagation of information integration revisions* to distributed information consumers.

KEYWORDS: Schema Discovery, Entity Retrieval, Heterogeneous Semi-Structured Data

ZUSAMMENFASSUNG

Das Web ist eine essenzielle Informationsquelle für viele Menschen in der heutigen Gesellschaft. Mit der Zunahme der Menge an Information, die über das Web ausgetauscht werden, wurden Lösungen entwickelt, die den Menschen helfen, Information besser aufzufinden und aufzunehmen. Dazu zählen Informationsrepräsentationsstandards wie XML oder RDF, sowie Techniken wie Information Retrieval und Informationsextraktion. Mit dem Aufkommen des Social Web haben die Anzahl an Informationsquellen, und daher auch die Vielfältigkeit von Informationsdarstellungen, stark zugenommen, sodass die maschinelle Verarbeitung von Informationen vor neue Herausforderungen gestellt wird.

Die Werke, die wir in dieser Dissertation vorstellen, widmen sich drei zentralen Herausforderungen, die spezifisch für diese sehr heterogene Umgebung sind: 1) Erkundung von Schemata; 2) Relaxation von Anfragen; und 3) Ausbreitung von Revisionen von Entitätsidentitäten. Die vorgestellten Lösungen sind Ansätze mit Informationsheterogenität umzugehen, ohne vergeblich ein einheitliches Datenrepräsentationsmodell anzustreben, das für alle Informationsquellen auf dem Web gelten muss.

Wir diskutieren eine Strategie zur *Erkundung von Schemata*, die Menschen eine Übersicht über die verfügbaren strukturierten Daten aus heterogenen Entitätssammlungen geben und sie dabei unterstützt, Datenabfragen bezüglich dieser Sammlungen zu formulieren. Mit unserer Arbeit im Bereich der *Relaxation von Anfragen*, zeigen wir wie wir duplizierte Entitäten ausnutzen können, um heterogene Daten basierend auf flexiblen Schemata abzufragen. Abschließend studieren wir die Evolution von Entitäten: wir erforschen Techniken zum Aufspüren und zur Revisionierung von sich verändernden Entitäten und analysieren wie die *Ausbreitung von Revisionen von Entitätsidentitäten* zu verteilten Informationsverbrauchern verbessert werden kann.

SCHLAGWÖRTER: Erkundung von Schemata, Abfrage von Entitäten, Heterogene Semi-Struktuierte Daten

PUBLICATIONS

CORE PUBLICATIONS

[114] Xuan Zhou, Julien Gaugaz, Wolf-Tilo Balke, and Wolfgang Nejdl. Query Relaxation Using Malleable Schemas. *SIGMOD*, pages 545–556, 2007. doi: http://doi.acm.org/10.1145/1247480.1247541. URL http://dx.doi.org/10.1145/1247480.1247541

[57] Julien Gaugaz, Jakub Zakrzewski, Gianluca Demartini, and Wolfgang Nejdl. How to Trace and Revise Identities. *ESWC*, 5554:414–428, May 2009. doi: 10.1007/978-3-642-02121-3. URL http://dl.acm.org/citation.cfm?id=1561533.1561574

[59] Julien Gaugaz, Xuan Zhou, Qingzhong Meng, Claudia Niederée, and Wolfgang Nejdl. Here is the Data. Where is its Schema? SQuaScheD: Unsupervised Schema Discovery for Heterogeneous Data. Technical report, 2015

RELATED PUBLICATIONS

[55] Julien Gaugaz and Gianluca Demartini. Entity identifiers for lineage preservation. In Paolo Bouquet, Harry Halpin, Heiko Stoermer, and Giovanni Tummarello, editors, *Proceedings of the 1st IRSW2008 International Workshop on Identity and Reference on the Semantic Web, Tenerife, Spain, June 2, 2008*, volume 422 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. URL http://ceur-ws.org/Vol-422/irsw2008-submission-7.pdf

[37] Gianluca Demartini, Julien Gaugaz, and Wolfgang Nejdl. A Vector Space Model for Ranking Entities and Its Application to Expert Search. *ECIR*, 5478:189–201, 2009. doi: 10.1007/978-3-642-00958-7. URL http://www.springerlink.com/content/x2w826v168677434/

[82] Zoltán Miklós, Nicolas Bonvin, Paolo Bouquet, Michele Catasta, Daniele Cordioli, Peter Fankhauser, Julien Gaugaz, Ekaterini Ioannou, Hristo Koshutanski, Antonio Maña, Claudia Niederée, Themis Palpanas, and Heiko Stoermer. From Web Data to Entities and Back. *CAiSE*, pages 302–316, June 2010. URL http://dl.acm.org/citation.cfm?id=1883784.1883817

OTHER PUBLICATIONS

[20] Ingo Brunkhorst, Paul-Alexandru Alexandru Chirita, Stefania Costache, Julien Gaugaz, Ekaterini Ioannou, Tereza Iofciu,

Enrico Minack, Wolfgang Nejdl, and Raluca Paiu. The Beagle++ Toolbox: Towards an Extendable Desktop Search Architecture. *Semant. Deskt. Work. ISWC*, November 2006. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.125.2754

[27] Sergey Chernov, Gianluca Demartini, and Julien Gaugaz. L3S Research Center at TREC 2006 Enterprise Track. *Text Retr. Conf. Proc.*, 2006

[28] Paul-Alexandru Chirita, Select Stefania Costache, Julien Gaugaz, and Wolfgang Nejdl. Desktop Context Detection Using Implicit Feedback. *Work. Pers. Inf. Manag. SIGIR*, 2006

[33] Nick Craswell, Gianluca Demartini, Julien Gaugaz, and Tereza Iofciu. L3S at INEX 2008: Retrieving Entities Using Structured Information. In Shlomo Geva, Jaap Kamps, and Andrew Trotman, editors, *INEX*, volume 5631 of *Lecture Notes in Computer Science*, pages 253–263. Springer, 2008. ISBN 978-3-642-03760-3

[56] Julien Gaugaz, Stefania Costache, Paul-Alexandru Chirita, Claudiu Firan, and Wolfgang Nejdl. Activity Based Links as a Ranking Factor in Semantic Desktop Search. *Lat. Am. Web Conf.*, pages 49–57, October 2008. doi: 10.1109/LA-WEB.2008.7. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4756161

[86] Enrico Minack, Raluca Paiu, Stefania Costache, Gianluca Demartini, Julien Gaugaz, Ekaterini Ioannou, Paul-Alexandru Chirita, and Wolfgang Nejdl. Leveraging Personal Metadata for Desktop Search: The Beagle++ System. *Web Semant. Sci. Serv. Agents World Wide Web*, 8(1):37–54, March 2010. ISSN 15708268. doi: 10.1016/j.websem.2009.12.001. URL http://dx.doi.org/10.1016/j.websem.2009.12.001

[32] Stefania Costache, Julien Gaugaz, Ekaterini Ioannou, Claudia Niederée, and Wolfgang Nejdl. Detecting Contexts on the Desktop Using Bayesian Networks. *Deskt. Search Work. SIGIR2010*, 2010

[60] Mihai Georgescu, Dang Duc Pham, Claudiu Firan, Julien Gaugaz, and Wolfgang Nejdl. Map to Humans and Reduce Error - Crowdsourcing for Deduplication Applied to Digital Libraries. *CIKM*, 2012

[58] Julien Gaugaz, Patrick Siehndel, Gianluca Demartini, Tereza Iofciu, Mihai Georgescu, and Nicola Henze. Predicting the Future Impact of News Events. In Ricardo Baeza-Yates, Arjen de Vries, Hugo Zaragoza, B. Barla Cambazoglu, Vanessa

Murdock, Ronny Lempel, and Fabrizio Silvestri, editors, *ECIR*, 2012

ACKNOWLEDGMENTS

First and foremost, I would like to thank Prof. Wolfgang Nejdl for having given me the opportunity to join the L3S Research Center. L3S is an inspiring research environment providing great possibilities of research and gathering a wide spectrum of competencies.

I thank Prof. Heribert Vollmer for his time, being on my PhD committee.

I also would like to thank Xuan Zhou, Claudia Niederée and Peter Fankhauser for their kind advices and useful and interesting discussions.

I thank my current employer XING AG for giving me the opportunity to put at good use my skills acquired during those PhD years, and my colleagues from the Data Science team for their understanding while I finished writing my dissertation next to my day job.

I would like to thank Gianluca Demartini and Avaré Stewart for sharing an office with me. I greatly appreciated all the time we spent together, and the many passionate discussions.

I thank whole-heartedly the institute and university staff and in particular Katia Capelli, Franziska Pfeffer, Iris Zieseniß, Angelika van Agen, Marion Wicht, Anca Vais and Willi Meyer; for their work of course, without which no research would be possible, but also for their kindness.

And finally, and dearest, I would like to thank my friends and family for their patience and outlasting support throughout those passionate years.

CONTENTS

1 INTRODUCTION

The internet is a formidable medium of communication allowing for instantaneous access to an ever-growing amount of information. The hypertext transfer protocol (HTTP) was elaborated in the early 1990's to allow the retrieval of hypertext documents (HTML) consisting mainly of text, images and hyperlinks, and intended primarily for human consumption. As the number of information increased, it became necessary to develop techniques to facilitate information retrieval. Those were first manually maintained catalogs of web pages, then, later, the first web search engines used inverted indexes (words as keys and documents as values) together with other techniques to allow to *search* for documents given a short keyword query. This approach to search for information on the web has been further developed over the years, with the famous introduction of PageRank[18] in 1998 giving birth to Google.

In parallel to the evolution of keyword based information retrieval, techniques and formats were developed to facilitate the processing by machines of the information available on the web. The ultimate goal of machine processing of web information is to allow humans to find and assimilate better information in a reduced amount of time. Standards for representation of *structured information* facilitating machine processing include XML and RDF, but also more trivial ones like comma separated values and spreadsheet files. Even though those are widely used, an important amount of information remains, primarily intended to humans and thus more difficult to process machinally. Those include of course text and images, but also video and audio data.

The years 2000 saw the advent of the *social web* characterized by the increase of user-contributed where information material on the web are no longer solely contributed by few publishers but also by an ever-increasing number of users publishing themselves information through platforms including forums, blogs, wikis and social networks. Examples include Wikipedia, Twitter, Facebook or Blogger, but also online retailers like Amazon where users can write comments and reviews about the articles offered. In order to render this information intended to human more easily machine processable, *information extraction* techniques were developed where different types of natural language texts can be transformed into more structured formats such as RDF.

Structured information representation standards and information extraction techniques enable machines to better process the information available on the web by allowing to identify entities and their properties that are described online. However, following the dramatic increase of the number of information sources, the *diversity* of information representations becomes itself a challenge requiring integration of this information to facilitate machine processing. *Information integration* of two sources of structured information like the database management systems of two merging companies is a well-studied problem and techniques exist that assist humans in this task. However, the integration at web-scale of highly heterogeneous information constantly produced by a high number of independent sources is new and presents challenges of different nature. We detail below some of the challenges that we address in this dissertation and outline its structure.

1.1 CONTRIBUTIONS

SCHEMA DISCOVERY In Section 2.1.5 we explain how schemas for structured information are useful to humans not only to know how to describe new information, but also as a summary of the information available and as a guide to express queries for this information. Schemas are also the basis for most existing information integration techniques. The increase of the number of structured information sources on the web leads to the dramatic increase of the number of schemas, whose purpose is canceled out by their multiplicity. In some cases a schema does not even exist or is not available. For this reason the first challenge for better machine processing of information in highly heterogeneous environments is to discover a schema for a given collection of structured entity descriptions.

In Chapter 3 we present an approach to discovering entity types and their associated attributes when the schema is non-available or non-existent for semi-structured data presenting a high degree of heterogeneity like the kind of data introduced above and further detailed in Section 2.2. In this chapter, we show how an information theoretic measure named Minimum Description Length (MDL)—a principle for inductive and statistical inference model selection—can be used as an objective quality measure to guide a schema discovery process. We devise an algorithm based on spectral graph clustering guided by this schema quality measure to carry out schema discovery efficiently.

QUERY RELAXATION With the help of a discovered schema, a human can have an idea of what kind of entities are described in the data; and she can browse the entity types revealing most common attributes and particular instance entities. Knowing what is in a dataset and how the entities are represented allows to express queries. However the high heterogeneity among attributes is problematic and leads to the second challenge: for the user to express queries, i.e., to refer to a particular object's property, the user has to express all duplicated attributes representing this property which is time-consuming and error-prone.

In Chapter 4 we show how we can exploit the dependency between entity and attribute duplications to devise a query relaxation scheme, thus allowing the user to issue more succinct queries avoiding the repeated mention of duplicated attributes. Our scheme utilizes duplicates in differently described data sets to discover the correlations among attributes, and then uses these correlations to appropriately relax the users' queries. In addition, it ranks results of the relaxed query according to their respective probability of satisfying the original query's intent.

PROPAGATION OF ENTITY IDENTITY REVISIONS In a highly heterogeneous and evolving information environment as the web of today, information integration decisions about entity descriptions have to be constantly revised: two distinct entity descriptions with different identifiers have to be merged because they describe the same entity, and conversely, one entity description has to be split in two or more descriptions because it ambiguously refers to more than one distinct real-world entity. It is impossible to make final information integration decisions about web entities, mainly for two reasons: 1) new information about the entities is constantly made available, so that merge or split decisions have to be revised; and 2) the entities themselves evolve over time, implying that new descriptions of the same entity might appear at any time. The last challenge is then to propagate those entity identity revisions to the agents consuming this information from all over the web, while minimizing the communication effort required to do so.

In Chapter 5 we propose a solution which relies on labelling the IDs with additional history information. These labels allow clients to locally detect deprecated identifiers they are using and also merge IDs referring to the same real-world entity without needing further communication or processing to resolve the identity conflict. We investigate how much this lineage labeling reduces the communication overhead and how this impacts the uniqueness of the identifiers in the consumers' local information space.

In the next chapter we introduce the concepts and vocabulary necessary to understand in detail the problems addressed in this dissertation and describe in further details the type of data considered. In the subsequent chapters we present our contributions to the solution of the challenges exposed above; and we conclude with ideas for future works and an overview of further academic contributions.

2 PRELIMINARIES & VOCABULARY

We define hereafter the vocabulary and concepts necessary to adequately speak of the matters addressed in the dissertation, and describe later herewith the nature of the data considered.

2.1 PHILOSOPHICAL BACKGROUND & VOCABULARY

Due to the fundamental nature of some of the subjects addressed here, experienced by all humans, the literature in philosophy and linguistics on the subject abounds in quantity and diversity, and spans a long time reaching back to ancient Greece. A rigorous overview of the subject is way out of the scope of this dissertation. Instead we share below our personal view on what a web entity is, what is machine processing of web entities and where discrepancies among web entities come from, with the goal of better explaining the challenges addressed and their solutions proposed in the next chapters of the dissertation. The concept exposed are thus not from a particular work or author, but the followings works were sources of inspiration:

[74] George Lakoff. *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. University of Chicago Press, 1990. ISBN 9780226468044

[91] Charles Sanders Peirce. On a New List of Categories. In *Proceedings of the American Academy of Arts and Sciences*, volume 7, pages 287–298, 1868

[98] Howard Robinson. Substance. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2014 edition, 2014

[109] Linda Wetzel. Types and tokens. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2014 edition, 2014

2.1.1 *Information and its Expression*

Humans experience their environment through their senses: sight, hearing, taste, smell and touch. Charles S. Peirce calls "substance" this immediate experience of the environment through the senses:

> "This conception of the present in general, or *it* in general, is rendered in philosophical language by the word *substance*." [91]

The substance is the raw, uninterpreted experience of the present, in general, and as such does not have a proper unity. Thanks to our capacity of abstraction we recognize *objects* in this always-evolving substance. This is not trivial: consider observing a small field mouse. The substance is continuously changing all the time, such that there is no two moments where the substance is exactly the same. And yet, among this raw, all-enclosing sensory experience, we recognize one mouse. Even though the mouse might be moving around sniffing, we recognize the mouse at two different moments in time to be the same. This abstraction we call *object*.

Definition 1 Object. *An object is any thing that one can think of, any thing that can be* object *of thought.*

An object is a construction of the mind and cannot, as such, be communicated to other humans. To do so requires to *express* the object in a way that can be sensed by others. This expressible representation of an object is what we call an *entity*.

Definition 2 Entity. *An entity is a representation of an object, possibly including representations of some of its properties—as defined below.*

In its simplest form, an entity is a simple token, representing an object. Examples are "mouse", "Maus" or "souris" all representing the same object: the abstract concept of a mouse. An entity can also include the expression of some of the object's qualities: in the statement "the mouse is white", the entity "mouse" is associated with the entity "white" expressing the object's *whiteness*. The expression of an object's quality is called a *property*.

Definition 3 Property. *A property is the expression of an object's quality. This is realized by the* association *of the object with one or more other objects.*

The simplest form of an entity consisting of only a token can be seen as the statement of a simple property: "the object is represented by the token *mouse*", expressing that the object has the quality to be represented by the token.

2.1.2 *Abstraction and Categories*

As we defined it above, every object is an abstraction. When we recognize objects out of our sensory experience—the substance—we abstract the fact that the substance is impermanent and ever-changing in order to recognize one object. Continuing our previous example of the mouse: we abstract the ever-changing position, shape and color of the mouse to recognize the mouse as one and the same object.

Definition 4 Abstraction. *Abstracting consists in removing some properties of one or several objects in order to generalize them and thus recognizing them as one. The result of an abstraction is an object and is itself also called an abstraction.*

Definition 5 Super-/Sub-ordinate. *If an object* a *is an abstraction of another object* b*, we say that* a *is a super-ordinate of* b *and, conversely,* b *is the sub-ordinate of* a.

Since a super-ordinate object a is an abstraction of properties from a sub-ordinate object b, every property of a is also a property of b for all a super-ordinate of b; the converse is however not necessarily true.

Given a domain of discourse, it is useful to define a lower bound on the level of abstraction of the objects considered. Where this is the case we make the distinction between those objects at the lowest level of abstraction that we call *instances* and other, more abstract, super-ordinate objects called *categories*. We say that an instance belong to a category if the category object is an abstraction over the instance object. There can be a super-/sub-ordinate relationship between categories, whereas this is usually not the case between instances. It is worth noting that the distinction between instances and categories depends on the domain of discourse, i.e., on the context in which the object are represented and mentioned. As a consequence an object considered as instance in one context can be considered as category in another context. This distinction is explored extensively in philosophy where instances and categories are usually named *tokens* and *types*.

2.1.3 *Nature of Entities*

An entity representing an object can take various forms, and the number of distinct object representations (entities) for a given object is potentially infinite: since an entity serves as the communication of an object and its properties between a source and a sink, for an entity to be a representation of a given object, it is sufficient that the source and the sink of the communication agree on the entity representing the object. And since there is a priori no limit on the number of entities the source and the sink can agree on, there is also no limit on the number of entities potentially representing a given object.

Humans have a long usage history of various kinds of object representations: the first object representations were gestures that Australopithecus (4-2 millions years ago) likely already used [48]. Vocal object representations appeared then at least 2-100,000 years ago with human speech. Oldest known pictorial representations, cave paintings, date back 40 thousand years [93]. And, defining the beginning of History, written object representations appeared after 6,000 BC [81].

Since the invention of the integrated circuit in the late 1960's [100] and the democratization of the internet in the 1990's, computers and internet play an ever increasing role in representation, transmission and processing of information in general, and of entities in particular.

2.1.4 *Entities on the Web*

This dissertation is concerned with entities on the Web and their processing by machines. Definitions 1 and 2 define an entity as a representation of an object, which is itself an object of thought. An object is a pure construction of the mind, and as such doesn't occur on the web. We find on the web only *representations* of objects, that is, entities. The kinds of entities present on the web and processed by machines differ from the kinds of entities we find elsewhere only by the *digital* nature of the former: pictorial, vocalized, symbolized—written. In the context of the web, those kinds of entities are more commonly referred to as image, audio, video and text.

We can distinguish two forms of machine processing of those digital entities: 1) transmission; and 2) computation. The *transmission* of entities is the communication of information about objects from one computer to another, or from one computer program to another, for the purpose of representing this information to a human or for further machine processing. Entity *computation* refers to machine processing of entities for other purpose than communication. There exists many tools to process digital entities, ranging from general purpose programming language like C or Java, to languages specialized for data management, like SQL. The number of possible operations specified by means of those processing languages is unbounded, but some operations are more fundamental and underlies many others. As examples of machine processing of web entities, we describe further below the following operations which we believe are fundamental: equality, grouping, ordering and aggregation. Note that this list is for example purpose only, and is by no means sufficient to perform all possible machine processing tasks.

Equality is a function determining whether two entities represent the same object:

$$\begin{aligned} eq : E \times E \rightarrow \quad & [true \,|\, false] \\ eq(e_1, e_2) \mapsto & \begin{cases} true & \text{if } e_1 \text{ and } e_2 \text{ represent the same object;} \\ false & \text{otherwise.} \end{cases} \end{aligned} \tag{2.1}$$

The equality operation is the basis for another important operation: grouping. The grouping operation arranges entities into groups such that all entities of a group maps to an entity representing a same object, for a given mapping function. The equality operation is necessary to determine to which group an entity belongs.

Ordering is a function imposing an order on a given set of entities:

$$
\begin{aligned}
<: E \times E \to \quad & [\mathrm{true}|\mathrm{false}] \\
e_1 < e_2 \mapsto & \begin{cases} \mathrm{true} & \text{if } e_1 \text{ represents an object strictly smaller} \\ & \text{than the object represented by } e_2; \\ \mathrm{false} & \text{otherwise.} \end{cases}
\end{aligned} \tag{2.2}
$$

Aggregation is a class of functions producing a *summary* entity out of a set of entities. The input set of entities is typically a group output from the grouping operation, and the output is one entity—or possibly another group of entities—summarizing the input. A specific example of an aggregation function would be the arithmetic sum.

Those basic operations are trivial to implement on textual entities that are themselves the object they represent, like arithmetic objects such as natural numbers. They are however more difficult to implement on other entities, like ones described in natural language in textual or audio form, or entities appearing in image or video format. Therefore, before a collection of entities can be machine-processed, it is necessary to *parse* those entities, i.e., transform an entity from one format into another format that is more efficiently machine-processable.

Many formats exist that are—more easily—machine-processable. The most common ones are simple tables like the ones found in HTML on web pages or spreadsheets, the relational model[31], entity-relationship model[26], XML and RDF. Those formats all share two characteristics: 1) they allow a machine to separate one entity from another in a unambiguous way; and 2) they express properties of an entity by a tuple consisting of a property category, called *attribute* and an *attribute value* which is itself another entity. The former is a prerequisite for any machine processing: before a machine can compute the equality operation for example, it must be able to distinguish one entity from another; that is, it has to be able to *count* the entities present in the collection considered.

Definition 6 Countable Entity. *An entity is said to be countable if it can be unambiguously and deterministically separated from other entities by a machine.*

The latter allows to distinguish between an entity's properties thanks to the attribute and to compute operations on the attributes' values.

Definition 7 Attribute. *An attribute is the representation of a category of properties; that is, an abstraction over properties expressing a quality of a category of objects.*

Note that an attribute is the representation of a category, and a category is an object; therefore an attribute is the representation of an object and hence itself an entity.

Definition 8 Attribute Value. *An attribute value is an entity associated with an attribute. An attribute value together with the corresponding attribute express an object's property; that is they represent an object's quality.*

Entities consisting of a set of attribute and value pairs are called *structured*. Structured entities are by definition countable. Entities that are not structured are called *unstructured*, and unstructured but countable entities are called *literal*.

To better understand the concepts defined here it is useful to precise the relation between them and the entities and relations as understood in the entity-relationship model. Entities and relations are defined as follows by Chen in [26]:

> "An *entity* is a 'thing' which can be distinctly identified. A specific person, company, or event is an example of an entity. A *relationship* is an association among entities. For instance, 'father-son' is a relationship between two 'person' entities." [26]

In *our* definitions above we do not make the distinction between entities and relationships: a relationship is an entity, and an attribute is what *relates* entities between each others, whether they are relationships or not. We intentionally avoid making the distinction between an entity and a relationship because this distinction is a modeling decision, that is, in our terms, just ways to represent a same set of objects with different entities, as Chen mentions it in a footnote to the previous citation:

> "It is possible that some people may view something (e.g. marriage) as an entity while other people may view it as a relationship. We think that this is a decision which has to be made by the enterprise administrator. He should define what are entities and what are relationships so that the distinction is suitable for his environment." [26]

2.1.5 *Schema*

The purpose to express entities in a structured way as a set of attributes and their values is to allow humans to instruct machines how to process those entities using query languages such as SQL or SPARQL [1] for example, or using more general purpose programming languages such as Java or C. Abstraction—the ability to refer with one entity to multiple other more specific ones—is the key characteristic allowing to describe processing operations using entity types and attributes. For this reason, attributes are usually expressed as literal entities: representations of the property abstraction on which the equality operation can be computed by exact matching, which is most efficiently machine computable. This is usually achieved by using the same textual entity as attribute for all entities sharing a property in the category represented by this attribute.

A set of attributes allowing to efficiently compute the equality function are called *key attributes*. Note that, given an entity, different sets of attributes can serve as key attributes; notably, if two attributes are individually key attributes, both attributes together are also key attributes.

In the same way attributes help to instruct machines how to process categories of properties, *entity types* help instruct machines how to process categories of entities.

Definition 9 Entity Type. *An entity type corresponds to a category and specifies the attributes to be used to describe entities representing objects belonging to the category. An entity type can be associated to an entity representing the category it corresponds to. This associated entity is called the entity type's* label.

A *schema* defines entity types to represent the objects part of the domain considered and specifies how to describe and interpret literal entities. The collection of entities to which a schema is assigned is called the *population* of the schema.

Countable entities as defined in Definition 6 complying to an available schema are said to be *structured* and countable entities for which no schema is known or exists, or which do not comply to the schema, are called *semi-structured*.

We list below the purposes of a schema for humans and for machine, inspired by [87, 94, 36]. A schema has the following purposes for *humans*:

EXPRESSION GUIDE Guide about how to express structured entities for a given domain. This is typically useful when filling web or other forms as interface to databases or other structured entity repositories. This includes what literal entities to use for expressing attributes or how to represent attribute value entities: must it be a literal entity or the value of a key attribute and which key attribute?

QUERY GUIDE Guide about how to query or program machines to process the schema's population. In particular the entity type labels and literal attributes are used by a human programmer to refer a collection of entities.

SUMMARY Summary about what kind of entities the schema's population contains. This is essential and complementary to the *query guide* function: if the query guide indicates humans *how* to refer to entities, the summary function indicates *what* there is to refer to.

In addition, a schema has also the following purposes for *machines*:

EQUALITY DEFINITION A schema can define, in a way that is interpretable by a pre-programmed entity repository like a relational database system, how to compute equality between entities.

STORAGE SPACE OPTIMIZATION The availability of the possible attribute literals, and possibly of constraints on the attribute values, an entity storage system can optimize the storage for space.

PROCESSING SPEED OPTIMIZATION The schema allows to trade space for processing speed by building indexes which allows faster selection of the entities. It also allows for automatically optimizing query execution plans as is typically the case in processing of SQL queries in relational database systems.

2.2 NATURE OF THE DATA CONSIDERED

2.2.1 *Open Information Extraction*

The overwhelming majority of the information on the web is written by and intended to humans and therefore naturally in textual and unstructured form. In order for a machine to process this information for tasks other than transmission and display, it is necessary to *parse* this unstructured textual information into structured or semi-structured entities.

Information Extraction (IE) is the task consisting of *parsing* unstructured textual information into structured entities[1]. Information extraction uses supervised machine learning techniques, such as Hidden Markov Models [50, 102], Rule Learning [103] or Conditional Random Fields [92]. Machine learning models are trained on natural language processing (NLP) features with manually annotated examples. This traditional information extraction is domain specific and the entities extracted are structured in the sense that the attributes and entity types are predefined and thus form a schema.

The need of manual training example annotation for information extraction makes it inappropriate to extract information at web-scale where the domain is unbounded and the information to extract unknown a priori. To address those issues Banko et al. proposed in 2007 the task of *Open Information Extraction* [9] consisting in parsing the entity attributes in addition to the subject and object of the attribute.

1 In the IE literature, the information extracted from text is mostly called "relation" in reference to the relational model, and the word "entity" refers to a *named entity*, i.e., a noun phrase in English grammar. An IE "relation" is a type of what we name an entity, due to the broad definition of the latter that we adopted.

Open information extraction takes in a text document and produces a collection of tuples of the form $\langle e_1; \mathtt{att}; e_2 \rangle$ where e_1 and e_2 are entities occurring in noun phrases and $\mathtt{att}$ is the attribute. Below are output examples from the OLLIE open information extractor presented in [80]:

1. After winning the Superbowl, the Saints are now the top dogs of the NFL.

   ⇒ ⟨ the Saints ; win ; the Superbowl ⟩

2. There are plenty of taxis available at Bali airport.

   ⇒ ⟨ taxis ; be available at ; Bali airport ⟩

3. Microsoft co-founder Bill Gates spoke at...

   ⇒ ⟨ Bill Gates ; be co-founder of ; Microsoft ⟩

4. Early astronomers believed that the earth is the center of the universe.

   ⇒ ⟨ the earth ; be the center of ; the universe ⟩

5. If he wins five key states, Romney will be elected President.

   ⇒ ⟨ Romney ; will be elected ; President ⟩

The seminal paper of Banko et al. presented TextRunner which works in two steps. First, a grammatical parser is used to train a Naive Bayes model classifying candidate tuples $\langle e_1; \mathtt{att}; e_2 \rangle$ as *trustworthy* or not. Second, candidates tuples are generated using heuristics based on part-of-speech (POS) tags and other simple textual features, and keeps the tuples labeled as *trustworthy*. This approach allows to perform open information extraction without a grammatical parser in the Second phase, which is interesting for scalability since grammatical parsers are significantly slower compared to a Naive Bayes classifier.

Following up, ReVerb [46, 45] gets rid of the initial learning step and uses shallow syntactic processing to extract tuples. WOE [111] uses Wikipedia infoboxes to learn extraction patterns. OLLIE [80] uses ReVerb to learn how to map dependency trees to open IE tuples thus increasing the number of attributes that can be extracted. OLLIE also adds contextual analysis, extracting attribution like in sentence #4 in the example list above and conditional modifiers (if, when, although, because, ...) like in sentence #5.

Data resulting from open information extraction is semi-structured in the sense that it is machine countable and that, clearly, no particular schema is ever used or defined: attributes used in the extracted data are the result of processing of human contributed and human intended natural language. Each entity resulting from open information extraction has two attributes and their corresponding value: 1) an identifying attribute whose value is the subject entity e_1 of the extracted relation; the attribute itself is implicitly specified by entity e_1 being the subject of the extracted relation; and 2) the extracted attribute att whose value is e_2 and object of the extracted relation. This naturally results in a proliferation of the representations of a same object in as many two-attributes entities as there is relations extracted about this object. Those two-attributes entities could be equated and automatically merged based on the value of the identifying attribute (the subject of the extraction relation). Those identifying entities present however two main challenges: duplication and ambiguity.

Entity *duplication* refers to the representation of one object by entities that are not automatically equatable. This happens often in natural languages to render communication faster: humans are good at contextualization and can therefore easily infer implicit information from this context. For example, in a political television show taking place in the US in 2014, the two entities "the president" and "Barack Hussein Obama II, born August 4, 1961, Honolulu, Hawaii, United States" would certainly be recognized to represent the same object by most humans—at least the ones susceptible to watch the show—whereas it is challenging for a machine to assert those two entities equal. The shorter time it takes to communicate the first entity than the second largely compensates the time overhead—if any—a human needs to appropriately recognize the object represented by "the president" *given the context*.

Entity *ambiguity* refers to the representation of *distinct* objects by the same entity. It happens similarly often than entity duplication for the same reason: given another context than the one described in the previous example, the same entity "the president" would represent another object than "Barack Obama." For example if the context is that of a political show in France, where the president is as a central political figure as in the USA, the entity "the president" would undoubtedly be recognized as representing the French president François Hollande.

Both entity duplication and ambiguity in natural language are the result of *omission* of some attributes in the entity describing an object in order to increase communication speed. Note the distinction between omission and abstraction described in Section 2.1.2: abstraction applies to objects and consists in *removing* some properties from the object in order to generalize it, whereas by omission the object intended stays the same but attributes are removed from the entity representing the object in order to reduce communication time and effort. In natural language the goal is to reduce communication time and effort for humans, which might be an inadequate optimization for machine processing.

2.2.2 *Web Tables*

Tables are a common way to present structured entities to humans and are naturally found on the web, in an HTML-encoded form or as images. In their work on WebTables [24, 22] Cafarella et al. "applied an HTML parser to a multi-billion-page crawl of the English-speaking web to obtain about 14.1B instances of the `table` [HTML] tag" [24]. Using a classification algorithm they could estimate that 1.10% of those tables present semi-structured entities[2], which is a small proportion but still represents an important absolute number of 150M according to their estimations, representing a formidable source of data for further machine processing. Adelfio et al. [2] improved the table recognition ratio and applied their approach to spreadsheets as well. They concluded that the number of semi-structured entities in tables is likely to be higher.

In their work on Octopus [23], following up on WebTables, Cafarella et al. proposed to *search* for relevant tables that can then be *contextualized* by adding attribute values from the text surrounding the table, and to *extend* the tables by finding other tables that can potentially be joined with the table considered. Part of this work is available to the public today in the Google Tables[3] and Google Fusion Tables[4] products.

Elmeleegy et al. [42, 43] showed that collections of entities can be extracted from HTML lists found on the web. The extracted entities are not quite structured because they don't have attributes, but only attribute values, corresponding to tables without column headers. Attributes can be assigned using class labeling techniques [104, 77, 106], thus yielding proper semi-structured entities as we defined them.

For a survey on table processing see [44] where Embley et al. report the state-of-the-art regarding detection and recognition of tables from a range of different input media—paper or electronic—and formats, and review the potential usage of table processing.

Entities described in web tables are destined to humans, but differ from entities described in natural language as considered by open information extraction by their relative structured nature. In this sense they can be considered as structured data intended for humans. But they are not structured—nor semi-structured—for machines in the sense that they are not unambiguously and efficiently countable, neither are their attributes and respective values; hence the need to *parse* them to render them machine-countable. Such entities extracted from web tables might have literal attributes in case the table has headers

2 They call those tables *relational* in reference to the relational model introduced by Codd [31] and they correspond to our definition of a semi-structured entity as a set of attribute-value pairs.

3 `http://research.google.com/tables`

4 `http://www.google.com/fusiontables`

and those can be extracted; in which case entities extracted from a same table share the same structure and attributes.

Table headers and attribute value entities—cell content—are produced by and intended for humans and thus present characteristics similar to entities in natural language. For this reason, entities extracted from web tables when considering at scale present characteristics similar to entities resulting from open information extraction: an inherent absence of schema, resulting in entity duplication and ambiguity, including for the attributes and their values.

2.2.3 *Social and Linked Data as a Whole*

Open Information Extraction and Web Tables extraction can be applied to traditional web pages, and also to contributions on more recent Social Web platforms like Wikipedia, Twitter, Facebook or Blogger. The social nature of those relatively new supports of information implies that the entities represented there come from an extremely high number of sources: before the advent of the Social Web, the information published on a web site would have a limited number of sources, i.e., the humans authoring information were relatively small and their publication were in most cases coordinated, thus limiting the heterogeneity. This is less true for Social Web platforms, where, by their very nature, a high number of authors publish, in a completely independent manner. This implies that the heterogeneity of the semi-structured information extracted by techniques mentioned above becomes greatly increased.

Some Social Web platforms support direct publication of (semi-) structured entities. Wikipedia for example allows to publish semi-structured information in its Infoboxes. The entities in Wikipedia's Infoboxes are not as highly heterogeneous as for Open Information and Web Tables Extractions as mentioned above, since a system of templates help contributors to re-use similar attributes and entity types. However the templates themselves are heterogeneous, and, to allow a greater liberty of expression, no mechanism exists that would enforce an Infobox to comply to any template. Freebase is also a Social Web platform where everybody can contribute structured information. Freebase's schemas are however more controlled and enforced than Wikipedia's Infoboxes, but still allows duplicate and ambiguous entities and attributes.

The Semantic Web initiative of the W3C encourages the publication of RDF (semi-) structured information on the web, along to traditional HTML documents. Good practice wants that an RDF instance document is also assorted with a schema (in RDFS or OWL). However, the high number of sources of RDF data implies unfortunately also a high number of schemas and similar heterogeneity problems (duplication and ambiguity) can be encountered when considering the Semantic

Web as a whole or when considering the results of Semantic Web search engines like Swoogle[5] or Sindice[6] that can retrieve entities using various schemas that can present heterogeneous characteristics.

The highly heterogeneous semi-structured data described in those sub-sections is the kind of data presenting challenges addressed in this thesis, which we develop in the following chapters.

5 http://swoogle.umbc.edu
6 http://sindice.com

3 UNSUPERVISED SCHEMA DISCOVERY BASED ON SCHEMA QUALITY

3.1 INTRODUCTION

As introduced in Chapter 2, schemas play a crucial role in management and consumption of structured data. To humans, a schema provides not only a summary about *what* can be found in the data, but also a guide on *how* to retrieve specific information from the data. To machines, an appropriate schema provides important information to enable efficient storage management and query processing. This is well-known in the case of relational database. Relational schemas, i.e. definitions of tables and their columns, are the key references for users to construct SQL queries and to explore a database. They are also important information for DBMS to perform effective query optimization.

Autonomy of structured data creation, ease of data publication and advance of text processing technologies like automated extraction have led to an explosion of the availability of structured data on the Web. However, due to their origin a substantial part of this data lacks a well-defined and well-understood schema that would enable users and machines to efficiently make use of it. This phenomenon is especially notable on the Web:

Open Information Extraction. The overwhelming majority of the information on the Web is written by and intended to humans and therefore naturally in textual and unstructured form. In order for a machine to process information efficiently on the Web, Information Extraction (IE) is widely applied to transform this textual information into structured entities. IE is traditionally designed for domain specific applications. It assumes predefined data schemas and relies heavily on supervised machine learning. This prevents it from scaling on the Web, where the domain is unbounded by nature and the information to extract therefore unknown a priori. To address these issues, Banko et al. proposed the task of Open Information Extraction [9], which aims to transform a textual document into a collection of triples of the form $\{e_1; \mathtt{att}; e_2\}$, where e_1 and e_2 are entities and $\mathtt{att}$ represents an attribute. Without a predefined schema, Open IE techniques [9, 111, 80] enable us to extract large amount of structured data from the Web. However, lacking a schema, the data generated by Open IE also pose a great challenge for various applications to put it to good use.

Web Tables. Tables are a common way to present structured entities to humans and are naturally widely found on the Web, in an HTML-encoded form or as images. A variety of methods [22, 23, 2] have been proposed to harvest the tables on the Web, and compose them into a common knowledge repository. As Web tables are created by different actors and for different purposes, they are usually highly heterogeneous in their structures and terminologies. When combined into a single repository, they lack a common schema for users to query and explore the data.

Collaborative Content Authoring. In recent years, Web 2.0 applications have created various platforms for Web users to contribute contents. Wikipedia is a very successful case. However, as most content authors do not consistently refer to common structures when creating the content, the resulting data lacks a well defined schema. For instance, the Infoboxes of Wikipedia are intended to provide structured information about real-world entities, visible as a table on Wikipedia articles. While Wikipedia provides various meta-data *templates* for content authors to create Infoboxes, the authors did not succeed to comply with it, due in part to the lack of coherency of those templates. As a result, the team of DBpedia [7] had to resort to laborious manual processes to map the Infoboxes to predefined schemas.

Considering the scale and the heterogeneity of the data in the aforementioned scenarios, manual schema construction does not seem realistic. It is desirable that a machine can automatically induce the schema from the data itself, without any interference or guidance from human. We call such an automated process *unsupervised schema discovery*.

To construct a schema for a given dataset, some common activities include categorizing the various data items into classes, characterizing each class using a set of attributes and establishing specialization relationships between the classes. A wide spectrum of choices exist for each of the activities: one can choose between a fine-grained categorization and a coarse one; when characterizing a class, one can exhaustively list all possible attributes or select only the most representative ones. As a result, a huge amount of syntactically valid schemas can be generated for a given dataset, while not all the schemas are "good" and provide effective means for digesting the data. Therefore, unsupervised schema discovery is typically faced with two fundamental questions: 1) What constitutes a good schema given a dataset? And 2) how to generate a good schema automatically?

In this chapter, we propose and answer for both challenges. We apply the idea of Minimum Description Length (MDL) [10, 65] to measure the quality of a schema, and then use this schema quality measure to find a good schema given a dataset. MDL is a principle for inductive and statistical inference model selection, which says that the best model is the one that requires the least space for encoding itself and the data. Practice shows that the principle is able to achieve an appropriate trade-off between the generality and the descriptive power of a hypothetical model. A schema can be regarded as an encoding model for its data. Along the lines of the MDL principle, we propose a schema quality measure which aims at encoding an entire dataset, including the schema itself, using the least space, that is, we aim at minimizing the description length of the schema and the dataset. Our experiments show that this quality measure is able to effectively guide unsupervised schema discovery in identification of good schemas.

Given the quality measure based on MDL, schema discovery can be regarded as an optimization problem, i.e., finding a schema of optimal quality. This problem turns out to be non-trivial, especially when the target dataset is big and heterogeneous. Thus, we propose a method called SQuaSchеD [1] for fully automatic schema discovery. Given a dataset about real-world entities, SQuaScheD aims to automatically identify the entity types, the representative attributes of each entity type and the specialization relationships between the entity types. The output of SQuaScheD is a hierarchical conceptual schema, which can support users in navigating through the data or in formulating structured queries. To maximize the quality of the resulting schema, SQuaScheD applies spectral graph clustering, which can accurately cluster both entities and attributes into conceptual categories. We have evaluated SQuaScheD on Wikipedia Infoboxes, which contain a million of data instances and thousands of entity types. The results demonstrate the feasibility and effectiveness of unsupervised schema discovery in highly heterogeneous settings.

The rest of the chapter is organized as follows. Section 3.2 provides a formal definition of the schema discovery problem. Section 3.3 presents our measure of schema quality based on MDL. Section 3.4 presents SQuaScheD, a quality guided algorithm to perform schema discovery. Section 3.5 presents the results of our experimental evaluation. Section 3.6 summarizes the related work and Section 3.7 concludes the chapter and discusses future research directions.

3.2 PRELIMINARIES

In this chapter, we restrict the problem of schema discovery to a simple and generic entity-centric model, though our solution can be applied to a broader scope of data models. We assume that a dataset is composed of a set of entities (representing real world or abstract objects), each being described by a set of attributes. For each attribute, we only consider its type (e.g. person.name) and ignore its value (e.g. *Michael Jackson*). We believe that attribute types contain enough

1 Spectral QUAlity-guided SCHEma-Discovery

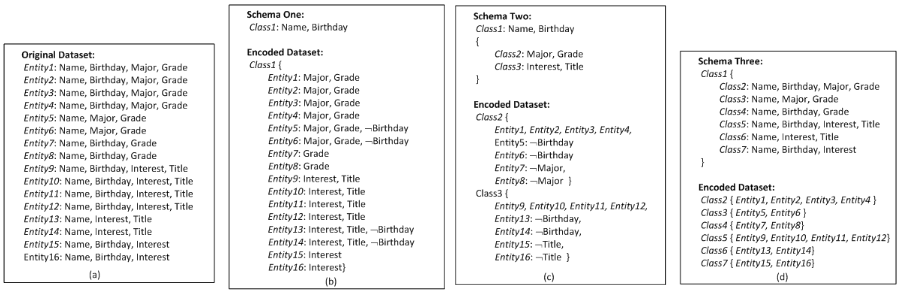


Figure 3.1: Illustrative Example for dataset Encoding and Schema Quality

relevant information for schema discovery in the way they co-occur in entities. While attribute values contain additional information for further enhancing the results of schema discovery, they represent an additional source of noise and requires further and more complex processing, which we intend to tackle in future work. For simplicity, we use the term *attribute* when we refer to attribute type in the rest of the chapter. Based on this notion of attribute, our definitions of dataset and schema are as follows.

Definition 10 *Dataset. A* dataset *is a triple* $D \equiv (A, E, da)$*, where* A *is a set of attributes,* E *is a set of entities, and* da *is an* entity-attribute assignment function $da : E \to \mathcal{P}(A)$ *from* E *to the power set of* A*, which assigns a subset of the attributes to each entity.*

An example of a dataset is given in Figure 3.1(a). It contains 16 entities described by overall 6 attributes. The first eight entities are eight students, whose typical attributes include Major and Grade. The last eight entities are eight researchers, whose typical attributes include $\mathsf{Interest}$ and Title.

Definition 11 *Schema. Given a set of attributes* A*, a* schema $S(A) \equiv (R, C, p, aa)$ *is a class hierarchy composed of a* root class R*, a set of* classes C *where* $R \notin C$*, a* subclass relationship $p : C \to C \cup \{R\}$ *mapping each class to its superclass, and a* class-attribute assignment function $aa : C \cup \{R\} \to \mathcal{P}(A)$ *assigning a set of attributes to each class. In addition,* aa *is subject to the inheritance constraint implied by the subclass relationship. That is to say, for each pair of classes* $c_1, c_2 \in C \cup \{R\}$ *such that* $p(c_1) = c_2$*, if* $a \in aa(c_2)$ *then* $a \in aa(c_1)$.

Basically, a schema defines the classes (i.e. entity types), their subsumption relationships, and the particular attributes for each class. *Schema Two* in Figure 3.1(c) is a possible schema for the data set in Figure 3.1(a), where $\mathsf{Class1}$ is the root and $\mathsf{Class2}$ and $\mathsf{Class3}$ are its subclasses. Each class is associated with two explicit attributes. According to the inheritance constraint, we can infer that Name and $\mathsf{Birthday}$ are also attributes of $\mathsf{Class2}$ and $\mathsf{Class3}$. As an interpretation, we can regard $\mathsf{Class1}$ as Person, $\mathsf{Class2}$ as $\mathsf{Student}$ and $\mathsf{Class3}$

as Researcher. *Schema One* and *Schema Three* in Figures 3.1(b) and 3.1(d) are further example schemas for the same data set. *Schema One* is special, as it contains only a single class, which is the root.

When there is a schema defined, the entities in a dataset can be described referring to the schema, instead of listing their attributes exhaustively. There are a variety of ways to encode a dataset based on a schema. We choose one encoding method for the purpose of illustration.

Definition 12 *Dataset Encoding. Given a dataset* $D \equiv (A, E, da)$ *and a schema* $S(A) \equiv (R, C, p, aa)$*, an encoding of* D *based on* $S(A)$ *is represented by* $En \equiv (ea, sda)$. ea *is an* entity-class assignment function $ea : E \rightarrow \mathcal{P}(C \cup \{R\})$ *assigning a set of classes to each entity.* sda *is a* schema based entity-attribute assignment function $sda : E \rightarrow \mathcal{P}(A \cup \neg A)$. $\neg A$ *represents all the negated attributes, each indicating the absence of an attribute.* sda *assigns to each entity a set of attributes and negated attributes based on the following principle: (1) given* $e \in E$*, if* $a \in da(e)$ *and* $a \notin aa(ea(e))$*, then* $a \in sda(e)$*; (2) given* $e \in E$*, if* $a \notin da(e)$ *and* $a \in aa(ea(e))$ *, then* $\neg a \in sda(e)$.

Figures 3.1(b), 3.1(c) and 3.1(d) show the encodings of the dataset in Figure 3.1(a) based on the *Schemas One*, *Two* and *Three*, respectively. In Figure 3.1(b), all the entities are assigned to the root class. As it is known that Name and Birthday are attributes of the root class, we do not need to encode them again when describing the individual entities in the class. Only when attributes are absent in some entities of a class, we assign their negation to those entities. In Figure 3.1(c), 8 entities are assigned to Class2 and the other 8 are assigned to Class3. As the attributes of Class2 and Class3 can cover the attributes of the entities quite well, only a small number of negated attributes is required to encode the entities. In contrast, Figure 3.1(d) uses a very fine-grained schema, such that no entity-attribute assignment is needed for encoding the entities.

3.3 QUANTIFYING SCHEMA QUALITY

The goal of schema discovery is to find a good schema for a given dataset. As illustrated by Figure 3.1, even for a very small dataset, there can be a large number of possible schemas. The question is how to determine which schema is the best. Comparing the three schemas in Figure 3.1, Schema Two seems to be the best fit. Schema One is overly general, as it is natural to classify the first 8 entities and the last 8 entities into two different classes. In contrast, Schema Three is a case of overfitting. Some of its classes can actually be merged. For instance, while the attribute Major is missing on Entity7, this does not seem sufficient to justify a separate class differentiating Entity7 from Entity1 to Entity4 – they share the other three attributes and missing attributes are very common in an open-world dataset. In comparison, Schema Two is sufficiently general while capturing the main characteristics of the dataset.

Our basic intuition is that a good schema should be precise enough to capture the regularities in the data while providing a useful amount of generalization to keep the number of classes under control. This intuition is similar to the spirit of the Minimum Description Length (MDL) principle [10, 65], which claims that the best hypothesis should allow for maximum compression and hence imply minimum description length. Inspired by this principle, we propose to measure the quality of a schema with respect to the dataset it describes as *the amount of storage space needed to describe both the schema and the encoded dataset*, i.e., the overall description length.

To understand the quality measure, consider the examples in Figure 3.1 again. For simplicity, we use the number of words to measure the description length, i.e., the storage space needed to describe the schema and the encoded dataset. We omit punctuation characters for the sake of simplicity. As shown in Figure 3.1, Schema One requires 3 words for describing the schema and 49 words for its encoded dataset. Thus, its overall description length is 52. Schema Two requires 9 words for the schema and 26 words for describing the dataset. Its overall description length is 35. Schema Three requires 27 words for the schema and 22 words for its encoded dataset. Its overall description length is 49. As expected, Schema Two exhibits the smallest overall description length and thus is of the best quality.

We propose a simple probabilistic model describing a schema and its related data, which allows us to compute the entropy and finally the minimum length required to describe the schema and the data.

As introduced in Section 3.2, a schema encodes two kinds of information: 1) assignment of attributes to types; and 2) assignment of entity instances to types. To describe an entity knowing its type and the attributes an instances of this type might have, we need only describe the anomalies this specific entity instance shows. This is what we refer to by describing the data with help of the schema.

Given a schema S and a dataset D, we formally define the overall description length $L(S, D)$ as the sum of the description length of the schema $L(S)$ and that of the dataset encoded with this schema $L(D|S)$:

$$L(S, D) = L(S) + L(D|S) \tag{3.1}$$

According to Definition 11, a schema S is composed of four items: R, C, p and aa. Among the four, the class-attribute assignment aa dominates the description length, as we can deduce R and C from aa. The parent function p is considered by dependencies within aa, as will be clarified later. Therefore, the description length of aa can be used to determine $L(S)$. Thus, we can write $L(S)$ as

$$L(S) = L(aa) \tag{3.2}$$

According to Definition 12, a dataset is encoded using an entity-class assignment ea and a schema based entity-attribute assignment sda. Therefore, the description length of the encoded dataset can be calculated as

$$L(D|S) = L(ea) + L(sda) \tag{3.3}$$

Given a schema S and a dataset D, there are multiple possible eas and their corresponding sdas. Nevertheless, we can compute the minimum $L(D|S)$ using information theory, without enumerating all the possible eas and sdas. In what follows, we show how to estimate $L(aa)$, $L(ea)$ and $L(sda)$, respectively.

3.3.1 *Description Length of Schema*

As mentioned earlier, the class-attribute assignment dominates the description length of a schema. Its description length can be calculated as the amount of information required to describe which attribute is assigned to which class. There are multiple ways to model this assignment. We choose to represent each class as a random variable, which takes its value from a binary alphabet $\{true, false\}$. The random variable has a certain realization on each attribute. The value of a random variable is $true$ if the attribute is present in the class and $false$ otherwise. We assume that the probabilistic distribution of a class's random variable depends solely on its parent's random variable. This reflects the parent relation p between a class and its subclasses, as it favors schemas where child-classes are similar to their parent – more precisely, of high mutual information.

With the model, given a schema $S(A) \equiv (R, C, p, aa)$, the description length of aa can be calculated as:

$$L(aa) = |A| \; H(aa) \tag{3.4}$$

where A is the set of attributes in the schema and $H(aa)$ is the joint entropy of the random variables as defined in the model.

We assume that the probabilistic distribution of a class's random variable depends solely on its parent's random variable. This reflects the subclass relation p between a class and its subclasses. Let aa_c represent the random variable of the class c. The dependency model defined above allows us compute the joint entropy as:

$$H(aa) = \sum_{c \in C} H(aa_c \mid aa_{p(c)}) \tag{3.5}$$

where C represents the complete set of classes and $p(c)$ represents the superclass of the class c. As the root class R has no superclass, its entropy is simply $H(aa_R)$. Note that the entropy of a discrete distribution P is $\sum_{p \in P} p \ \log(1/p)$.

3.3.2 *Description Length of Dataset*

3.3.2.1 *Entity-Class Assignment*

We measure the information of an entity-class assignment analogously to that of class-attribute assignment. We also correspond each class to a random variable, while the realization of each random variable on each entity depends on whether the entity belongs to the class corresponding to the variable. Then, given a schema $S(A) \equiv (R, C, p, aa)$ and an encoded dataset $En \equiv (ea, sda)$, the description length of the entity-class assignment can be calculated as:

$$\begin{aligned} L(ea) &= |E| \ H(ea) \\ &= |E| \sum_{c \in C} H(ea_c \,|\, ea_{p(c)}) \end{aligned} \tag{3.6}$$

where E represents the complete set of entities, C the complete set of classes, and ea_c the random variable of the class c.

3.3.2.2 *Schema Based Entity-Attribute Assignment*

There are two ways to model an entity-attribute assignment probabilistically – either treat each entity as a random variable or treat each attribute as a random variable. We opt for the latter, since the number of entities is likely to be greater than the number of attributes, so that the marginal probability estimation can be more accurate. In other words, we model each attribute as a random variable. The random variable has a certain realization on each entity. It takes the value $\mathtt{true}$ for the entities on which it occurs and $\mathtt{false}$ for the entities on which it does not occur.

In analogy to the previous models, given the schema S, the amount of information needed to describe the entity attribute assignment sda can be calculated as:

$$L(sda) = |E| \ H(da \,|\, S) = |E| \ H(da \,|\, S_A) \tag{3.7}$$

where E represents the complete set of entities and $H(da \,|\, S)$ the conditional entropy of the attribute random variables given the schema S. Concretely S is represented by the entity-assignment ea and the attribute assignment aa described above, i.e., random variables $S_A \equiv \{s_a \forall a \in A\}$ taking value $\mathtt{true}$ for entities assigned in a class where a is assigned and $\mathtt{false}$ otherwise. Note that the sda defined in Definition 12 is for the purpose of illustration. It is not necessarily the optimal encoding method that can achieve the minimum description length.

3.3.2.3 *Conditional Distribution of Attribute Variables*

The conditional entropy $H(da\,|\,S)$ can in principle be expressed as the difference between two joint entropies, that is, $H(da\,|\,S_A) = H(da \wedge S_A) - H(S_A)$. However, as there can be a large number of attributes (in the order of thousands or more), the samples for estimating the joint probabilities are usually insufficient, i.e. the number of entities, resulting in a problem of under-sampling. We choose to adapt the approach of Chow & Liu in [29] to approximate the joint distributions, as it is able to deal with the under-sampling problem while still considering dependency between the schema and the dataset. This approach approximates the joint distributions as the products of a subset of the possible second- and first-order distributions, such that a random variable depends at most on one other random variable, thus building a forest of trees where a child variable depends only on its parent. In the case of description length for schema quality, we want to constrain each attribute-entity assignment da_a to depend on the corresponding variable s_a for the same attribute a, since we want to measure how well the schema ($s_a \forall a \in A$) help describe the data. Therefore, the Chow & Liu selection of second-order distributions is limited to the variables in S_A. If we use $\hat{H}(S_A)$ to denote the maximum likelihood estimate of $H(S_A)$ computed using the Chow & Liu method, we can write the joint entropies as:

$$H(da \wedge S_A) \approx \sum_{a \in A} H(da_a\,|\,s_a) + \hat{H}(S_A) \tag{3.8}$$

where A represents the set of attributes, da_a the random variable of the attribute a realized in the entity-attribute assignment da, and s_a the random variable of the attribute a realized in the schema S as described above. Since $\hat{H}(S_A)$ is exactly the quantity that we deduce from the joint entropy to yield the conditional entropy of da given S_A, we can avoid the computation of $\hat{H}(S_A)$ and write the description length of the entity-attribute assignment conditioned on a schema as:

$$L(sda) = L(da\,|\,S_A) \approx |E| \sum_{a \in A} H(da_a\,|\,s_a) \tag{3.9}$$

This completes our schema quality measure. Finally, we can write the description length of a schema with respect to a dataset as:

$$\begin{aligned} L(S, D) = \quad & L(aa) + L(ea) + L(sda) \\ \approx \quad & |A| \sum_{c \in C} H(aa_c\,|\,aa_{p(c)}) \\ & + |E| \sum_{c \in C} H(ea_c\,|\,ea_{p(c)}) \\ & + |E| \sum_{a \in A} H(da_a\,|\,s_a) \end{aligned} \tag{3.10}$$

3.3.3 *Resilience Against Data Heterogeneity*

Previous approaches to schema induction [53, 62] emphasize syntactical compatibility between the schema and the data. These approaches do not apply well to the data on the Web, which is so heterogeneous that syntactical compatibility does not necessarily reflect the correctness of the discovered schema. There are a few types of heterogeneity in Web data. First, Web data makes an open-world assumption, such that missing attributes are very common. Second, synonymy and polysemy can be found among different data entities. The synonymy of attributes is observed when two or more lexically distinct attributes are used to describe one meaning of a property. For example, when *first_name* and *forename* are both used to describe the given name of a person, we say that they are synonymous attributes. Conversely, if a same attribute is used to describe more than one meaning, we say that the attribute is polysemous. For example, *title* is used simultaneously to refer to the title of a book and a person's professional position.

Fortunately, Minimum Description Length is an information theoretical measure rather than a syntactical measure. It is therefore quite resilient against noise and heterogeneity. This section provides a brief explanation about why MDL is resilient against the three types of heterogeneity described above. To this end, we assume that S is the optimal schema for a dataset D, that is, S has achieved the minimum description length. We intentionally introduce additional heterogeneity into D and assess the likelihood that it turns S' into a suboptimal schema.

MISSING ATTRIBUTES. To introduce missing attributes into D, we randomly select a set of entities with a common attribute A and drop A from these entities. As a result, we transformed D into D'. We consider two schemas, S and S', where S is the original best schema with the minimum description length $L(S, D)$, and the description length of S' is worse than that of S, i.e., $L(S, D) < L(S', D)$. We consider the likelihood to turn S' into a better schema on D', that is, the chance of $L(S', D') < L(S, D')$. It turns out that the likelihood depends on both the gap between $L(S, D)$ and $L(S', D)$ and the distribution of the selected entities whose attributes are dropped. When the selection of the entities is completely random and independent from other information of D, as the MLD in Equation (3.10) is an information theoretical measure based on entropy, we can infer that $L(S, D) - L(S, D') \approx L(S', D) - L(S', D')$. That is to say, the missing attributes will incur the same change on the description lengths of both S and S'. This will certainly not turn S' into a better schema than S. To make S' a better schema, the distribution of the selected entities has to be skewed. In other words, if we want to make S a suboptimal schema, we need first identify a schema S' with sufficiently small gap

$L(S', D) - L(S, D)$, and skew the distribution of missing attributes so it happens to favor S'. The likelihood of this case is very small, as missing attributes in real world are rather random. When more missing attributes are introduced, the randomness would increase, which would make the case even less likely.

SYNONYMOUS ATTRIBUTES To introduce synonymous attributes into D, we randomly select a set of entities with a common attribute A, and replace A with its synonym A'. This case is analogous to the case of missing attributes. Therefore, we can reach the same conclusion – when the synonymous attributes are randomly assigned to the dataset, it is unlikely to turn an optimal schema into a suboptimal one.

POLYSEMOUS ATTRIBUTES The case of polysemous attributes is different from the cases of missing attributes and synonymous attributes. To introduce polysemous attributes into D, we select two attributes A and A' and merge them into one attribute A. The merge transformed D into D'. Again, we consider two schemas, S and S', where S is the optimal for D and S' is not. We assess the likelihood that S' becomes a better schema than S on D'. Based on Equation (3.10), if A and A' do not co-occur in the same class in S', the description length of S' will remain unchanged. Therefore, only when A and A' co-occur in the same class C, it is possible to make S' better. Besides, we also need the distribution of A and A' is skewed, so it exhibits a certain correlation with the other attributes in C. This again is an unlikely case.

In summary, while Web data is heterogeneous, the heterogeneity is normally random. Due to such randomness, it has limited influence on the information theoretical measure of MDL.

3.4 QUALITY-GUIDED SCHEMA DISCOVERY

The schema quality measure presented in the previous section allows for unsupervised schema discovery. It can be used to guide the decision making required by supervised methods, such as the decision on the number of subclasses of each entity type or the size of the schema. We present hereafter an unsupervised schema discovery algorithm leveraging the quality measure.

3.4.1 *Problem Definition of Schema Discovery*

We define schema discovery as a score optimization problem, i.e, to find the schema encoding that minimizes the description length $L(S, D)$ as defined in Equation 3.1. For this purpose, we introduce the notion of *Schema Space* as a basis for defining the *Schema Discovery Problem*:

Definition 13 *Schema Space. Let* D *be a dataset over an entity set* E *and an attribute set* A *as defined in Definition 10. The* schema space $\mathcal{S}$ *for* D *is the set of all possible schemas (see Definition 11) over* D.

Definition 14 *Schema Discovery Problem. Given a dataset* D *over an entity set* E *and an attribute set* A*, and its schema space* $\mathcal{S}$*, find the optimal schema* $s^* \in \mathcal{S}$ *such that the description length is minimized, i.e.,* $L(s^*, D) = \min_{s \in \mathcal{S}} L(s, D)$.

Lemma 1 *The schema discovery problem is NP-Hard.*

Proof 1 *The schema discovery problem is reducible from the problem of finding the schema* of depth one *that minimizes the description length. This is equivalent to finding the root's subclasses that minimizes the description length. As per Definitions 11 and 12, a class consists of a set of attributes and entities. Therefore, the latter problem is further reducible from the problem of finding the set of sets of attributes that minimizes the description length. This is the minimum entropy set-cover problem [68], a variant of the set-cover problem, which is proved to be NP-Hard. Hence, the schema discovery problem is reducible from the minimum entropy set-cover problem, and is therefore NP-Hard.*

As schema discovery is an NP-Hard, a practical solution is to design a heuristic algorithm that produces near-optimal results.

3.4.2 *The SQuaScheD Method*

Our proposed SQuaScheD method considers schema discovery as a search problem, where the search space is defined by the schema space reachable from an initial schema through sequential application of some schema transformation operations. Specifically, it regards the initial schema $s_0 \in \mathcal{S}$ as composed of one class C_0 containing all entities and all attributes. The search process assembles that of top-down hierarchical clustering, which expands a single class step by step to form a hierarchy of classes. It mainly applies the following two types of schema transformation:

EXPAND-K Choose a leaf-node class C_i in the current schema, perform clustering on its attributes and entities to group them into K subclasses, and add the subclasses as children of C_i.

The clustering is supposed to separate the attributes and entities that are least correlated, so that the schema quality can be optimized. For this purpose, we use spectral graph clustering method, which will be detailed subsequently. The number of subclasses K is determined by the minimum description length described in 3.3.

PULL-N After an expansion, move N attributes back from the subclasses to the parent; if all the attributes of an entity are moved up, also move the entity to the parent class.

Intuitively, the expansion step partition all the attributes and entities into the subclasses. However, some attributes or entities should better stay in the superclass. For instance, "name" should be rather an attribute of "person" than that of its subclass "student". The pulling step is intended to identify attributes or entities that fits better with the superclass and moves them backwards.

Given a class C_i that has been divided into several subclasses, attributes satisfying the following properties should be moved from the subclasses back to C_i: 1) a significant proportion of the entities in the subclasses of C_i have the attribute; 2) the entities described by the attribute are spread uniformly among the subclasses. To quantify the two properties, we use the entropy of the distribution of an attribute's entities among the subclasses, and multiply it with the number of the attribute's entities. Given an attribute a and a set of subclasses C, this measure can be calculated as:

$$I(a) = -|E(a)| \sum_{c \in C} \log \frac{|E(a)|}{|E(C)|} \tag{3.11}$$

where $E(a)$ are the entities described by a, $E(C)$ are all the entities in the subclasses.

To perform the pulling, we sort the attributes in descending order of $I(a)$, and select top-N attributes to pull. The number of attributes to be pulled can be arbitrary and is determined by the minimum description length in the discovery process.

The schema discovery process is composed of a sequence of expansion phases. In each phase, an *expand*-K step and a *pull*-N step are executed consecutively on a chosen leaf class of the schema. Minimum Description Length, the schema quality measure introduced in Section 3.3 is used to determine K and N: start with $K = 2$, for each K, find the optimal N that yields the minimum MDL. This MDL is called the *pulling-optimum* MDL for a specific K. Repeat the operation by increasing K until the pulling-optimum MDL increases. The lowest pulling-optimum MDL is then called the *expansion-optimum* MDL. Its corresponding expansion and pulling steps are executed if the MDL decreases. Algorithm 1 provides an overview of the SQuaSchED method.

3.4.3 *Spectral Graph Clustering*

Most traditional clustering algorithms require that the items to be clustered share a common set of features, based on which a distance can be computed. The *expand* step of schema discovery differs from traditional clustering in that it needs to cluster two kinds of items simultaneously: attributes and entities. Besides, the clustering should consider the interdependence between attributes and entities – entities sharing more attributes are more likely to belong to the same class; and inversely, attributes shared by more entities are more likely to belong the same class.

One can choose to first cluster entities and assign attributes to entity classes based on the schema quality measure. However, we found that such an approach does not leverage the interdependence between entities and attributes well. Therefore, we propose to model entities and attributes as a bipartite graph, and use spectral graph clustering to group them into subclasses. Basically, given a class c_i, its entities and attributes form a bipartite graph $G(c_i) = (V(c_i), D(c_i))$, where the set of vertexes $V(c_i)$ is composed of the entity set $E(c_i)$ and the attribute set $A(c_i)$. An undirected edge $(a, e) \in D(c_i)$ exists between an entity $e \in E(c_i)$ and an attribute $a \in A(c_i)$, if and only if the attribute a is used to describe the entity e.

Spectral graph clustering consists of two steps: 1) it first computes the eigenvectors graph laplacian corresponding to the smallest K eigenvalues, where K is the number of desired clusters; then 2) it applies a standard clustering algorithm, such as K-Means, on the graph nodes, which are represented using the eigenvectors. An introduction to spectral graph clustering can be found in [3]. Several variants of the graph laplacian exist. In SQuaScheD, we use the generalized formulation of the eigenvalue problem of *RW-Normalized* graph laplacian: $Lv = \lambda Dv$, where v and λ are the eigenvectors and -values, D is the graph's degree matrix, $L = D - A$ is the unnormalized graph laplacian, and A is the graph's adjacency matrix. For the second step we use simple K-Means. Experiments showed that they work well for schema discovery.

Figures 3.2a to 3.5c show the eigenvectors corresponding to the 2^{nd} to 5^{th} smallest eigenvalues of various graph Laplacians for the Tunnel[2] and Event datasets computed with an exact and the Arnoldi methods. We don't show the first eigenvector since the smallest eigenvalue of all graph Laplacians is 0 with unit eigenvector and thus present no interest for graph clustering[3]. In Figure 3.2b for example, each entity and attribute of the Tunnel dataset are represented as a point in each of the four graphs. Its color denotes the class to which it

2 The limited size and number of classes of the Tunnel dataset make it the most appropriate for a comprehendable graphical representation.

3 In practice, the smallest eigenvector may present some variation due to the limited precision of digital computing or approximation errors for approximation methods.

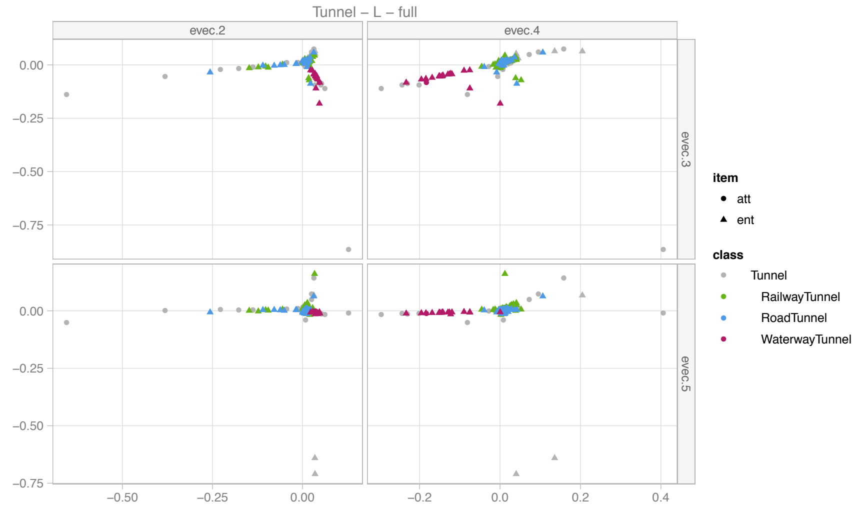
belongs[4]. The graph columns depict the coordinates of the eigenvectors corresponding to the second and fourth smallest eigenvalues and the graph rows those coordinates of the eigenvectors corresponding the the third and fifth smallest eigenvalues. Each figure thus constitutes a good basis to compare the relative elements position according to the smallest eigenvectors.

Figures 3.2a and 3.2b show the eigenvectors of non-normalized and normalized graph Laplacian of the Tunnel dataset respectively, computed with an exact method. Besides containing more flagrant outliers, the non-normalized laplacian depicted in Figure 3.2a yields more compact eigenvector graphs allowing to separate the Waterway Tunnel class but not to seprate Railway from Road Tunnels. The normalized version of the graph Laplacian in Figure 3.2b makes a clear distinction between Railway and Road Tunnels, especially on the fourth eigenvector. Note that since the number of clusters is guided by the schema's minimum description length, it is not necessary for the classes to display clusters with evident inter-cluster distance.

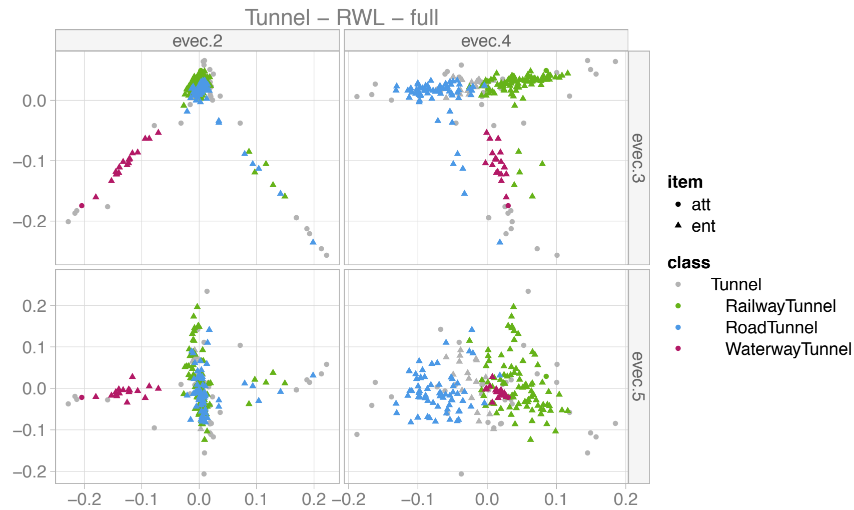
In the light of Figure 3.2b and accounting for similar graphs for the other datasets, spectral graph clustering present good features for schema discovery, ideally allowing to prune the search space for an optimal schema. However, eigenvalue problem solution is computationally expensive: the exact eigenvalue decomposition of the Laplacian for the Event dataset didn't terminate after 5 days.[5] For this reason it is necessary to consider an approximation method: we selected the implicitly restarted Arnoldi method implemented of the ARPACK Fortran package. This method solves the eigenvalue problem for a reduced matrix of the size of the order of the few eigenvectors desired, which represents a considerable gain of time and space.

Figure 3.3 shows the time necessary to solve the eigenvalue problem for the five smallest eigenvalues and their corresponding eigenvectors. Note the logarithmic time scale in minutes on the Y-axis. Even though the five smallest eigenvalues solution is desired, the exact eigenvalue solver (named "full" in the graph) solves the eigenvalue problem for all values, which scales badly as we see: the time compares to the one of required by the approximated solution of the ARPACK package for the smallest Tunnel dataset, but requires already 1-2 orders of magnitude more time than ARPACK on the next smallest sample dataset. The exact eigenvalue solver did not terminate after five days of computation on the Event dataset. On the other hand we see that the ARPACK solver serves its purpose and requires much less time to solve a limited number of eigenvalues, and this in a time depending linearly on the cardinality of the full problem which is shown by the log-shape of the ARPACK curves on the lin-log axes of Figure 3.3. The times to solve the three kinds of

4 The highest class in the hierarchy for attributes and the lowest class for entities.
5 Using the CERN's Colt library implementation of the Householder/QL algorithms.



(a) Standard graph Laplacian.



(b) RW-normalized graph Laplacian.

Figure 3.2: Eigenvectors corresponding to the 2^{nd} to 5^{th} smallest eigenvalues of various Laplacian of the Tunnel dataset's graph.

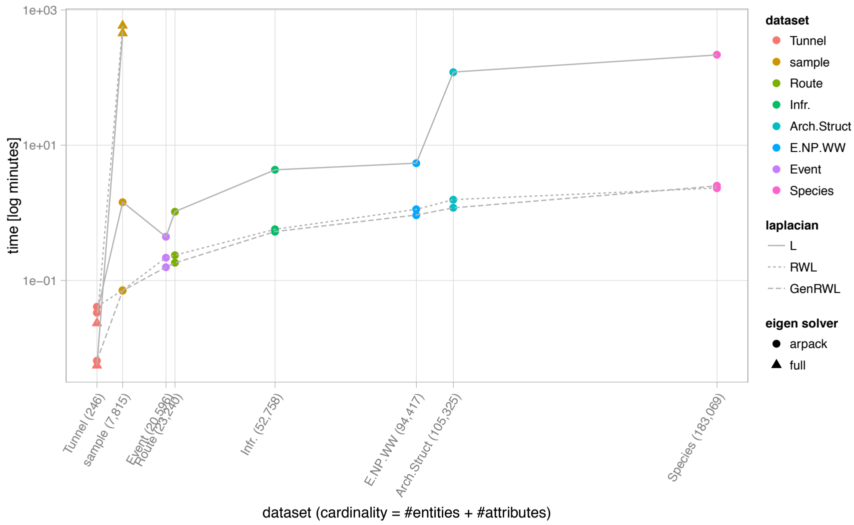
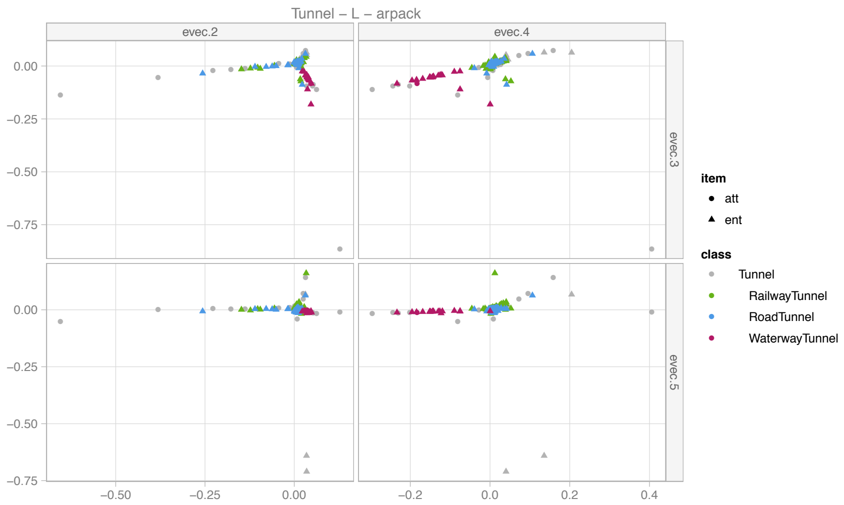


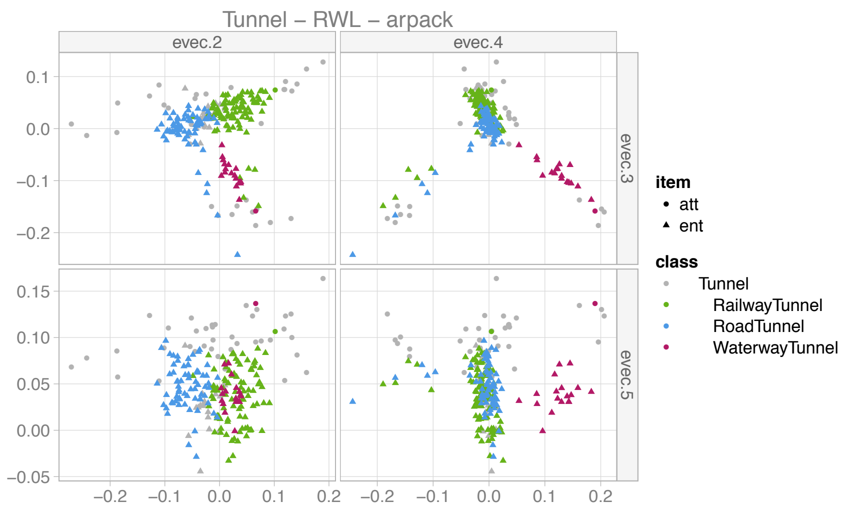
Figure 3.3: Time spent to solve the eigenvector problems.

eigenvalue problems (standard Laplacian, normalized Laplacian, and the generalized eigenvalue problem for the normalized Laplacian) appear to have a similar slope as a function of the cardinality of the full problem, with the standard Laplacian displaying more variance. Furthermore, the normalized Laplacians' eigenvalues are solved much faster than the standard Laplacian ones by 1 order of magnitude. We also clearly see that the generalized eigenvalue problem (GenRWL) requires less time to solve than it's standard formulation (RWL). The implicitly restarted Arnoldi method used by the ARPACK package is an iterative process running until convergence. Thus, requiring less time to converge to a stable solution for GenRWL than for RWL is an sign that GenRWL is more stable and thus delivers a approximated solution closer to the exact solution than RWL, which confirm the observations we made above inspecting the eigenvector plots in Figures 3.2a to 3.5c. We suspect that the difference in time an quality of the solutions of in theory equivalent problems is due to the limited precision of digital arithmetic: the standard problem statement (the normalized Laplacian) is the result of divisions, whereas the generalized problem statement does not.

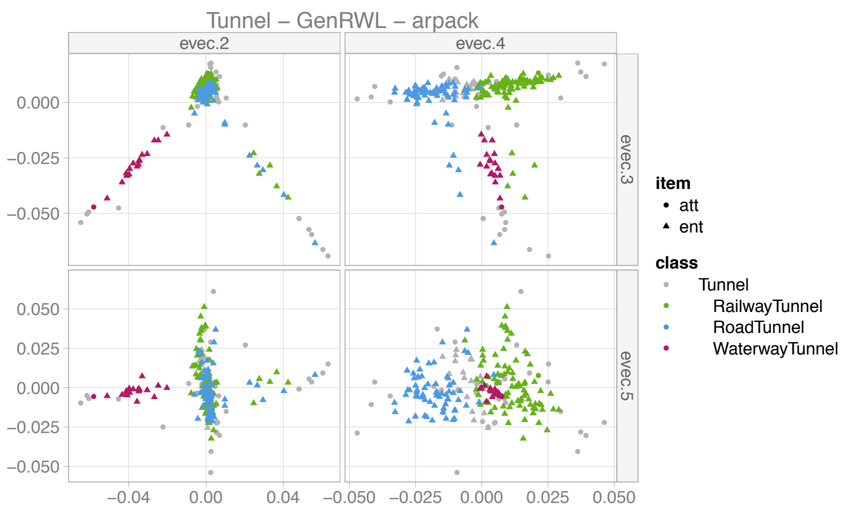
The eigenvectors of the non-normalized Laplacian in Figure 3.4a computed with ARPACK are a very good approximation of the exact solution shown in Figure 3.2a. The eigenvectors of the normalized Laplacian computed with ARPACK in Figure 3.4b however differ significantly from the exact solution in Figure 3.2b. The eigenvectors of the equivalent generalized eigenvalue problem in Figure 3.4c however, is an excellent approximation of the normalized Laplacian's eigenvectors. While the solution of the standard eigenvalue problem still produces clusters similarly separable compared to the generalized problem for the Tunnel dataset, on all other datasets, the eigen-



(a) Standard graph Laplacian.



(b) RW-normalized graph Laplacian.



(c) Generalized eigen-value problem of RW-normalized graph Laplacian.

Figure 3.4: Eigenvectors corresponding to the 2^{nd} to 5^{th} smallest eigenvalues of various Laplacian of the Tunnel dataset's graph solved with ARPACK.

vectors of the generalized problem corresponding to the normalized Laplacian computed with ARPACK present much better separation properties than the standard problem, as can be seen for the Event dataset in Figures 3.5b and 3.5c.

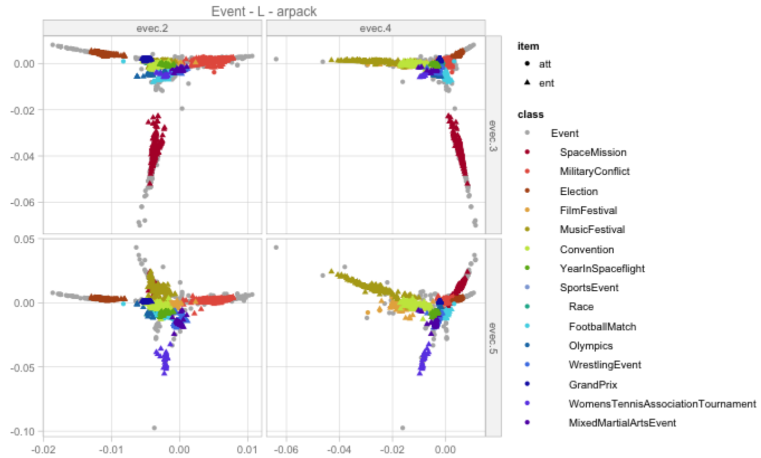
Figure 3.6 gives an objective overview over all dataset of the observations we made by observing the eigenvector plots in Figures 3.2a to 3.5c: it presents the Davies-Bouldin clustering indexes based on the coordinates on the first 5 eigenvectors and the ground truth classes. The Davies-Bouldin index is an internal clustering evaluation measure indicating the ratio between the "tightness" and the "separatedness" of the clusters: the lower the value the better the clustering is. Figure 3.6 confirms the observation we made earlier and generalizes them to other datasets:

- the full eigen solver gives identical results to ARPACK on the non-normalized Laplacian (L);
- the full eigen solver gives better results than ARPACK on the normalized Laplacian (RWL);
- the full eigen solver on the normalized Laplacian (RWL) gives comparable results to ARPACK on the generalized formulation of the same eigenvalue problem (GenRWL);
- eigenvectors of the normalized Laplacian (RWL and GenRWL) are better clustering features than the standard Laplacian (L) on all datasets, except on the Event dataset where they show both the best clustering features compared to other datasets;
- none of the standard (RWL) or generalized (GenRWL) formulation of the eigenvalue problem on the normalized Laplacian presents consistently better clustering features than the other across all datasets.

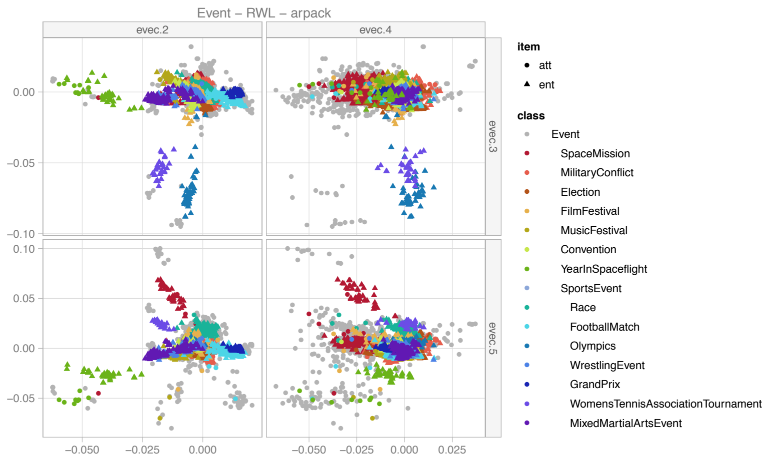
In the light of the above observations and considering the fact that the generalized formulation of the eigenvalue problem of the normalized Laplacian converges fastest, we recommend the latter for schema discovery and this is also the one we used in the following experiments.

3.5 EXPERIMENTAL VALIDATION

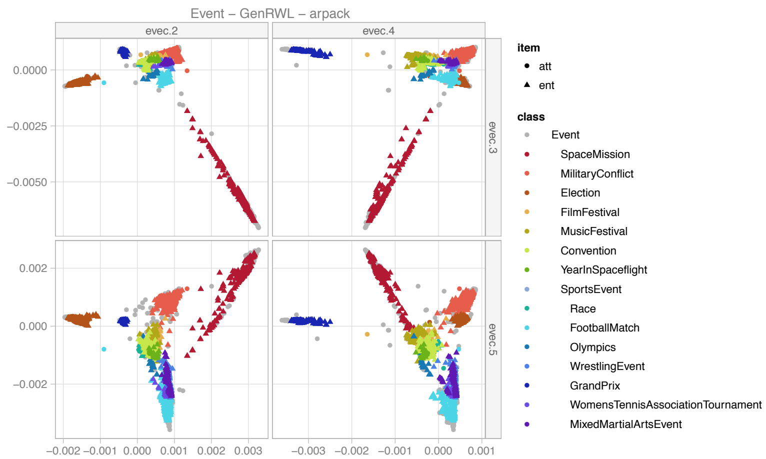
Our evaluation aims to answer the following questions: 1) Does the schema quality measure proposed in Section 3.3 reflect the quality of a schema? 2) Is SQuaSched able to extract a reasonable schema from a highly heterogeneous dataset effectively? Before answering those questions, we introduce the dataset (3.5.1), the evaluation metrics (3.5.2), and implementations specifics (3.5.3).



(a) Standard graph Laplacian.



(b) RW-normalized graph Laplacian.



(c) Generalized eigen-value problem of RW-normalized graph Laplacian.

Figure 3.5: Eigenvectors corresponding to the 2^{nd} to 5^{th} smallest eigenvalues of various graph Laplacians of the Event dataset's graph solved with ARPACK.

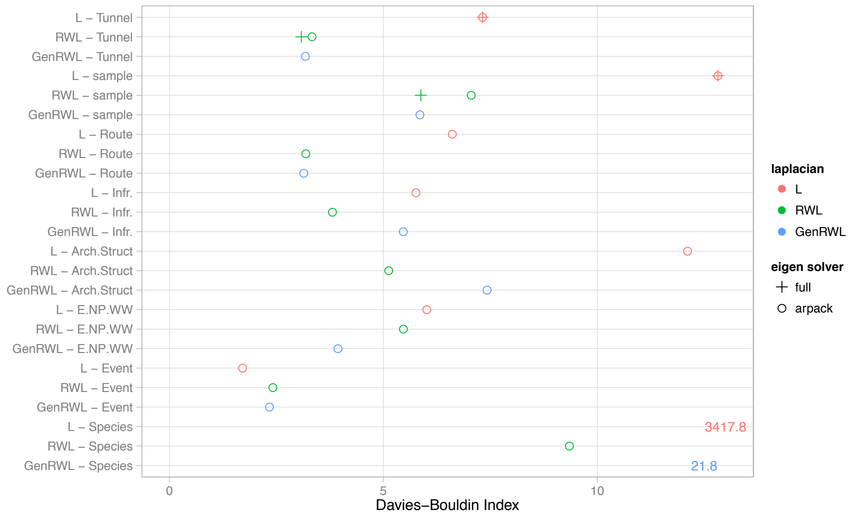


Figure 3.6: Davies-Bouldin clustering indexes based on the eigenvectors corresponding to the 2^{nd} to 5^{th} smallest eigenvalues for all datasets, laplacians and eigenvalue solvers. The lower the better. The "sample" dataset is a sample of "Act.EduInst".

3.5.1 *Dataset: Wikipedia Infoboxes*

A Wikipedia Infoboxes is a table presenting the key properties about a Wikipedia entity, usually located in the top-right corner of a Wikipedia article. Like the rest of Wikipedia, those info-boxes are contributed by users. DBpedia [7] encapsulates those Infoboxes in RDF format, so that they are easily processable by a machine. This structured information comes in two versions in DBpedia: 1) raw Infoboxes directly extracted from Wikipedia without further processing other than parsing and RDF representation; and 2) a processed dataset whose properties and entities are cleaned and mapped in a community effort to a manually created OWL ontology. As the users of Wikipedia did not comply with a common template when creating the Infoboxes, the data structures of the Infoboxes are highly heterogeneous. In contrast, as the DBpedia OWL ontology was created manually, the entity descriptions based on the ontology are much cleaner. Table 3.1 shows the number of duplicate and ambiguous attributes among those attributes from the raw Infoboxes that are mapped to a single property in the ontology. The larger the numbers, the more heterogeneous the data.

To evaluate our schema discovery method, we applied it to the *raw* Wikipedia Infoboxes in DBpedia version 3.7 to generate schemas, and used the DBpedia OWL ontology and associated mappings as the ground-truth to measure the accuracy of the discovered schemas. In fact, there is no absolute ground-truth, as people can define a variety of good schemas to organize the Infoboxes. In addition, what makes an ontology good also depends on the purpose of the application.

top class	# descendant classes	depth	branching factor	# entities	# attributes in raw data	# attributes in ontology	# ambiguous attributes	# duplicate attributes
Act.EduInst	8	2	2.7	38,810	867	236	46	119
Arch.Struct	25	4	5.0	103,900	1,425	304	45	149
Event	15	2	7.5	19,974	622	124	15	36
Infrastructure	14	3	4.7	51,818	940	221	30	98
Route	9	2	4.5	22,744	496	153	22	58
Species	25	4	5.0	182,702	367	86	8	21
Tunnel	3	1	3.0	196	50	33	1	4
E.NP.WW	34	4	4.2	92,979	1,438	266	39	124

Table 3.1: Datasets for experiments: subtrees from the DBpedia 3.7 ontology rooted in the mentioned class

Nevertheless, we believe that DBpedia at least provides a reference perspective to evaluate our method.

To ensure the generality of our evaluation, we selected a number of sub-trees of the DBpedia ontology as the evaluating schemas, each representing a particular domain. Accordingly, we divided the Infoboxes into several test sets, each corresponding to one evaluating schema. As shown in Table 3.1, the test sets are of different characteristics. Some are simple and specific, such as *Tunnel* and *Activity-EducationalInstitution*, some are generic, such as *Species* and *Route*, and some are deep, such as *Architectural-Structure*. The last test set is a combination of three different test sets, i.e. *Event*, *NaturalPlace* and *WrittenWork*, representing a dataset with a higher diversity.

3.5.2 *Evaluation Metrics*

To evaluate the effectiveness of schema discovery, we compare the discovered schemas against the ground-truth introduced previously. This comparison can be conducted in two directions covering two aspects: 1) how many classes from the ground-truth can be found in the discovered schema, and 2) how many classes from the discovered schema can be found in the ground-truth. These two aspects are in their spirit similar to *recall* and *precision* used by IR evaluation. Therefore, we call the two aspects *class-recall* and *class-precision*, respectively. Note that the subclass relationship can be determined by the entities in the classes. Therefore, if a discovered schema has a class-precision and a class-recall of 1.0, then it also has the exact class hierarchy of the ground-truth.

To measure how well a class in a discovered schema matches a class in the ground-truth, we compare their entity sets. For real-world data, there is rarely a perfect match. Therefore, we identify the matches based on a the F1-Measure between the two sets. For each class in a discovered schema, we regard the class in the ground-truth with the best F1-Measure as its match and average those best F1-Measures using the *harmonic mean* weighted by the number of entities in the discovered classes; this gives the *class-precision*. The *class-recall* is computed similarly, inverting the role of ground-truth and discovered classes. We end up with the following definition of class-recall $\mathtt{classR}$ and class-precision $\mathtt{classP}$:

$$\mathtt{classR} = \frac{\sum_{gc \in GC} |E(gc)|}{\sum_{gc \in GC} \max_{dc \in DC} \frac{|E(gc)|}{\mathtt{setF1}(gc,dc)}} \tag{3.12}$$

$$\mathtt{classP} = \frac{\sum_{dc \in DC} |E(dc)|}{\sum_{dc \in DC} \max_{gc \in GC} \frac{|E(dc)|}{\mathtt{setF1}(gc,dc)}} \tag{3.13}$$

where GC represents the set of ground-truth classes, DC represents the set of discovered classes, $|E(gc)|$ and $|E(dc)|$ represent the number of entities in the classes gc and dc, respectively, and $\mathtt{setF1}(gc, dc)$ represents the standard IR F1-Measure between gc and dc computed as $\frac{\mathtt{precision}(gc,dc) \cdot \mathtt{recall}(gc,dc)}{\mathtt{precision}(gc,dc) + \mathtt{recall}(gc,dc)}$.

To combine precision and recall into a overall evaluation measure, F1 is usually used, which regard precision and recall equally important. In the case of schema discovery, we prefer to put more weight on recall. On the one hand, good precision is easier to achieve by restricting the number of classes to be discovered. On the other hand, it is sometimes preferable that a discovered schema has a deeper class hierarchy than the ground-truth. An increased depth does not affect the usability of a schema, as a user can decide which level of granularity to use and access the schema only until that level. The F2-measure puts more weight on recall and penalizes less a more fine-grained hierarchy than F1, and is the only one passing all validation tests we conducted. We thus use the F2-measure to combine class-precision and class-recall, resulting in an overall evaluation metric we call *classF2*:

$$\mathtt{classF2} = \frac{4 \cdot \mathtt{classP}(gc, dc) \cdot \mathtt{classR}(gc, dc)}{\mathtt{classP}(gc, dc) + 4 \cdot \mathtt{classR}(gc, dc)} \tag{3.14}$$

A schema also contains attributes. To evaluate the assignment of attributes to class, we apply *class-precision*, *class-recall* and $\mathtt{classF2}$ specific to attributes.

3.5.3 *Implementation and Hardware*

We implemented SQuaSched and the experiment framework in Scala 2.9.2, a JVM programming language. The COBWEB algorithm mentioned later in the section was part of the WEKA 3.6 Java library [67].

We ran all our experiments on Amazon's AWS `m2.2xlarge` machines. Eigenvectors for spectral graph clustering were computed using the SciPy Python framework, which relies on the ARPACK Fortran77 package for the Arnoldi process for eigenvector solution and the UMFPACK C package as linear system solver in support to the Arnoldi process.

3.5.4 *Evaluation of Schema Quality Measure*

In this section, we show that Minimum Description Length is an appropriate schema quality measure for guiding schema discovery. More specifically, we show how MDL (Minimum Description Length measure) evolves along the steps of the SQuaSCHED method and how it compares with the evaluation measures of the discovered schema with respect to the ground truth.

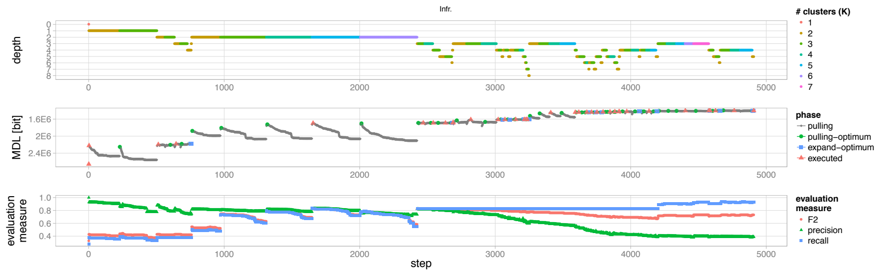


Figure 3.7: Evolution of the MDL, class-precision, -recall, and -F2 along the steps of the SQuaSCHED process on the Infrastructure dataset.

Figure 3.7 depicts the evolution of MDL as well as that of entity based class-precision, class-recall and class-F2 during the schema discovery process for the Infrastructure dataset. The X-axis represents each step of the schema discovery process. The graph on the top shows the depth at which the current process is operating; the color shows the number of children considered for the expansion step. The graph in the middle shows the evolution of the schema quality value: each gray dot is the MDL of the discovered hierarchy returned at line 35 of Algorithm 1; each green dot is the pulling-optimal MDL for a pulling step, returned at line 39; each blue square is the expand-optimal MDL for an expansion step, returned at line 17; red triangles represent the MDLs of the instantiated hierarchies, i.e. updates of `bestMDL` at line 25. The graph at the bottom shows the evolution of F2, precision and recall of the discovered hierarchy with respect to the ground truth.

A good schema is supposed to have a *high* class-F2, -precision and -recall, and a *low* MDL if it is a good quality measure. In other words, MDL should be ideally *anti*-correlated with the evaluation measures. We can see that the evolution trend of MDL is highly anti-correlated with that of recall and somehow positively correlated with that of precision. Please note that the MDL Y-axis is reversed: the lower the MDL, the better the schema quality. This is expected. In the process of the hierarchy expansion, more and more ground-truth classes are discovered, and more false positive classes are supposed to be generated too. Nevertheless, the trend of MDL is clearly anti-correlated with that of class-F2, which indicates that MDL does reflect the schema quality.

To confirm this conclusion, we recorded for each test set the value of the ML, class-F2, -precision and -recall for each step of the SQuaSCHeD method at line 35 of Algorithm 1. We then computed the Spearman's rank correlation coefficient between MDL and the evaluation metrics. As MDL is supposed to be *anti*-correlated with the schema quality, a Spearman's ρ of -1 would be the ideal case.

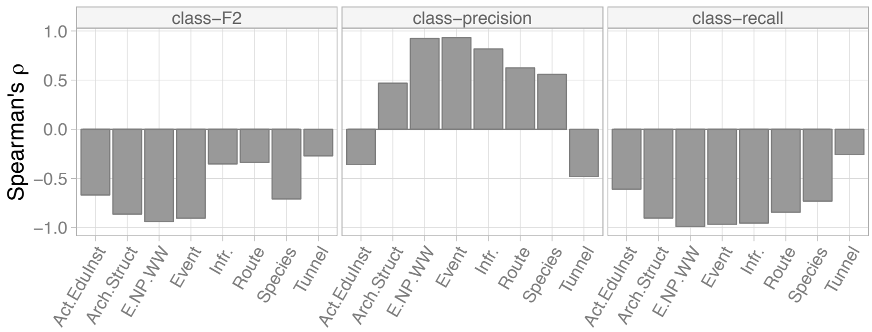


Figure 3.8: Spearman's rank correlation coefficients comparing MDL used by SQuaSCHeD to the F2, precision and recall with respect to the ground truth. A perfect MDL would show a correlation of -1: a good discovered hierarchy has a low MDL and a high F2, precision and recall.

The Spearman's rank correlations between the MDL and the evaluation measures are shown in Figure 3.8. We observe that MDL is strongly anti-correlated with the class-recall, with a ρ value of -0.61 or lower (i.e., better) for all datasets except Tunnel. In contrast, MDL is positively correlated with class-precision, even though this correlation is weaker than the anti-correlation with class-recall. This is natural, as SQuaSCHeD adopts an expansion approach, which would unavoidably include more false positive results as the recall increases. Finally, class-F2 is anti-correlated with MDL with ρ values below 0.67 on 5 out of 7 datasets. This shows that the MDL as a schema quality measure reflects well the class-F2 measure, slightly favoring recall over precision. This is a remarkable property considering that our MDL-based schema quality is completely unsupervised and of course has no knowledge of the ground truth.

3.5.5 *Comparison with COBWEB & Freebase*

In this section, we evaluate how good the SQuaSched method is by comparing it with a state of the art method and Freebase, a human-designed hierarchical schema. Unsupervised hierarchical schema discovery over heterogeneous datasets has not been addressed in previous works, thus no well-established method exists to solve this particular problem. There are 4 characteristics a method needs to fulfill to be a candidate for the comparison:

1. ability to produce a hierarchical classification;
2. ability to determine the number of children of each node, in an unsupervised manner;
3. ability to determine the height and granularity of the hierarchy, in an unsupervised manner;
4. ability to cluster both attributes and entities simultaneously;

COBWEB [47] is a hierarchical clustering algorithm that fulfills the first 2 requirements, and the 3rd one given a predetermined parameter. It uses a utility metric (the category utility) to guide the search of the best hierarchy, and is a conceptual clustering method similar in some aspects to schema discovery. This makes it an interesting candidate for the comparison. Thus, we decided to adapt COBWEB for schema discovery and use it for evaluation.

COBWEB takes as input a set of items described by a fixed set of features and produces a hierarchical classification of the items. Each item is assumed to take a categorical value for *all* the features. This is not the case for schema discovery in heterogeneous environments, as two entities can be described with two completely different sets of attributes. In our schema quality measure, we do not count the values of attributes but only whether an attribute is used to describe an entity. We take the same approach for COBWEB, that is, an entity takes a binary value for each attribute in the dataset based on whether the attribute is used to describe the entity. After clustering the items, we assign each attribute to the cluster whose entities it is most frequently used to describe. After clustering the entities, we partition the attributes among the clusters based on the following principle: an attribute is assigned to the cluster it appears most frequently in the entities.

An additional aspect of COBWEB is that it is an incremental method: the hierarchy is built incrementally by adding the items one at a time. It appears that the order for adding items has non-negligible influence on the results, as we will see in the result of the experiments. Therefore, we considered 3 orderings for adding the entities to the COBWEB hierarchy: in order of *increasing* or *decreasing* number of attributes, or *randomly*.

Finally, the height of a COBWEB hierarchy is controlled via a parameter called "cutoff" – a node with a category utility below the cutoff will not be expanded. Because of this parameter, COBWEB cannot be regarded as a completely unsupervised approach. To facilitate the comparison, we set the cutoff to the value that yields a hierarchy with more nodes than the hierarchy discovered by the SQuaScheD method, and iteratively remove the children of the node with the lowest category utility, until the number of nodes are equal to that of the hierarchy discovered by SQuaScheD. This makes the COBWEB results fairly comparable to the SQuaScheD results in terms of both precision and recall.

Even though the choice of COBWEB for hierarchical schema discovery is reasonable, it was not designed for the task, and it is difficult to demonstrate how SQuaScheD compares with manually created schemas. To allow this, we extracted entity hierarchies from Freebase[6] corresponding to each dataset presented in Section 3.5.1 by means of the links to Wikipedia English pages available in Freebase. For each entity we selected the Freebase class defined by the `notable_types` property, thus representing a human-supervised entity hierarchy that we can use to compare our results. Those Freebase hierarchies are reported below along the other 4 algorithms.

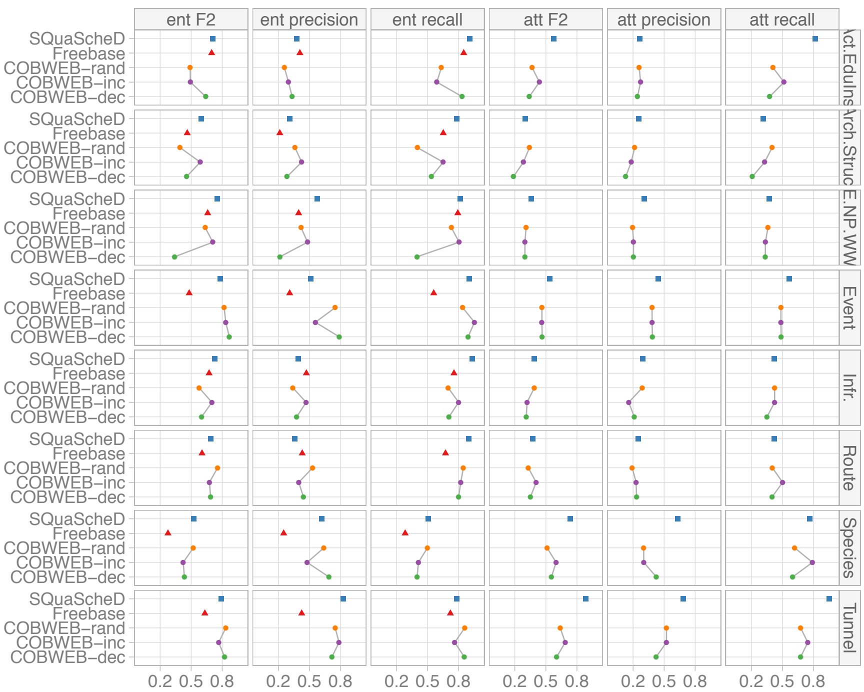


Figure 3.9: Details F2-Precision-Recall graphs comparing schema discovery algorithms among all datasets.

6 http://www.freebase.com

evaluated elements	algo	F2	P	R
entities	COBWEB-dec	6.9 %	40.9 %	* 1.9 %
entities	COBWEB-inc	6.8 %	38.9 %	* 2.8 %
entities	COBWEB-rand	7.2 %	57.4 %	* 2.6 %
entities	Freebase	* 0.3 %	* 4.3 %	* 0.2 %
attributes	COBWEB-dec	* 0.3 %	* 0.3 %	* 1.2 %
attributes	COBWEB-inc	* 0.7 %	* 0.4 %	13.7 %
attributes	COBWEB-rand	* 2.2 %	* 0.9 %	8.6 %

Table 3.2: Paired one-tailed T-Test p-values with unequal variances with the null hypothesis "SQuaSched's mean is lower than *algo*'s (COBWEB's or Freebase's)".

Figure 3.9 shows the class-precision, class-recall, and class-F2 for the entities and attributes over 8 datasets for the 4 schema discovery methods plus the Freebase hierarchies. The results of SQuaSched and Freebase hierarchies are plotted in a distinct color, to distinguish it from other methods, they do not have a specific order. Table 3.2 presents the corresponding p-values of the paired one-tailed T-Test (unequal variance) comparing F2, precision and recall of the results of SQuaSched with that of the COBWEB variants. P-values below 5% are marked with a star.

As we can see from the T-Test's p-values in Table 3.2, SQuaSched's entity recall is significantly higher than any of the COBWEB variants' entity recall. Results with a similar tendency can be observed for the attribute recall, even though SQuaSched does not achieve a significantly higher attribute recall than COBWEB-increasing and COBWEB-random. How the COBWEB variants compare with SQuaSched with respect to entity precision is however less clear: the precision plots in Figure 3.9 show that SQuaSched achieves a better precision than the COBWEB methods on some datasets, and similar or worse on others. This is confirmed by the p-values of the entity precisions in Table 3.2, which are close to 50%. Attribute precision is however clearly at the advantage of SQuaSched with p-values below 1%. SQuaSched appears to achieve a better entity F2 than the COBWEB variants, even though not significantly. SQuaSched's attribute F2 is however clearly better than any of the COBWEB variants with very low p-values.

SQuaSched has also a significantly better F2, precision and recall than the human-supervised Freebase hierarchies, which is also observable in Figure 3.9. We can see that there can be big variation among human generated schemas, because there are different valid ways to classify data. Therefore, the results of SQuaSched and COB-

WEB appear reasonably good. To get an insight about the quality of the discovered schema, please visit the visualization on our Web site.

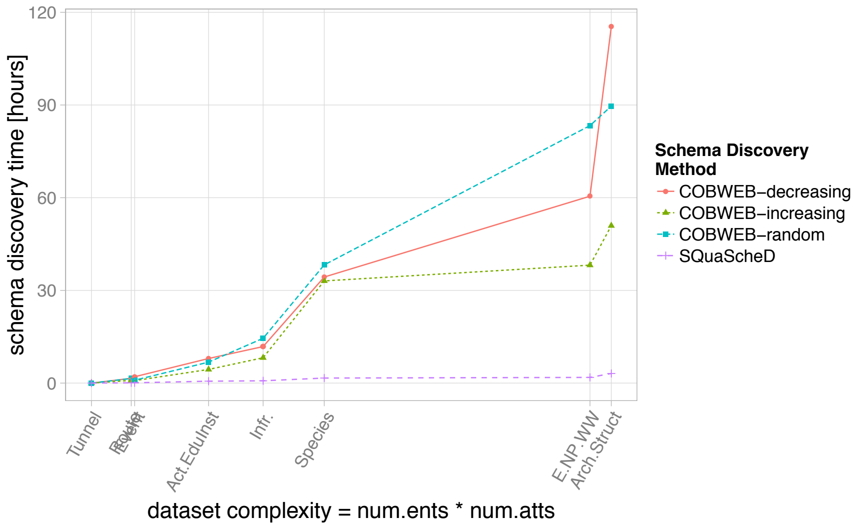


Figure 3.10: Schema discovery times.

Figure 3.10 shows the execution times of the COBWEB variants on the y-axis with respect to the datasets complexities (#entities × #attributes) on the x-axis. For the purpose of timing, all COBWEB variants were run with a same cutoff of 0.5. We clearly see that COBWEB takes much more time than SQuaScheD: where SQuaScheD is able to stop adequately when a reasonable granularity in the discovered hierarchy is reached, COBWEB might discover a much bigger hierarchy. As mentioned earlier the influence of the cutoff on the discovered hierarchy depends on the number of attributes and entities, which makes it difficult to set a priori.

To summarize, SQuaScheD has three advantages over the adapted COBWEB method. First, SQuaScheD is a completely unsupervised approach. Second, it can obtain better schemas than the COBWEB method, as demonstrated in the experimental evaluation. Third, SQuaScheD is much faster than COBWEB – in our experiments, SQuaScheD discovered all hierarchies within 3 hours, COBWEB requires up to 115 hours for the same datasets.

3.5.6 *The Discovered Schemas*

According to the results in the previous section, whereas SQuaScheD can achieve a good class-recall of 0.8, its class-precision is relatively low, which can make wonder whether the automated method can yield useful schemas. In Figure 3.11, we can see that the depths of the schemas discovered by SQuaScheD are similar to the depths of the ground truth, while the number of discovered classes is higher than that of the ground truth. This explains in part the low preci-

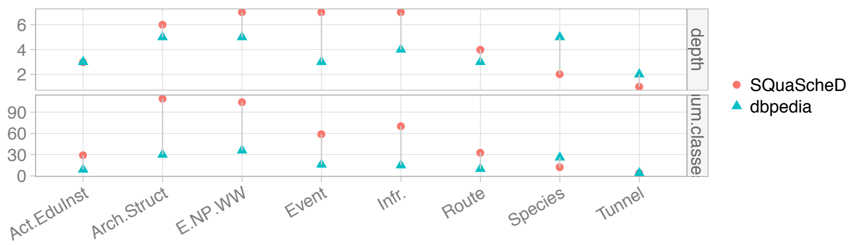


Figure 3.11: Comparison of hierarchy depth and number of classes between the ground truth and the schemas discovered by SQuaScheD

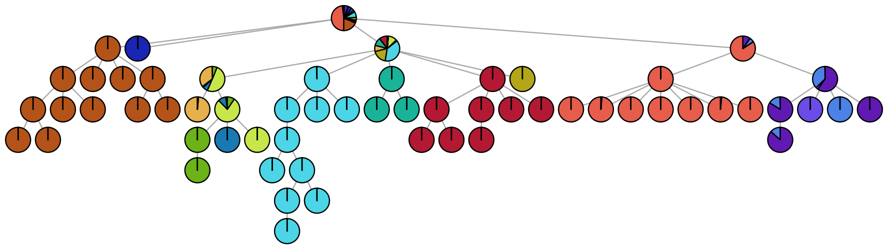


Figure 3.12: Distribution of the bottom-most ground-truth classes in the discovered class hierarchy for the Event dataset.

sion – SQuaScheD tends to discover a more fine-grained schema. Figure 3.12 shows a class hierarchy by SQuaScheD for the Event test set, where the colors show how the different bottom-level classes in the ground truth distribute in the discovered classes – colors similar in hue represent classes closer to each other in the hierarchy. We can see that discovered hierarchy is reasonably good, as the discovered classes are rather homogeneous. This can be explained by the good class-recalls achieved by SQuaScheD, as presented in the previous section.

3.6 RELATED WORK

With the advance of data publication practices, the volume and diversity of the available digital information keep growing. To ease the usability of large and evolving data, researchers have made extensive efforts, among which data summarization and schema induction are two important sets of techniques. They both aim at creating an abstract aggregating layer on top of a dataset, so that data can be browsed, digested and queried more efficiently by humans and machines. In this sense, they are within the same scope as this chapter. However, as the existing approaches were proposed for particular data models or application scenarios, they do not constitute a general solution. Neither do they propose an explicit measure for schema quality.

Data summarization aims to automatically generate a summary of a dataset. Such a summary usually serves two purposes. First, it provides an overview of the information contained in the dataset. Second, it offers a browsing structure for information seeking. The prevailing approach to data summarization is clustering. A number of clustering algorithms (e.g. [17, 88]) have been developed for extracting categorical structures from a heterogeneous dataset. Several methods have also been proposed for identifying keywords for representing each data cluster [90, 84]. Even for conventional structured databases, summarization has been suggested to improve their usability [113]. Nevertheless, as a data summary is mainly used in browsing, it usually does not provide a basis for formulating structured queries. Moreover, data summarization usually applies supervised approaches. Without a concrete quality measure, most of the existing approaches depend on either human intervention or supervised learning to determine the granularity of the summary.

Schema induction, on the other hand, aims to induce a schema from a dataset. This schema can be used as meta-data for efficient query processing. Early schema induction techniques were mainly targeted on semi-structured data, such as XML or OEM data. Being self-describing, such semi-structured data can and often does omit a predefined schema. However, for speedup of data processing, an a posteriori created schema can be very helpful. For this purpose, a number of schema induction techniques such as [62, 89, 63, 6, 54, 107, 108] were introduced for semi-structured data. XTRACT [53] uses minimum description length to quantify the quality of such automatas. In this sense, its shares the spirit of SQuaScheD, as both utilizes the principle of MDL to reach a trade-off between the generality and descriptive power of a schema. However, the problem of schema induction is fundamentally different from the problem addressed in this paper. In their settings, "schema" has different meanings. For schema induction, a schema is a definition of syntactical patterns. For SQuaScheD, a schema is an hierarchical ontology over a large number of heterogeneous data entities. Although both approaches may resort to the measure of MDL, it applies MDL in different ways and for different purposes.

In [21], the author considered schema discovery for Web data. Similar to our work, they aim to identify the entity types of a set of data instances. Also, the authors proposed a measure related to the quality of a schema and modeled schema discovery as a score optimization problem. In contrast to our approach, they measure schema quality by fitness, which assumes that the fewer the missing attributes, the better the schema. However, missing attributes are very common in Web data, where an open-world assumption is made. Thus, the fitness measure does not apply well to heterogeneous datasets. In addition, as their approach does not consider the subclass relationship among entity types, it is unable to generate a schema with a categorical hierarchy, which limits its applicability to large scale datasets.

The field of ontology learning, despite its name similar to schema discovery, addresses a different and broader task: given a collection of textual documents, build a taxonomy of concepts, possibly enriched with non-taxonomic relationships and axioms. [110] describes the first steps of the ontology learning's process:

1. extract terms from textual documents;
2. form concepts by aggregating synonym terms; and
3. discover taxonomic relations (hierarchy) among concepts.

The latter, taxonomic relation discovery is the task approaching most schema discovery because they both aim at discovering a hierarchy. The items organize in a hierarchy are however of very different nature: concepts in ontology learning are sets of synonym terms, whereas classes in schema discovery are sets of entities and attributes and each entity is itself a set of attributes. The attributes are not synonyms and they describe the entity by their combination rather by their intrinsic meaning. This important difference implies that features used to discover taxonomic relations in ontology learning are not directly applicable to schema discovery. For example, external resources like WordNet are not directly usable for schema discovery as they are for ontology learning. Also entities we consider in schema discovery do not have textual documents associated to them such that co-occurrences analysis nor natural language processing techniques as they are used in ontology learning can be used to support schema discovery.

3.7 CONCLUSION AND OUTLOOK

We presented SQuaSCheD, an unsupervised schema discovery method for highly heterogeneous datasets, based on spectral graph clustering and guided by a novel objective measure of schema quality. The quality measure, inspired by the Minimum Description Length principle, achieves an appropriate trade-off between generality and the descriptiveness of a model, in our case the schema. We conducted experiments on Wikipedia Infoboxes, a real-world user-contributed heterogeneous dataset, and compared SQuaSCheD against COBWEB, a hierarchical conceptual clustering method we adapted for schema discovery. The results showed: 1) MDL is an appropriate measure for schema quality; 2) SQuaSCheD outperforms COBWEB in terms of both quality and time complexity; 3) The schemas generated by SQuaSCheD were reasonably well structured, compared to manually created schemas and perform better than the Freebase hierarchies evaluated on DBpedia.

In our future work, we plan to apply unsupervised schema discovery to real applications, such as browsing and query formulation. We will also investigate new methods to further improve SQuaScheD. For instance, we would like to investigate how attribute values can be leveraged to improve the quality of schema discovery. We would like to explore how schema discovery can both improve and profit from information integration techniques as they are used in schema matching and duplicate detection.

Algorithm 1 The SQuaSchеD Method

```
function discoverSchema(D, A, E)
                    ▷ a dataset D over set of attributes A and set of entities E
    root ← new class with all attributes A and all entities E
    add root to an new empty hierarchy H
    expand(H, root)
function expand(H, node)
    bestMDL ← mdl(H)
    bestH ← H
    K ← 2
    repeat
        g ← G(node)                                ▷ graph of elements in node
        C ← spectralClustering(g, K)                           ▷ Section 3.4.3
        add children to node corresponding to clusters in C
        H ← pull(H, node)
        currentMDL ← mdl(H)
        if currentMDL < bestMDL then
            bestMDL ← currentMDL
            bestH ← H
            K ← K + 1
    until currentMDL > bestMDL || K > num elements in node
    for all c ∈ children of node do
        H ← expand(bestH, c)
        currentMDL ← mdl(H)
        if currentMDL < bestMDL then
            bestMDL ← currentMDL
            bestH ← H
    return bestH
function pull(H, node)
    bestMDL ← currentMDL
    bestH ← H
    A ← attributes in node ordered by decreasing I(a) as per Equation 3.11
    for all a ∈ A do
        remove a from all children of node and add it to node
        remove each entity e from each child c of node that are such that no non-inherited attribute in c describe e
        currentMDL ← mdl(H)
        if currentMDL < bestMDL then
            bestMDL ← currentMDL
            bestH ← H
    return bestH
```

4 QUERY RELAXATION WITH MALLEABLE SCHEMAS

4.1 INTRODUCTION

Highly heterogeneous information environments like the social and semantic web poses specific challenges that we introduced in Chapter 1 and developed in Chapter 2, namely:

1. Schema Discovery
2. Query Relaxation
3. Propagation of Entity Identity Revisions

In the previous chapter we presented our approach to discover a schema for data from such environments, enabling the user to formulate a concrete query to answer her information needs using specific attributes. However, the high degree of redundancy of the attributes describing object's properties renders it difficult for a user to formulate a query that would mention all attributes used to describe a desired property. In this chapter we address the second challenge and show how to automatically *relax a query* by expanding the properties expressed by attributes in the query to other attributes likely to describe similar properties, and we show how to rank the results of such expanded queries: entities matching attributes more similar to the one specified by the user in the original query are ranked higher than those entities matching less similar attributes.

Vague schemas can be the result of schema discovery on the web as we presented in Chapter 3. Other well-known examples include Personal Information Management (PIM) [71, 66] and Enterprise Information Management (EIM) [19] systems. In these applications, structured and unstructured data are always mixed: unstructured data usually include documents, images, audio/video and all kinds of human consumable information, while structured data include properties associated with unstructured items (such as *title, creation date, origin, file format*) and various relationships among them (such as *author, reference*). Such property and relationship information can be defined by users, generated by specific applications or extracted using Information Extraction techniques. This information is usually very diverse and vague, so that it is difficult to handle with conventional DB systems, which rely on a clear and rigid data schema.

In [40], the authors proposed *malleable schemas* as a modeling tool for the diverse and vague data structures in the real world. The approach recognizes that the structure of a particular domain is usually

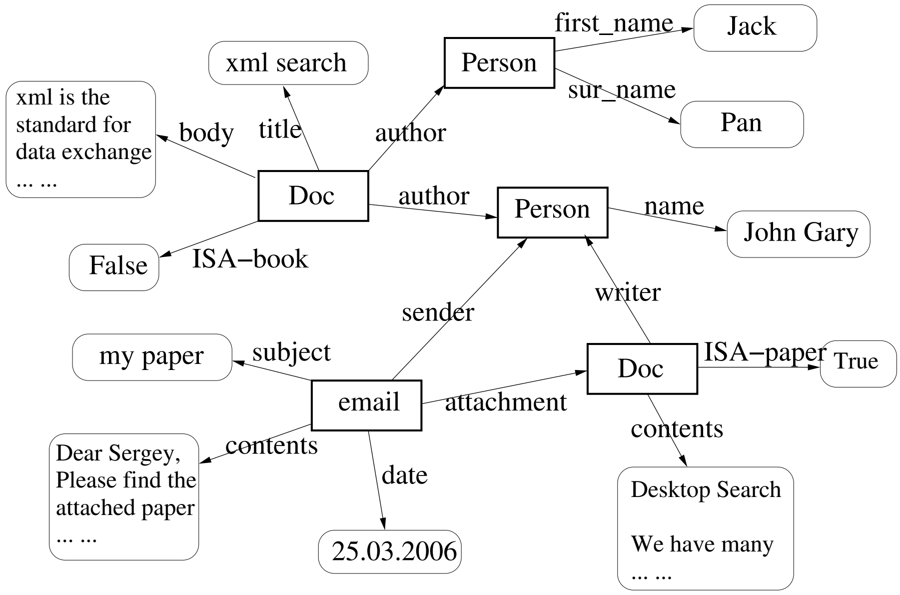


Figure 4.1: Data Modeled by Malleable schema

extremely complicated, and by nature its model should allow some vagueness or redundancies in order to capture all intended semantics. In contrast to the predefined and rigid schemas in conventional database systems, malleable schemas intentionally contain heterogeneous and overlapping definitions of data structures that can be enriched and queried anytime. For example, Figure 4.1 shows a data instance modeled by a malleable schema. There exist various attributes and relationships. We can see that *name*, *first_name* and *sur_name* are overlapping attributes; so are <*title*, *subject*>, <*body*, *contents*> and <*author*, *writer*>. The schema contains a lot of redundancies, so that it is able to effectively capture the diverse semantics in the domain.

Due to these characteristics, malleable schemas are a suitable tool for managing the vaguely structured data in various PIM and EIM systems. In this chapter, we complement the initial work on malleable schemas in [40] by proposing a query scheme that enables users to effectively and efficiently search structured and unstructured information by exploiting malleable schemas for query relaxation

4.1.1 *Challenges*

While a malleable schema is intended to capture heterogeneous data structures (i.e. properties and relationships), a data instance modeled by the malleable schema is usually described by only a part of the structures based on its specific usage. Hence, when a user queries the data using a malleable schema, he has to appropriately relax the query in order to retrieve the complete set of relevant results. For example, a user might issue the following query to search for a person whose first name is "Daniel".

```
Q1: SELECT Person WHERE first_name = 'Daniel'
```

Not all *persons* in the system will be represented with both attributes *first_name* and *sur_name*, though. As shown in Figure 4.1, some *persons* may only use a single attribute *name*. Thus, to find all the persons named "Daniel", the user needs to relax Q1 to Q2.

```
Q2: SELECT Person WHERE first_name = 'Daniel' OR name
    ni 'Daniel'
```

Although the relaxation can find more relevant results, it might also introduce some irrelevant results, because not all persons having *name* $\ni$ *"Daniel"* also satisfy *first_name* = *"Daniel"* ("Daniel" could be a sur_name). A reasonable trade-off is to rank the query results according to their probabilities of relevance, such that the user's need can be satisfied as soon as possible. Considering Q2, the results satisfying *first_name* = *"Daniel"* should be returned prior to the results satisfying *name* $\ni$ *"Daniel"*, as the latter may contain irrelevant results.

In summary, to query data using malleable schemas, we should be able to:

CHALLENGE ONE

1. properly relax the query such that it allows to retrieve all relevant results;
2. rank query results according to their probabilities of matching the original query's intent.

To provide proper query relaxation, we must first identify the correlated elements in a malleable schema. For example, we need to know that *first_name* is a part of *name* in order to know that Q1 should be relaxed to Q2. Furthermore, to enable the ranking of query results, we need to quantify these correlations. If we know that the *title* of an *article* has a stronger correlation to its *abstract* than to its *body*, we can infer that the results of relaxing *title* to *abstract* are more relevant than the results of relaxing *title* to *body*.

Therefore, to address Challenge One effectively, we need to address the following challenge first:

Challenge Two:

1. identify correlated schema elements;
2. quantify these correlations.

This chapter aims to handle these two challenges to enable effective query relaxation using malleable schema.

There is a significant body of work on schema matching [97, 38, 79, 69], focusing on how to discover correspondences between two different schemas. These techniques utilize all kinds of resources, such as schema description, data instances and domain knowledge, to find the schema elements that are similar to each other. Some systems also assign weights or ratings to their outcomes, as quantifications of detected schema matches. However, the semantics of those matches and quantifications are different from the schema correlations needed in query relaxation (see Section 4.3), so that we cannot rely on them to obtain good rankings of query results. In real-world data, there are usually some objects that have been classified and stored in more than one way. In this chapter, we present a novel scheme which utilizes these duplicates within the data sources to discover and quantify the correlations within a malleable schema.

There has been intensive research recently on approximate querying over structured data, like XML documents [51, 25, 5, 95, 76]. Similar to our work, they focus on query relaxation and ranking query results based on various similarities. However, as they only consider the explicit correlations given by the hierarchies of XML trees, they cannot be used to handle queries over malleable schemas. Our results are therefore a useful complement to that field of research.

The contribution of this chapter can be summarized as follows:

1. We propose a query relaxation model that enables users to query vaguely structured information by exploiting malleable schemas (Section 4.2).

2. We propose a scheme that utilizes the duplicates in the database to find and quantify the correlations in a malleable schema (Section 4.3).

3. We devise a query engine that can efficiently perform query relaxation and return ranked results (Section 4.4).

4. We conduct extensive experiments on practical data crawled from Web sources like the IMDB collection and the Amazon catalog to show the effectiveness and performance of our approach in typical scenarios (Section 4.5).

Section 4.6 compares our approach against related work, and Section 4.7 concludes the chapter as well as discusses directions for future research.

4.2 DATA AND QUERY MODELS

We first present the data model that malleable schemas use to define vaguely structured data, and will then introduce a query model and a probabilistic model for query relaxation.

4.2.1 *The Data Model*

Following [40], we assume that malleable schemas use a Entity-Relationship data model, which expresses data by a number of entities, attributes and relationships. An entity is represented by a set of attributes, where each attribute is a binary relation between the entity and a value (usually text). A relationship is a binary relation between two entities. For simplicity, we do not consider relationships involving more than two entities, and we do not assign attributes to relationships. The types of entities, attributes and relationships are identified by keywords, which enables users to easily issue queries. Figure 4.1 shows some example data represented in this model. It includes entities such as *document*, *person* and *email*, attributes such as *name*, *title* and *date*, and relationships such as *author*, *sender* and *attachment*. We consider this data model appropriate and sufficient, because data in real PIM and EIM systems is usually organized around entities such as articles, photos, emails as well.

We will represent categorical attributes of an entity in pivoted form. In other words, we express that an entity belongs (or does not belong) to a category by assigning it an attribute like *ISA-category = true/false*. For example, Figure 4.1 states that the first *document* is not a book and the second *document* is a paper. As shown in Section 4.3, such a representation will enable us to find correlations between individual categories.

A malleable schema is created based on this data model. Different from a rigid schema in traditional relational database, a malleable schema does not need to be concise, but contains imprecise and overlapping definitions of attributes or relationships. In this way, a malleable schema can capture such heterogeneous data structures as in Figure 4.1.

4.2.2 *The Query Model*

Since in PIM and EIM systems user search is usually targeted on entities (like documents, emails and web pages), we use an entity oriented query model for malleable schemas. In this model the objective of a query is always a single entity. A user can express her needs by specifying the attributes of the entity or the relationships between this entity to other entities. For example, the user can issue the following query based on the schema in Figure 4.1,

```
Q3: SELECT Doc AS E1
    WHERE E1.title CONTAINS 'XML Query'
    AND E1.ISA-paper CONTAINS 'true'
    AND E1.author CONTAINS E2
    AND E2.name CONTAINS 'Daniel'
```

which queries for a *paper* whose title contains "XML Query" and whose author is named "Daniel".

To simplify subsequent analysis, in the following we restrict the comparison operator in a query to containment ($\ni$) only, and the connector between selection criteria to conjunctions only (no disjunction or negation). Hence, our queries can be expressed as a conjunction of predicates, each in the form of "$A \ni t$", where A is an attribute or relationship and t is a term or entity. For example, Q3 can be written as:

$$\begin{aligned} Q3 = \{E1 \mid\ & E1.\mathit{title} \ni \text{'XML'} \wedge \\ & E1.\mathit{title} \ni \text{'Query'} \wedge \\ & E1.\mathit{ISA\text{-}paper} \ni \text{'True'} \wedge \\ & E1.\mathit{author} \ni E2 \wedge \\ & E2.\mathbf{name} \ni \text{'Daniel'}\ \} \end{aligned}$$

All entities satisfying these predicates will be correct answers to the query. While the expressiveness of this query model is limited, we believe it can satisfies user needs in most practical cases. Moreover, it can be straightforwardly extended to include more operators. For example, when disjunction is involved, we can first transform the query to disjunctive normal form and process each conjunctive clause separately.

4.2.3 *A Probabilistic Query Relaxation Model*

As each data instance uses only a subset of the attributes or relationships defined in a malleable schema, the predicates in a query have to be properly relaxed to retrieve all relevant results. Such relaxation is achieved by extending the types of attributes or relationships. For example, Q3 can be relaxed by extending *E2.name* $\ni$ *'Daniel'* to *E2.first_name*$\ni$*'Daniel'*, which would still retrieve relevant results. By query relaxation, a query will be turned into a set of queries. We call those the *relaxed queries* of the original query. For example, the relaxed queries of Q3 contain:

$$\begin{aligned} Q5 = \{E1 \mid\ & E1.\mathit{title} \ni \text{'XML'} \wedge \\ & E1.\mathit{title} \ni \text{'Query'} \wedge \\ & E1.\mathit{ISA\text{-}paper} \ni \text{'True'} \wedge \\ & E1.\mathit{author} \ni E2 \wedge \\ & E2.\mathbf{first\_name} \ni \text{'Daniel'}\ \} \end{aligned}$$

and

$$\begin{aligned} Q6 = \{E1 \mid\ & E1.\textbf{subject} \ni \textit{`XML'} \wedge \\ & E1.\textbf{subject} \ni \textit{`Query'} \wedge \\ & E1.\textit{ISA-paper} \ni \textit{`True'} \wedge \\ & E1.\textbf{writer} \ni E2 \wedge \\ & E2.\textit{name} \ni \textit{`Daniel'} \}. \end{aligned}$$

As stated earlier, because query relaxation might introduce irrelevant results, the system should return query results based on their probabilities of relevance. That means, given a query Q_0 that could be relaxed to $Q_1 \vee Q_2 \vee ... \vee Q_n$, we should return their results according to the probabilities $P(Q_0|Q_1), P(Q_0|Q_2), ..., P(Q_0|Q_n)$, where $P(Q_i|Q_j)$ represents the probability that a result of Q_j is also a relevant result of Q_i. As an example, $Q_0 = \{E|A \ni a \wedge B \ni b\}$ is relaxed to $Q_1 \vee Q_2$, where $Q_1 = \{E|A_1 \ni a \wedge B_1 \ni b\}$ and $Q_2 = \{E|A_2 \ni a \wedge B_2 \ni b\}$. If we know that $P(A \ni a \wedge B \ni b|A_1 \ni a \wedge B_1 \ni b) < P(A \ni a \wedge B \ni b|A_2 \ni a \wedge B_2 \ni b)$, we will return the results of Q_2 prior to the results of Q_1, because Q_2 will retrieve more relevant results than Q_1.

Thus, query relaxation requires estimating the correlation between the original query and each of its relaxed queries, i.e. $P(Q|Q_j)$. These conditional probabilities cannot be estimated straightforwardly, as there exist a huge number of query variances even in a simple malleable schema. We therefore use a probabilistic model that relies on two basic assumptions to make the computation of $P(Q|Q_j)$ feasible.

Assumption 1 *Suppose* $Q = \{E|A_1 \ni a_1 \wedge A_2 \ni a_2 \wedge ... \wedge A_k \ni a_k\}$ *is a query, and* $Q' = \{E|A'_1 \ni a_1 \wedge A'_2 \ni a_2 \wedge ... \wedge A'_k \ni a_k\}$ *is a relaxed query of* Q. *For any* $i, j \in [1, k]$ *such that* $i \neq j$, $A_i \ni a_i$ *is independent of* $A_j \ni a_j$, *and* $A'_i \ni a_i$ *is independent of* $A_j \ni a_j$.

Assumption 1 states that all the predicates in a query are independent and their corresponding predicates in the relaxed query are also independent. This assumption is reasonable, because a user seldom uses two correlated predicates in one query. For instance, a user will not say "I am looking for a person whose name is 'Daniel' and whose first_name is also 'Daniel' ", as this is too wordy. Similar assumptions have been widely used in IR models [99, 96]. Some techniques [69] even can identify synonyms by assuming that synonyms rarely cooccur in the same paragraph.

Assumption 2 *For each pair of terms* a *and* b, $P(A \ni a|A' \ni a) = P(A \ni b|A' \ni b)$. *Subsequently, we use* $P(A|A')$ *to denote* $P(A \ni x|A' \ni x)$.

Assumption 2 states that the correlation between two predicates is independent of the values in the predicates. This is equivalent to saying that the correlations in a malleable schema apply to all data instances defined by the schema. Though this might not always be true, we believe it is a sufficiently good estimate of data correlation at the schema level. Functional dependencies in relational databases are an analogous concept.

Theorem 1 *If Assumptions 1 and 2 hold, for any query* $Q = \{E|A_1 \ni a_1 \wedge A_2 \ni a_2 \wedge ... \wedge A_k \ni a_k\}$ *and one of its relaxed queries* $Q' = \{E|A'_1 \ni a_1 \wedge A'_2 \ni a_2 \wedge ... \wedge A'_k \ni a_k\}$, $P(Q|Q') = P(A_1|A'_1)P(A_2|A'_2)... P(A_k|A'_k)$.

Theorem 1 decomposes the correlation between two queries into the correlations (i.e. $P(A|A')$) between the attributes or relationships in a malleable schema. This makes the actual computation of query correlations much easier. In the next section, we introduce our method to discover and estimate the correlations between the attributes or relationships in a malleable schema.

4.3 DISCOVERING CORRELATIONS IN A MALLEABLE SCHEMA

As stated earlier, to achieve query relaxation we need to find and quantify the correlated attributes and relationships in a malleable schema. With the above query relaxation model, the task becomes estimating the conditional probability $P(A|A')$ (i.e. $P(A \ni x|A' \ni x)$) for any pair of attributes or relationships.

A brute force approach to estimate $P(A|A')$ is to examine the data instances where A and A' co-occur. For example, if we find that the terms in the attribute *first_name* also appear in the attribute *name* on all entities, we can estimate that P*(name|first_name)*$= 1$. Similarly, if only half of the terms in the *name* appear in *first_name*, we have P*(first_name|name)*$= 0.5$. However, in real cases, it is very unlikely that correlated attributes always co-occur on a common entity. For instance, if a *person* has been registered using attributes *first_name* and *sur_name*, usually it will not be registered for the attribute *name* anymore, because this is redundant. As a result, there will not be enough samples to make such statistical estimates. We propose to use the duplicates in the data sources instead. These duplicates are entities that are described by different attributes but refer to the same real-world instance, and are very common in multi-application systems such as PIM or EIM. Once being detected, they can serve as good samples for estimating the correlations in a malleable schema.

Therefore, discovering correlations in a malleable schema consists of two steps:

1. detecting duplicates in data;

2. using the duplicates to quantify the correlations in malleable schema.

	title	subject	author	writer	pub-date	rec-date
E1	XML		Daniel		Jan 1999	
E2		XML		Daniel		Dec 2003
E3	DB		Ullman		Jul 1994	
E4		DB		Ullman		Nov 2001
E5	AI		Stuart		Nov 2001	
E6		Logic		Stuart		Nov 2001

Table 4.1: Duplicates under Malleable Schema

Please note that the semantics of schema correlation is slightly different from that of schema match in classic data integration tasks. Schema match measures the likelihood that two attributes refer to the same real-world concept, while schema correlation measures how much the contents of two attributes overlap. As an example, *title* and *abstract* are very different concepts, but they are correlated in contents. Nevertheless, schema match can capture schema correlation to some extent, so that it could be a back-up solution when we do not have sufficient duplicates. We do not address it in this chapter, though it can be an interesting problem for future research.

4.3.1 *Duplicate Detection with Malleable Schema*

The task of duplicate detection has been extensively studied in data integration and data cleaning [41]. However, most techniques assume a rigid schema and cannot be directly applied to malleable schemas. Especially when entities are described by different schema elements, detection results usually become very imprecise. Therefore, duplicate detection with malleable schemas should integrate with the process of discovering schema correlations. On one hand, knowing the correlations, we can more accurately detect the duplicates. On the other hand, the detected duplicates are quality evidences to infer the correlated attributes or relationships in the schema. We will therefore rely on a new algorithm that allows duplicate detection and discovering schema correlations to reinforce each other thus generating more precise results.

4.3.1.1 *An Example*

Table 4.1 shows an example of malleable schema data. It contains 6 entities that are described by 6 attributes. An inspection of the data gives us the impression that *title* vs *subject*, *author* vs *writer* and *pub-date* vs *rec-date* seem to be 3 pairs of correlated attributes, and E1 vs E2, E3 vs E4 and E5 vs E6 seem to be 3 pairs of duplicated entities. If we use the 3 pairs of duplicates to test the 3 attribute correlations, we discover that *pub-date* vs. *rec-date* are actually not correlated (one is publication date and the other is receiving date), because their values are different in most of the duplicates (i.e. E1 vs. E2 and E3 vs. E4). After eliminating this attribute correlation, we will further discover that E5 and E6 are no longer likely duplicates, because they are similar in only one pair of correlated attributes (*author* vs *writer*), which is not sufficiently convincing. In the end, we can conclude that only *title* vs *subject* and *author* vs *writer* are truly correlated, and E1 vs E2 and E3 vs E4 are authentic duplicates.

The above process can be summarized into the following algorithm:

1. **Duplicate detection:** based on current schema correlations, find all possible duplicates.

2. **Correlation detection:** based on current duplicates, reassess the schema correlations.

3. If the schema correlations did not change in step 2, stop the process. Otherwise, go to step 1.

Our algorithm performs duplicate detection and correlation detection iteratively until the resulting schema correlations and duplicates do not change anymore. Finally, the detected duplicates are much more precise than those detected without considering schema correlations. As a side effect, schema correlations will be disclosed, too. The following sections present our detailed algorithms to detect duplicates and schema correlations.

4.3.1.2 *The DSCD Algorithm*

Our algorithm for duplicate and schema correlation discovery (DSCD) is based on the observations above. When presenting the algorithm, we first ignore the relationships between entities and consider only attribute correlations.

Let $c_1, c_2, ..., c_n$ be the pairs of attributes that are likely to be correlated, where each c_i consists of two attributes represented by $c_i = (A_i, A_i')$. Let $d_1, d_2, ..., d_m$ be the possible duplicate pairs, where each d_j consists of two entities represented by $d_j = (E_j, E_j')$. C is a n-dimensional vector, where each element $C(i)$ is a weight indicating our belief of that c_i is a true correlation. D is a m-dimensional vector where each $D(i)$ indicates our belief of that d_i is a true duplicate. S is a $n \times m$ matrix called *evidence matrix*. Each element $S(i,j)$ measures the similarity between the attribute A_i on entity E_j and the attribute A_i' on entity E_j', namely $S(i,j) = Sim(E_i.A_j, E_i'.A_j')$. Obviously, if the attributes in c_j are truly correlated and the entities in d_i are true duplicates, this similarity should be high, and vice versa.

Given the belief of the schema correlations and the evidence matrix S we can deduce our belief in the duplicates. In particular, if the correlated attributes on two entities are more similar, we are more confident that the two entities are duplicates. This can be represented by

$$D(i) = \sum_{k=1}^{n} C(k) * S(i, k)$$

which can be written into matrix form

$$D = C \times S$$

Analogously, we can deduce C from D and S, too. In particular, if two attributes are more similar on the duplicates, we are more confident that the two duplicates are really correlated. This can be represented by

$$C(i) = \sum_{k=1}^{m} D(k) * S(k, i)$$

which could be written into matrix form

$$C = D \times S^T$$

Our DSCD algorithm performs the two deduction processes iteratively until our belief of schema correlations and our belief of duplicates are consistent with each other. As a result, each schema correlation and duplicate is given a weight indicating its likelihood to be true. The detailed algorithm is:

1. $C_0 = (0.5, 0.5, ..., 0.5)$ and $i = 1$;
2. $D_i = C_{i-1} \times S$;
3. $C_i = D_i \times S^T$;
4. If $C_i \neq C_{i-1}$, $i = i + 1$ and go to step 2; otherwise, end the process and output C_i and D_i.

Since we have no proof about valid schema correlations in the beginning, we assign them equal belief, which can be any real value greater than 0 (we use 0.5 here). With the algorithm going on, our beliefs of the schema correlations and duplicates will be repeatedly verified by the evidence matrix S and thus become increasingly clear. Combining he equations in Steps 2 and 3, we have

$$C_i = C_0 \times \left(S \times S^T\right)^i$$

$$D_i = C_0 \times S \times \left(S^T \times S\right)^{i-1}$$

Interestingly, the resulting equations are exactly the same as those of the Kleinberg HITS algorithm [72], which is used to compute page

(E1,E2)	1	0.7	0
(E3,E4)	1	0.7	0
(E5,E6)	0	0.7	1

Table 4.2: Matrix of Table 4.1

ranks in Web searches. Kleinberg shows that according to standard results of linear algebra, C_i is going to converge to the principal eigenvector of $S \times S^T$, and D_i is going to converge to the principal eigenvector of $S^T \times S$. This ensures that also the above algorithm will terminate.

Applying this algorithm to the example in Table 4.1, we have $c_1 =$ (*title,subject*), $c_2 =$ (*author,writer*), $c_3 =$ (*pub-date, rec-date*), and $d_1 = (E1, E2)$, $d_2 = (E3, E4)$, $d_3 = (E5, E6)$. The evidence matrix S can be instantiated to the one in Table 4.2. (E_i.*author* = E_j*writer* will have smaller similarity values because they are more probable to coincide than E_i.*title* = E_i.*subject* and E_i.*pub-date* = E_i.*rec-date*. This is explained in Section 4.3.1.3.) The outcomes (after normalization) will be $C = (1.0, 0.89, 0.28)$ and $D = (1.0, 1.0, 0.56)$. As expected, c_1 and c_2 are more likely to be true than c_3, and d_1 and d_2 are more likely to be true than d_3.

4.3.1.3 *The Similarity Measure*

To instantiate an evidence matrix S, it is important to know how to measure the similarity between two attributes on two entities, namely $Sim(E_j.A_i, E'_j.A'_i)$. In the IR area, there exist a number of methods to measure the similarity between two bag of terms, such as the cosine similarity between TF×IDF vectors. However, we do not think those measures are adequate for application in our case, as they cannot capture the relationship between the compared attributes. For example, they cannot capture that *first_name* is actually a part of *name*. In order to preserve the relationship information, we consider the order of schema correlations and thus distinguish between (A_i, A'_i) and (A'_i, A_i). We also consider the order of similarities and distinguish between $Sim(E_j.A_i, E'_j.A'_i)$ and $Sim(E'_j.A'_i, E_j.A_i)$. Hence, we use the following similarity measure:

$$Sim(E_1.A_1, E_2.A_2) = \frac{|E_1.A_1 \cap E_2.A_2|}{|E_2.A_2|} \times H(A_1, A_2)$$

The first factor measures the percentage of terms in $E_2.A_2$ that are also in $E_1.A_1$. This factor alone does not constitute a good measure, as it is significantly depending on the alphabet sizes of attributes A_1 and A_2. For example, if A_1 and A_2 can only contain two particular terms (e.g. *true* and *false*), the measure will be very likely to be high ($\geqslant 50\%$ if A_1 and A_2 are independent). Therefore, we normalize it by the second factor, the joint entropy of A_1 and A_2, which measures the amount of information of knowing the values of A_1 and A_2. This entropy can be statistically estimated using sample values of the A_1 and A_2 attributes.

4.3.1.4 *Extension to Relationship Correlations*

So far, our approach only deals with attribute correlations. To involve relationship correlations, we only need to know how to measure the similarity between two separate relationships on two entities, namely $Sim(E_j.R_i, E'_j.R'_i)$. As attributes are binary relations between entities and values, we consider two attributes as being similar, if they contain similar values. In contrast, relationships are binary relations between entities (sub-entities) and entities (ob-entities), so we say that two relationships are similar if they are associated to similar ob-entities. To assess whether two ob-entities are similar, we measure the similarity between the attributes on the two ob-entities. This results in the following measure:

$$Sim(E_1.R_1, E_2.R_2) = \frac{|E_1.R_1 \cap E_2.R_2|}{|E_2.R_2|} \times H(R_1, R_2)$$

where

$$E_1.R_1 = \{x|x \in E'_1.A \wedge E'_1 \in E_1.R_1\}$$

$$E_2.R_2 = \{x|x \in E'_2.A \wedge E'_2 \in E_2.R_2\}$$

It aggregates the terms in all attributes on the ob-entities, and uses them to represent the terms of the relationship, such that the similarity between relationships can be computed in the same way as that between attributes. In practice, we can selectively use the attributes that are most representative of the ob-entities.

4.3.2 *Quantifying Schema Correlations*

The DSCD algorithm generates a weight for each duplicate candidate and for each schema correlation candidate. While the weights of schema correlations indicate our belief in their correctness, they are not yet the conditional probabilities $P(A|A')$ we need to use in query relaxation. Those probabilities can only be estimated by studying the individual duplicates. We choose a set of duplicates that are most likely to be correct, and use them as samples to assess the conditional probability between two attributes or relationships.

When choosing sample duplicates, we need to find an appropriate trade-off between quantity and quality. If we take too many duplicates, we might include a lot of false duplicates. If we take too few, the sample size is too small to give unbiased estimates. The method to find good threshold will be presented in the experimental section.

With the sample duplicates, we use the maximum likelihood method to estimate the conditional probability between two attributes. It results in

$$P(A|A') = \frac{\sum_{(E_i,E_j)} \frac{|E_i.A \cap E_j.A'|}{|E_j.A'|}}{|(E_i,E_j)|}$$

where each (E_i, E_j) is a pair of duplicates. We use a similar equation to estimate the conditional probability between two relationships:

$$P(R|R') = \frac{\sum_{(E_i,E_j)} \frac{|E_i.R \cap^T E_j.R'|}{|E_j.R'|}}{|(E_i,E_j)|}$$

where $E_i.R \cap^T E_j.R'$ denotes the common ob-entities between $E_i.R$ and $E_j.R'$ and the duplicated ob-entities in $E_i.R \cup E_j.R'$ with weights larger than T. We estimate $P(A|A')$ only when the DSCD algorithm gives a significant weight to the correlation between A and A', because only correlated attributes or relationships will be used in query relaxation.

4.3.3 *Implementation Issues*

When performing the DSCD algorithm to detect duplicates and schema correlations, we need first to obtain the candidates for duplicates and the candidates for schema correlations. Given n entities, there exist n^2 possible pairs of duplicates. Given m attributes, there exist m^2 possible attribute correlations. If we take all of them as candidates for duplicate and schema correlation, run times become infeasible. Therefore it is necessary to prune some candidates that are very improbable to be correct. For pruning duplicate candidates, we can transform each entity to a text fragment by concatenating all its attributes and employ IR techniques (such as cosine similarity) to pre-select those pairs of entities that are similar enough to be real duplicates. As shown later on in experiments, the DSCD algorithm works sufficiently well with a limited number of duplicates. Thus we do not need to feed all possible duplicates to it. When assessing the conditional probabilities, we can use the schema correlations vector C to find more duplicates. To prune the candidates of correlated attributes or relationships, we can resort to various schema matching techniques, which utilize schema description and data types to eliminate the pairs of attributes or relationships that are impossible to be correlated.

Thus, our procedure of identifying and quantifying correlations in malleable schema consists of 3 steps.

1. **Preparation:** Find all pairs of attributes that are possibly correlated; select some duplicate candidates.

2. **Verification:** Perform the DSCD algorithm to find the authentic duplicates and schema correlations.

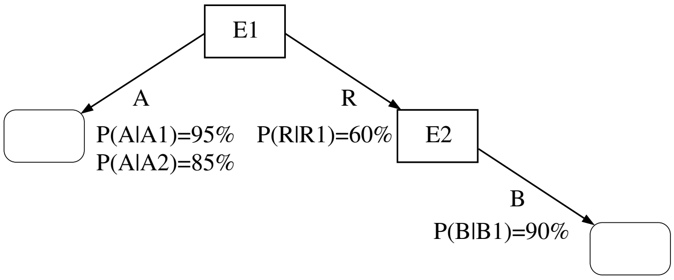


Figure 4.2: Query Example – Q_0

3. **Quantification:** Use the schema correlations vector C to identify more duplicates and use the duplicates to quantify the schema correlations.

Several methods could be used to further optimize the performance of the DSCD algorithm, such as applying a sigmoid function to vectors C and D to amplify the reinforcement effect. We omit the details in this chapter.

4.4 QUERY PROCESSING

With the correlations in a malleable schema, we can properly relax a user query and rank results according to their probability of relevance.

4.4.1 *Query Relaxation Planning*

We assume that the entity-relationship data are stored in relational database. In contrast to ordinary queries in relational database, a query using malleable schema will be relaxed to multiple queries that are executed on different columns or tables. The major performance consideration is to find a plan that executes as less queries as possible to retrieve sufficient relevant results. The optimal plan is to execute relaxed queries in a sequence based on the expected precisions of their result sets. For example, figure 4.2 shows a query $Q_0 = \{E1|E1.A \ni x \wedge E1.R \ni E2 \wedge E2.B \ni y\}$, which searches for an entity E1 which has attribute $A \ni x$ and relationship R to an entity E2 with $B \ni y$. Attribute A is correlated to attributes A1, A2, with probabilities $P(A|A1) = 95\%$, $P(A|A2) = 85\%$. Attribute B is correlated to B1, with probability $P(A|A1) = 90\%$. Relationship R is correlated to R1 with probability $P(R|R1) = 60\%$. With these correlations, Q_0 can be relaxed to queries like $Q_1 = \{E1|E1.A1 \ni x \wedge E1.R1 \ni E2 \wedge E2.B \ni y\}$, $Q_2 = \{E1|E1.A2 \ni x \wedge E1.R \ni E2 \wedge E2.B1 \ni y\}$ and so on. Based on the query relaxation model in Section 4.3, we have $P(Q_0|Q_1) = P(A|A1)P(R|R1) = 95\% \times 60\% = 57\%$, and $P(Q_0|Q_2) = P(A|A2)P(B|B1) = 85\% \times 90\% = 76.5\%$. It means that the results of

Q_2 are more precise than the results of Q_1, so that we should return the results of Q_2 prior to the results of Q_1.

If a query contains n attributes and relationships, and each of them is correlated to m other attributes or relationships, then the query relaxation will end up with m^n relaxed queries. Sometimes, it is infeasible to evaluate all queries. In practice, we try to evaluate the relaxed queries in the order of their precisions until we obtain more than k results or the user is satisfied and stops the processing. To achieve this, we exploit the relationship between the relaxed queries.

Parent-child relation between relaxed queries: Given a query $Q = \{E|A_1 \ni a_1 \wedge A_2 \ni a_2 \wedge ... \wedge A_k \ni a_k\}$. Let $A_i^1, A_i^2, ..., A_i^k$ denote the set of attributes/relationships that are correlated to A_i, and $\forall u < v$: $P(A_i|A_i^u) > P(A_i|A_i^v)$. We say that A_i^{j+1} is the child of A_i^j w.r.t. Q. (A_i^1 is the child of A_i.) Let Q1 and Q2 be two relaxed queries of Q. We say that Q1 is a parent of Q2 iff we can turn Q1 into Q2 by substituting an attribute/relationship in Q1 with its child attribute/relationship.

For example, given Q_0 in Figure 4.2, $Q_4 = \{E1|E1.A1 \ni x \wedge E1.R \ni E2 \wedge E2.B \ni y\}$ is a child of Q_0, and $Q_5 = \{E1|E1.A2 \ni x \wedge E1.R \ni E2 \wedge E2.B \ni y\}$ and $Q_6 = \{E1|E1.A1 \ni x \wedge E1.R1 \ni E2 \wedge E2.B \ni y\}$ are children of Q_4. If we use edges to connect each relaxed query of Q_0 to its child queries, we end up with the partial order graph in Figure 4.3.

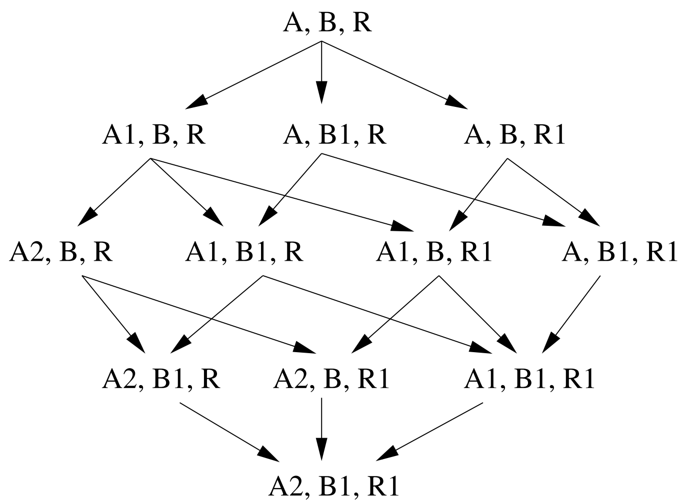


Figure 4.3: Query Relaxation Graph

Proposition 1 *Given a query* Q *and two relaxed queries* Q1 *and* Q2, *if* Q1 *is a parent of* Q2 *then* $P(Q|Q1) > P(Q|Q2)$.

Proposition 1 indicates that a relaxed query always yields better precision than its child queries, so that it should always be evaluated prior to its child queries. Thus, query relaxation using malleable schemas can be performed through the algorithm in Figure 4.4. The algorithm guarantees that the returned top k results are from those

```
func query(Q, k)
   initialize Processed list;
   initialize Ready list;
   initialize Result list;
   add Q to Ready;
   while Ready is not empty and |Result| < k, do
      pick Q_i from Ready with largest P(Q|Q_i);
      add the results of exec(Q_i) to Result;
      add Q_i to Processed;
      foreach Q_j that is a child of Q_i, do
         if Q_j's parents are all in Processed
            add Q_j to Ready;
      end do;
   end do;
   return Result;
end func
```

Figure 4.4: Query Relaxation Algorithm

```
//Cache is emptied at the start of query()
func exec(Q)
   foreach selection σ in Q, do
      if σ's results are not in Cache
         execute σ;
         store σ's results to Cache;
   end do
   search in Cache for the temp results that
   need the least joins to complete Q;
   execute the joins;
   store the results of the joins to Cache;
   return the final results;
end func
```

Figure 4.5: Query Execution Algorithm

queries that yield the best precision. It also minimizes the number of relaxed queries to be evaluated. In [8], the authors proposed a similar algorithm for relaxing queries over multiple relaxation paths in ontologies.

4.4.2 *Query Execution*

With normal relational indexes, each relaxed query can be executed efficiently. However, further optimization can be achieved by caching temporary results of each relaxed query in order that they can be reused by subsequent relaxed queries.

The execution of a relaxed query comprises a number of selection operations and a number of join operations. For example, query $Q' = \{E|E.A \ni x \wedge E.B \ni y \wedge E.C \ni z\}$ can be implemented in the following way:

$$\left[\sigma_{A \ni x}(E)\right] \bowtie \left[\sigma_{B \ni y}(E)\right] \bowtie \left[\sigma_{C \ni z}(E)\right]$$

As another example, the Q_0 in Figure 4.2 can be implemented in the following:

$$[\sigma_{A \ni x}(E1)] \bowtie_{E1.R \ni E2} [\sigma_{B \ni y}(E2)]$$

The temporary results of the selections and joins can be cached and reused. For example, if the above query Q' can be relaxed to $Q'_1 = \{E|E.A1 \ni x \wedge E.B \ni y \wedge E.C \ni z\}$, then the results of $[\sigma_{B \ni y}(E)] \bowtie [\sigma_{C \ni z}(E)]$ can be reused by Q'_1. Therefore, our algorithm of query execution maintains a cache to store the temporary selection and join results. Figure 4.5 presents a greedy algorithm for query execution.

4.5 EXPERIMENTS

This section presents our experimental results, which demonstrate that our query relaxation scheme is able to accurately detect correlations in malleable schemas and effectively perform query relaxation in a fully automatic way. Section 4.5.1 introduces the datasets we used in our experiments. Section 4.5.2 presents the study on detecting duplicates and schema correlations. Section 4.5.3 shows the effectiveness of query relaxation. And Section 4.5.4 demonstrates the efficiency of the query relaxation scheme.

4.5.1 *Datasets and Experiment Setup*

Being a new concept, malleable schemas have not yet been adopted in real world applications. Hence, it is difficult to find adequate datasets using authentic malleable schemas. In order to conduct our experiments, we constructed a dataset by combining real world data from different sources.

Our dataset is a combination of information about movies provided by `www.imdb.com` and DVD/video items crawled from `www.amazon.com`. Though both sources describe similar data they organize and describe their data in different ways. The IMDB dataset contains information on 850,000 movies and TV series, which were joined into a single table containing 32 attributes. The Amazon.com dataset contains information on 115,000 DVDs and VHS videos, which were joined into a table containing 28 attributes. The attributes used to describe the movies and DVDs are very different, but of course closely correlated, so that it adequately simulates the scenarios that use malleable schemas. As there is a big overlap between movies and DVDs, the dataset offers sufficient duplicates for evaluating our query relaxation scheme.

We implemented our system in Java 5. We conducted all the experiments on a PC with a 2.7GHz CPU and 1GB RAM. The DBMS we used is MySQL 4.1.11.

	Amazon	IMDB	confidence	P(A|I)
1	Title	title	0.701	0.619
2	Actors	actors	0.655	0.587
3	Directors	directors	0.642	0.753
4	Languages	languages	0.382	0.711
5	Edit~Review	keywords	0.132	0.086
6	Directors	producers	0.102	0.072
7	Title	akatitles	0.090	0.097
8	ReleaseDate	year	0.081	0.098
9	Directors	writers	0.080	0.173
10	Actors	misc	0.072	0.047
11	Edit~Review	plots	0.067	0.076
12	Actors	writers	0.061	0.098
13	Directors	proddesigners	0.059	0.002
14	Directors	cinematographers	0.054	0.023
15	Title	movielinks	0.049	0.050
16	Edit~Review	taglines	0.046	0.056
17	Actors	producers	0.046	0.072
18	Actors	directors	0.043	0.094
19	Audi~Rating	certificates	0.042	0.136
20	Actors	composers	0.042	0.056

Table 4.3: (Amazon, IMDB) Correlations

4.5.2 *Performance in Detecting Duplicates and Schema Correlations*

4.5.2.1 *The First Run*

Following the 3 steps introduced in Section 4.3.3, we ran our system to automatically detect and quantify the correlations between the IMDB schema and the Amazon schema.

PREPARATION We pre-selected possible schema correlations by assuming that no correlation should exist between attributes of different types. This left us with 59 candidates for attribute correlations. To preselect the duplicate candidates, we randomly picked up 1000 IMDB entities, concatenated the attributes of each entity and compare them with the Amazon entities using a TF-IDF cosine similarity. Then, we selected the 100 most similar pairs as duplicate candidates. To accelerate the process, we only used the most important and representative attributes which satisfied the following general criteria:

1. typed as text;
2. limited length ($\leqslant$ 100Bytes);

	IMDB	Amazon	confidence	P(I|A)
1	title	Title	1.000	0.923
2	actors	Actors	0.794	0.720
3	directors	Directors	0.666	0.792
4	akatitles	Title	0.380	0.303
5	languages	Languages	0.348	0.621
6	distributors	Publisher	0.264	0.335
7	distributors	Manufacturer	0.264	0.335
8	distributors	Label	0.264	0.335
9	distributors	Studio	0.264	0.335
10	movielinks	Title	0.262	0.296
11	producers	Directors	0.152	0.186
12	writers	Directors	0.104	0.259
13	year	ReleaseDate	0.081	0.098
14	technical	AspectRatio	0.073	0.108
15	actors	Creators	0.067	0.063
16	actors	Directors	0.066	0.135
17	proddesigners	Directors	0.059	0.003
18	certificates	Audi~Rating	0.058	0.220
19	cinematographers	Directors	0.058	0.029
20	plots	Edit~Review	0.036	0.052

Table 4.4: (IMDB, Amazon) Correlations

3. high selectivity ($\geqslant 0.05$);

4. frequently used (more than 30% are not null).

Besides the 100 duplicate candidates, we also included 100 non-duplicate candidates (random pairs) as baseline.

VERIFICATION We ran our DSCD algorithm on the pre-selected schema correlations and duplicates/non-duplicate candidates. The algorithm computed a confidence value for each schema correlation indicating the extent of belief that the match is correct. We ranked all correlations based on the confidences, and list the top 20 correlations for both directions in Tables 4.3 and 4.4 respectively. Please note that the confidence assigned to correlation (*Amazon.attribute, IMDB.attribute*) is different from that assigned to (*IMDB. attribute, Amazon.attribute*). The former shows how much the system believes that an Amazon attribute contains an IMDB attribute, and the latter the other way around.

We can see from Tables 4.3 and 4.4 that (*Title, title*), (*Actors, actors*), (*Directors, directors*) and (*Languages, languages*) are the most obvious correlations. This is consistent with the results of a manual inspection where we can actually find those attributes to be semantically equivalent. It is interesting to see that the algorithm also recognizes the direction of correlations. For example, it gave more confidence to (*IMDB.title, Amazon.Title*) than to (*Amazon.Title, IMDB.title*), as most IMDB titles contain the release years of movies while the Amazon titles do not. Apart from the most obvious correlations, our system found other interesting correlations as well. For example, the *directors* in Amazon is correlated to the *writers* and the *producers* in IMDB, as it is very common that the director of a movie also acts as writer and as one of the producers. Moreover, there exist correlations between the *EditorialReview* in Amazon and the *keywords*, *plots*, and *taglines* in IMDB, because these attributes are all related to summaries of movies. A semantically meaningful query relaxation should thus be performed on these best matching attributes.

Besides schema correlations, the algorithm verified the pre-selected duplicate candidates too. Figure 4.6 shows the distribution of the confidences the algorithm gave to the total of 200 duplicate candidates and non-duplicate candidates. We can see that out of the 100 pre-selected duplicate candidates only around 22 are likely to be true (with confidence larger than 50%). We manually assessed the 22 pairs of entities and found that they are indeed all real duplicates. We can also see that the DSCD algorithm was able to distinguish the true duplicates and the false duplicates quite clearly, as the confidence values assigned to true duplicates and those for false duplicates differ significantly.

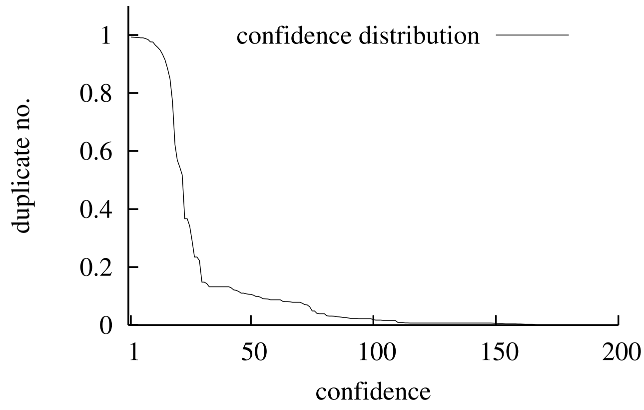


Figure 4.6: Confidence Distribution over Duplicates

QUANTIFICATION In this step we estimate the probabilities that will be used for query relaxation, namely P(*Amazon.attribute*| *IMDB.attribute*) and P(*IMDB.attribute*| *Amazon. attribute*). We used the K-Mean algorithm to automatically cluster the duplicate candidates into true duplicates and false duplicates based on their confidence values, and set the median of the two clusters as a threshold for duplicate detection. Then, we used the schema correlations discovered previously to find 200 more duplicates that have confidence larger than the threshold, and finally applied the functions in Section 4.3.3 on these duplicates to calculate the requested probabilities. The results are presented in Tables 4.3 and 4.4. They show that although confidences and probabilities are often similar, they are not completely consistent. For example, the *Languages vs. languages* are given confidence 0.3, but the corresponding probability is bigger than 0.6. Since both attributes have limited possible values, it is very likely that they are similar on real duplicates, even if they are not significantly correlated.

4.5.2.2 *Result Quality of the DSCD Algorithm*

After the first run, we conducted a set of experiments to evaluate the result quality of the DSCD algorithm, namely how precisely it can detect the schema correlations and duplicates. The precision of such detection determines the overall performance of our query relaxation scheme. To make such an evaluation, we used the the schema correlations (and their confidence values) discovered by the algorithm to detect additional duplicates in the database. The precision of the detection measures the accuracy of our algorithm in finding both schema correlations and duplicates.

In order to obtain objective results, we divided the 100,000 Amazon items into two sets, each containing 50,000 items. The first set was used as a training set to perform the DSCD algorithm. Then the discovered schema correlations were applied to the second set (test-set) to measure the accuracy in detecting duplicates. In particular, we used the pre-selection criteria of the first run to find 800 duplicate candidates from the Amazon test set and the IMDB dataset. Then we manually inspected the 800 pairs of items and picked out all pairs that are real duplicates . This resulted in 153 real duplicates. After that, we randomly selected 1000 pairs of entities from the Amazon test set and the IMDB dataset and in another manual inspection picked out all pairs that are not duplicates, which ended up with 919 non-duplicates. Finally, the 153 real duplicates and the 919 non-duplicates constituted the final test-set for our evaluation.

We used two metrics for our evaluation. The first metric was the mean average precision (MAP), which has been widely used in IR. It measures how accurately our system can order the pairs of entities in the test-set according to their likelihood of actually being duplicates. While this measure is widely applicable, it still does not clearly show whether the system can find an appropriate threshold to clearly distinguish between real duplicates and non-duplicates. Therefore, we also used classical precision and recall to measure how accurately our system can make such distinction. After performing the DSCD algorithm in the training set, we again used the K-Mean algorithm to cluster the duplicate candidates into real duplicates and non-duplicates based on their confidence values, where we used $K = 2$ and set the initial centroids to the max and min confidences respectively. Afterwards we used the central point of the two clusters as the threshold to distinguish between true duplicates and false duplicates in the test-set.

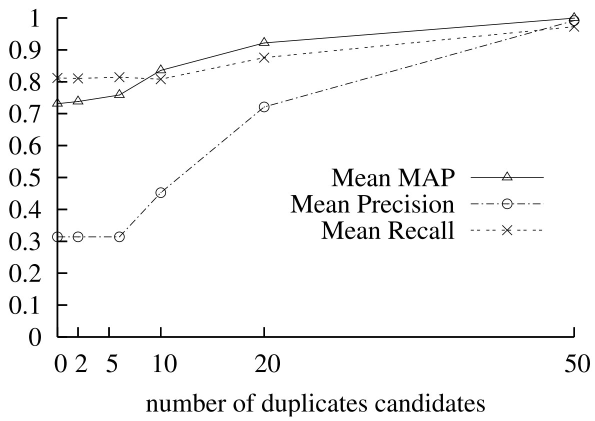


Figure 4.7: Sensitivity to Pre-selected Duplicates

In the first set of experiments, we used the same criteria as in the first run to pre-select a certain number of duplicate candidates (varying from 0 to 50) from the Amazon training set and the IMDB dataset, and mixed them with 150 random non-duplicate candidates. We applied the DSCD algorithm on these duplicate and non-duplicate candidates, and then used the output schema correlations to detect the duplicates in the test-set. Figure 4.7 shows the performance of the duplicate detection. When there was no duplicate candidates, the DSCD algorithm was only able to detect correlations like *Language vs. language* and *ReleaseDate vs. year* which contain attributes with small value ranges. Hence, it was not really able to find any duplicates with these correlations. Though it shows a MAP of 75%, the precision is only 30%, which is very bad given that there are many actual duplicates in the test-set. However, the accuracy of the DSCD algorithm increases very fast as the number of duplicate candidates grows. As shown in Figure 4.7, with 25 duplicate candidates, the MAP grows close to 1 and the precision and recall grows close to 80%. With 50 duplicate candidates, the precision and recall are already 97%, which means that our algorithm can almost perfectly detect all duplicates. According to the results of the first run, around 20% of the pre-selected duplicate candidates are true duplicates. This indicates that the DSCD algorithm can indeed accurately detect correlations between the Amazon and IMDB schemas with a small number of duplicates.

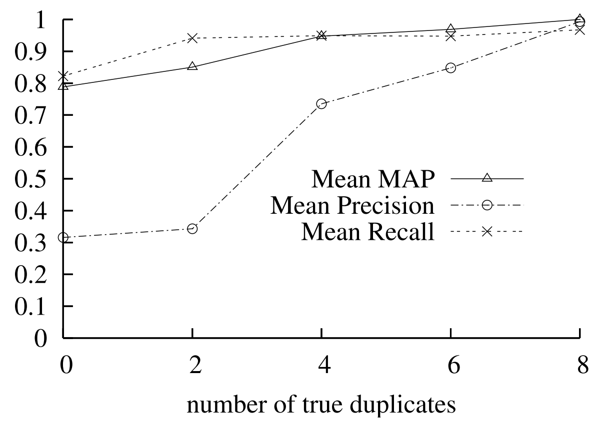


Figure 4.8: Sensitivity to True Duplicates

In the second set of experiments, we intended to study the effects of true duplicates to the results of the DSCD algorithm. We used the same criteria as in the first run to pre-select a certain number of duplicate candidates, and manually changed the number of the true duplicates inside, such that the number of true duplicates ranged from 0 to 8 and the total number of duplicate candidates remain at 50. We ran the DSCD algorithm on the 50 duplicate candidates and another 150 non-duplicate candidates, and again used the results to detect duplicates in the test-set. The performance of the detection is shown in Figure 4.8. As we can see from the results, the true duplicates play an important role in the DSCD algorithm. When there were no true duplicates in the 50 pre-selected duplicate candidates, the outcome of the DSCD algorithm was hardly able to detect duplicates in the test-set (a precision of 0.3). When the number of true duplicates increased to 8, even though they were still few, the precision quickly increased close to 1.

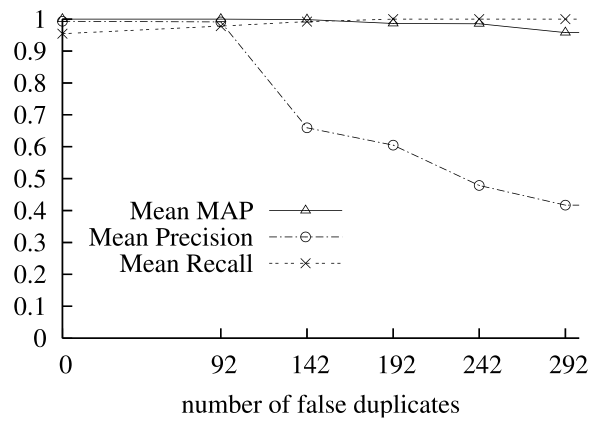


Figure 4.9: Sensitivity to False Duplicates

The performance of the algorithm is also influenced by the false evidence in the pre-selected duplicate candidates. In the last set of experiments, we limited number of true duplicates in the pre-selected duplicate candidates to 8, and manually increased the duplicates candidates which were not true but had high pre-selection scores. As shown in Figure 4.9, both MAP and precision-recall in the final duplicate detection start to drop when the number of the false duplicates increases to a certain point (between 92 and 142). This means that when there are too many false duplicates, they will dominate the effect of true duplicates and lead the system to believe a set of wrong schema correlations. This actually happens to our human belief system too. Fortunately, our algorithm is still very tolerant to the false evidences. As we can see from Figure 4.9, only when there are more than 92 false duplicates (10 times more than the true duplicates), it started to take effects. In practice, false duplicates can also be relatively decreased by raising the criteria of pre-selection.

4.5.2.3 *Schema Correlation Quantification*

We also conducted a set of experiments to study how accurately our system can quantify the schema correlations. We used the schema correlations and the threshold obtained by the DSCD algorithm to find duplicates from 50 pre-selected candidates from the Amazon training set and the IMDB dataset. Then, we used the functions in Section 4.3.3 to estimate the probabilities based on those duplicates. We varied the number of duplicate candidates and repeated the process for several times. Figure 4.10 shows the correlation-coefficients between the estimated probabilities and the probabilities we obtained from the 153 real duplicates in the test-set. As expected, when there are more duplicate candidate, our system can more precisely identify duplicates and thus obtain better estimation of the probabilities.

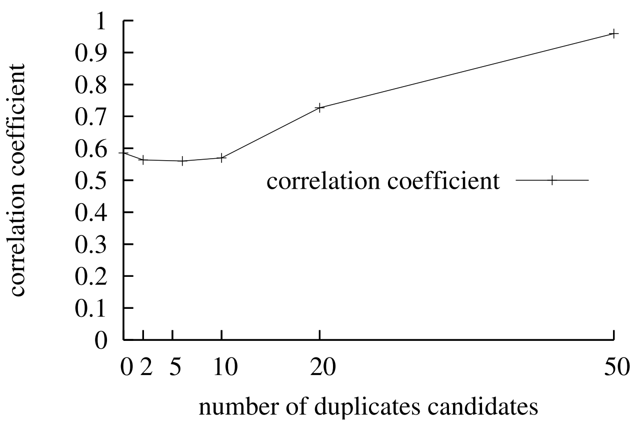


Figure 4.10: Accuracy in Correlation Quantification

4.5.3 *Effectiveness of Query Relaxation*

To test our query relaxation scheme, we used the correlations discovered in the first run and issued a set of queries posed to the original Amazon scheme to be relaxed to attributes of the IMDB database. We observed how the queries were relaxed and results were retrieved.

For example, we searched for the movie "Titanic (1997)" which was very popular ten years ago. As there are a lot of movies whose title contains "Titanic", we used the director "James Cameron" as an additional selection criterion, and the query managed to retrieve the unique answer "Titanic (1997)" from the Amazon data. Then, we issued the same query to the IMDB data. Using the schema correlations detected in the first run, our system relaxed the query to a number of relaxed queries and executed them in a sequence based on their probabilities of getting correct results. Table 4.5 shows the relaxed queries, as well as the corresponding results retrieved by them. We can see that the *Title* and *Directors* in Amazon was first extended to the *title* and *directors* in IMDB. Then, attribute *directors* was relaxed to *writer* and *producer*, and attribute *title* to *akatitle* and *movielinks*. All the relaxed queries actually retrieved the correct result "Titanic (1997)". Thus, even though if the attributes *title* and *directors* were missing in IMDB, we can still find the movie by using the correlated attributes. As expected, the query relaxation also introduced some incorrect results, such as "Ghosts of the Abyss" and "Last Mysteries of Titanic", which are relevant to "Titanic" and "Cameron" but not the exact answer we wanted.

As another example, we looked for the famous Disney cartoon "Mulan (1998)", which was adapted from a Chinese legend. Assume we had forgotten the title of the movie and only remembered that it is a Disney work about a Chinese story. Thus, we had to search by the *EditorialReview* attribute using keywords "Disney" and "Chinese". The result we got from Amazon was exactly the movie "Mu-

Original Query in Amazon	*results (Title)*
Title ∋ "Titanic" Directors ∋ "Cameron"	Titanic

Relax query in IMDB	*results (title)*
title ∋ "Titanic" directors ∋ "Cameron"	Titanic
title ∋ "Titanic" writers ∋ "Cameron"	Titanic
akatitles ∋ "Titanic" directors ∋ "Cameron"	Titanic Ghosts of the Abyss
title ∋ "Titanic" producers ∋ "Cameron"	Titanic Last Mysteries of Titanic Titanic Explorer
movielinks ∋ "Titanic" director ∋ "Cameron"	Titanic Ghosts of the Abyss
akatitles ∋ "Titanic" writers ∋ "Cameron"	Titanic
movielinks ∋ "Titanic" writers ∋ "Cameron"	Titanic
akatitles ∋ "Titanic" producers ∋ "Cameron"	Titanic Ghosts of the Abyss Titanic Explorer

Table 4.5: Query Relaxation Case One

lan (1998)". The process and results of performing the same query on IMDB are shown in Table 4.6. We can see that the attribute *EditorialReview* in Amazon was relaxed to the corresponding attributes *keywords*, *plots* and *taglines* in IMDB and retrieved three results. Among the results, we still found "Mulan (1998)".

To further evaluate the effectiveness of query relaxation, we compared it against keywords search. We selected the most frequently used three attributes in Amazon, namely "Title", "Actors" and "Directors", and created query templates using arbitrary combinations of these attributes. For example, $\{E|\text{Title} \ni x\}$ and $\{E|\text{Actors} \ni x \wedge \text{Director} \ni y\}$ are two query templates. Thus, we ended up with 7 query templates. We used these 7 templates on the Amazon items in the 153 manually evaluated real duplicates to create queries. For each of these Amazon queries, we conducted it on the IMDB data to see if it can effectively retrieve the corresponding duplicate in IMDB. One way to conduct the query is to use our query relaxation scheme. The other way is to combine all the terms in the Amazon query to form a key-

Original query in Amazon	*results (Title)*
Edit~Review ∋ "Chinese" Edit~Review ∋ "Disney"	Mulan

Relax query in IMDB	*results (title)*
keywords ∋ "Chinese" keywords ∋ "Disney"	American Dragon
keywords ∋ "Chinese" plots ∋ "Disney"	Mulan
plots ∋ "Chinese" keywords ∋ "Disney"	Aladdin
plots ∋ "Chinese" plots ∋ "Disney"	*null*
keywords ∋ "Chinese" taglines ∋ "Disney"	*null*
taglines ∋ "Chinese" keywords ∋ "Disney"	*null*

Table 4.6: Query Relaxation Case Two

words search query, and conduct it on all the attributes of the IMDB table.

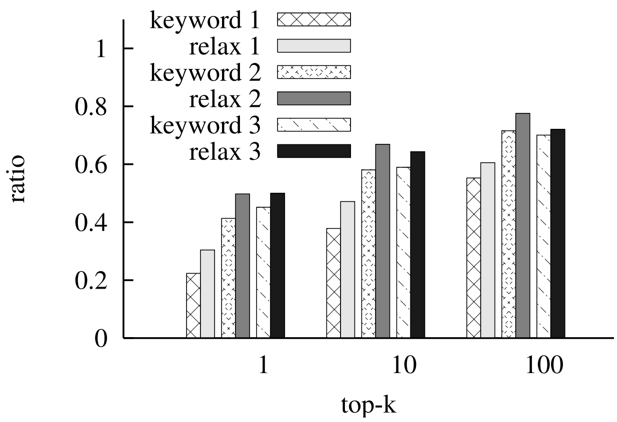


Figure 4.11: Chance of Finding Intended Item in Top-k Results

In our experiments, we used the 7 templates and the 153 Amazon items to automatically generate 3000 queries. Each query was created by filling every query attribute with 1 to 3 terms selected randomly from the corresponding attribute of an Amazon item. We use "keyword k" to denote the queries that are conducted as keywords search and have k terms in each query attribute. We use "relax k" to denote the queries being conducted as query relaxation. We compare the results of query relaxation and keywords search in Figure 4.11 and Figure 4.12. Figure 4.11 shows the fractions of queries that can retrieve the intended IMDB items within the top 1, top 10 and top-100 results respectively. We can see that query relaxation outperforms keywords search in each type of queries. Figure 4.12 compares the mean average precisions of query relaxation and keywords search with varying number of terms in each query attribute. We can see that query relaxation clearly outperforms keywords search. As shown in both figures, the precision of query results increases with the number of terms in each query attribute. This is because increasing the number of terms will increase the selectivity of queries, so as to reduce false positive results.

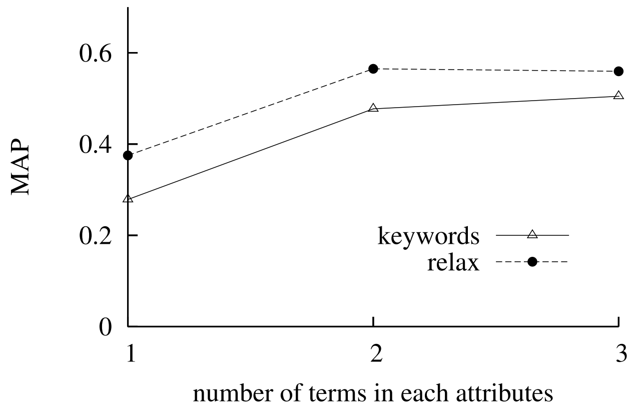


Figure 4.12: Result Accuracy v.s. Number of Terms

We found that query relaxation does not always outperform keywords search in every individual query. The main reason is that query relaxation sometimes returns empty result, as it requires that each term in the query should appear in some attributes. In contrast, keyword search always returns non-empty results, as it is based on cosine similarity. Therefore, we believe that the performance of query relaxation could be further improved by softening its conditions. This will be investigated in future research.

4.5.4 *Efficiency of Query Relaxation*

We did some initial experiments to evaluate the efficiency of our query relaxation scheme. We generated 500 Amazon queries using the Amazon items. Each query contained 1 to 5 attributes. Similar to previous experiments, we performed the 500 Amazon queries on the IMDB data using both query relaxation and keywords search. We limited the number of schema correlations that are used in query relaxation, by setting a threshold to their confidence. The thresholds we used were 0.2, 0.1 and 0.05 respectively. Figure 4.13 shows the running times of various types of queries. As expected, the running

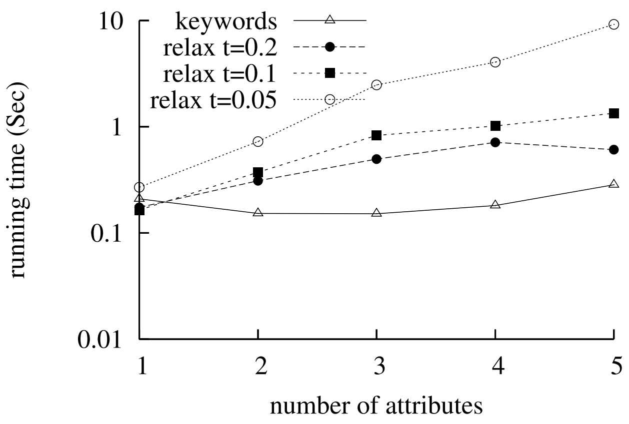


Figure 4.13: Query Running Time

time of query relaxation can be increased by either decreasing the threshold or increasing the attributes in each query, because both will increase the number of relaxed queries to be executed. In particular, the running time grows exponentially with the number of attributes. Nevertheless, the exhibited performance was overall acceptable. With threshold equal to 0.1, query relaxation can already return very satisfactory results, and most queries can be completed within 1 second. When the threshold is equal to 0.05, all queries can be completed in 10 seconds. Moreover, as the relaxed queries are executed progressively, users can see early results in a very short time.

4.6 RELATED WORK

Extensive research have been done on automatic schema matching [97, 38, 79, 69]. As stated earlier, although their results could help in discovering the correlated elements in a malleable schema, they do not give semantically accurate quantifications of those correlations. To the best of our knowledge, such quantification can only be achieved by studying the co-occurrences of the correlated schema elements, which are embodied by the duplicates in real data. In [12], the authors presented the idea of using duplicates to discover schema matches. However, they did not address methods of using duplicates to quantify the schema matches so that they can be used in query. Moreover, they did not explore how to use discovered schema matches to reinforce duplication detection. Another interesting work is GLUE [39], a system that employs machine learning techniques to find mappings between different ontologies. Similar to our strategy, it uses data instances to statistically assess the correlations between concepts. However, its mechanism only applies to categorical information, and cannot be used to quantify the correlations between attributes and relationships in malleable schemas. In [36], the authors proposed to use content similarity between attributes for automatically detecting foreign key relationships. However, their approaches are not able to differentiate attributes that are conceptually different but similar in contents. Our approach does not suffer from this problem, as it uses duplicates for detecting schema correlations.

Duplicate detection in databases [41] is mainly used in data cleaning. The existing techniques usually assume a common schema and attempt to find tuples which are slightly different but refer to the same real world entity. When data schemas are different, they normally require a data transformation step [78] to integrate the data into a common schema before performing detection. In contrast, our approach in this chapter takes advantage of the association between duplicates and schema matches to discover both of them simultaneously.

Query relaxation was also an important research topic in cooperative database systems [61, 30, 73], which are designed to automatically relax user queries when the selection criteria are too restrictive to retrieve enough results. Such relaxation is usually based on user preferences and values, and it is seldom applied to schemas. Recently, these techniques have been extended to relax XML queries [4, 75] using the hierarchical structures of XML data. In [8], the authors proposed a framework for relaxing user requests over ontologies. In contrast to those proposals, our work is not concerned about user preferences but the ambiguity in malleable schemas.

Recently there has been a lot of research on approximately querying XML data [51, 25, 5, 95, 76]. These techniques mainly rely on the tree structures (which does not exist in malleable schema) in XML. To relax user queries, they gradually move the selection criteria from leaves of the tree to the root. However, they normally ignore the possible correlations outside of the tree hierarchy. In this chapter, we proposed a method to automatically discover the correlations within a data schema and use them for query relaxation.

4.7 CONCLUSION

In this chapter, we presented a query relaxation scheme that enables effective search using malleable schema. We introduced a probabilistic model that allows us to perform query relaxation in a reasonably simple way. We proposed the DSCD algorithm which utilizes the duplicates in data to effectively detect and quantify the correlations within a malleable schema. We discussed the key issues in processing the query relaxation. We conducted extensive experimental study using real dataset to evaluate the performance of our system. Much investigation still needs to be done, in order for malleable schema to be used by real world applications. The future research could be focused on storage management, query interface and flexible mechanism for updating malleable schemas.

5 LINEAGE ANNOTATIONS TO TRACE AND REVISE IDENTITIES OF EVOLVING ENTITIES

5.1 INTRODUCTION

In this dissertation we addressed so far two challenges arising when dealing with digital entities on the web of today: schema discovery addressed in Chapter 3 enables users to formulate queries, and query relaxation addressed in Chapter 4 allows to effectively expand queries and rank entities when the schema is vague, allowing to adopt a malleable schema and pay-as-you-go information integration approach [40]. As efforts are made by different actors on the web to integrate information, it is desirable to be able to communicate it to other information consumers. Using unique identifiers like URIs is one way to communicate identity of entities in a way that is efficiently machine processable: two entities, i.e., two object representations, can be quickly recognized by a machine as representing the same object by simple equality of their identifiers. However, in a highly decentralized environment like the web, communication of an entity's identity with URIs suffers from duplicity: several distinct URIs are used to refer to a same object, such that one of the two URIs should be used instead of the other, which information need to be communicated to other web actors. In this chapter we propose to annotate entities with lineage information that allows to detect potential identity conflicts and reduces communication needs to resolve them.

This work was first developed for another but related scenario where communicating identity conflicts is necessary: the *Entity Name System* (ENS) [15]. Deciding whether two entity descriptions refer to the same entity is a difficult problem, and the ENS aims at providing entity matching as a service: instead of deciding on their own whether two entity descriptions refer to the same entity, web actors can give a description of an entity as, for example, persons, locations or products to the ENS, which provides globally unique URIs for such real-world entities. Even if the ENS is a centralized solution to the entity matching problem, it does not solve to the problem of communicating identity revisions. Because entity descriptions available to the ENS for deciding whether two entity descriptions refer to the same real-world entity (i.e., entity identity) are changing over time, the system sometimes has to revise its past decisions: one real-world entity has been given two different URIs [1] (e.g., `http://dbpedia.org/resource/`

1 It is possible to find such duplicates by using a semantic search engine such as, for example, `http://sindice.com/search`

Tim_Berners-Lee and http://data.seman-ticweb.org/person/tim-berners-lee) or two entities have been attributed the same URI. For example, http://dbpedia.org/page/Paris could be the URI assigned to the city in France but, in DBpedia, it currently "disambiguates" (that is, refers) to 57 different entities. This could happen when not enough information is available when assigning a URI to an entity. When other entities called Paris are discovered new URIs should be assigned and the information be propagated. Thus, to be sure to be up-to-date with respect to the ENS state, distributed clients would need to constantly request for the latest identifier of the entities they are processing, since it could be that the ENS identified two identifiers to refer to the same object or one identifier to refer to two distinct object. This obviously does not scale well and is in addition very costly in terms of communication and computation. In this chapter we propose to annotate identifiers with additional lineage information allowing the clients to detect some of the identifier conflicts, solve them locally in some cases and only contact the ENS in the other cases. Our main research question in this chapter is how to propagate entity decision revisions to the clients, which make use of the URIs provided by the ENS.

Providing a solution to the problem of propagating entity decision revisions is a crucial aspect of Semantic Web when the goal is to enable the Web of Entities. When software applications exchange information about entities it is essential to uniquely resolve the entities so that all the agents in the Web refer to the same real world entity using the same name, that is, the same URI. As entities can change over time, URI can be used multiple times by different sources to refer to the same entity, it is important to provide a framework for handling such identity revisions.

In this chapter:

- We describe a model for entity identifiers evolution on the Web (Section 5.2).

- We propose different labeling schemes allowing to dramatically reduce the number of identifier update requests to the Entity Name Service the client needs to perform, while preserving a reasonable uniqueness quality for its identifiers (Section 5.4).

- We perform extensive experiments measuring the quality of identifier uniqueness. The experiments, using both artificial and real revision history from Wikipedia, also measure the drop in identifier update requests when using the proposed labeling schemes (Section 5.5).

- We present related work in Section 5.6.

- We conclude the chapter and describe possible future work in Section 5.7.

5.2 APPLICATION SETTING AND SCENARIOS

5.2.1 *Motivation*

It is known that "the Semantic Web can become an open and scalable space for publishing knowledge (in the form of RDF data) only if there will be a reliable (and trustworthy) support for the reuse of URIs" [16]. The ENS is providing such an infrastructure to the Semantic Web users.

It is also clear that in the current Web there is a proliferation of different URIs for the same real world entity. It is then necessary to provide a way of notifying the Semantic Web users of changes in the URIs. For example, a person such as Tim B. Lee has many different URIs assigned by different sources: it is desirable for applications and/or people on the Web referring to him to use the same identifier.

In this chapter we present methods for the propagation of identity decision revisions taken by the ENS. In the following section we describe the context of identity decision revisions and introduce different approaches for propagating such information to the interested clients.

5.2.2 *The Entity Name System.*

The main goal of the ENS is to enable the Web of Entities. This is accomplished by supporting the use of globally unique identifiers (URIs) for entities. It is important to clarify that the information about entities (i.e., attribute names and values) stored in the ENS serves solely the purpose of distinguishing entities between themselves. The ENS is defined in [15] as: "a service which stores and makes available for reuse URIs for any type of entity in a fully decentralized and open knowledge publication space." The main functionalities of the ENS are: search for the identifier of an entity, generation of entity identifiers, matching entities present in the repository with external ones, and ranking entities by similarity to a given one. Notice that the question of how the ENS actually disambiguates entity identities is out of scope of this chapter and will therefore not be discussed here.

5.2.3 *Setting*

We now describe by way of an example the possible operations that can be applied to an entity. A knowledge-base system, at some point in time, might have a wrong representation of an entity: for example, initially, a system has the knowledge that a person is called John Doe, but, after having acquired more evidence, the system knows that another entity representation, with name John J. Doe, refers actually to the same entity. In this case the two entity representations as well as their identifiers need to be *merged*. Also, the entity itself can change over time (e.g. semantic evolution, evolution of documents [11]): for example, the attribute *affiliation* of a person can change over time. For these reasons, we need to define the possible changes that the entity identifiers might undergo. The following operations can be applied on an entity:

CREATION When an entity is first encountered, a new ID is generated.

SPLIT When the system discovers that the same entity representation describes two different real world entities, two new IDs have to be created. For example, the system knows that a person is called "Andrea Rossi" but, at some point, it discovers that there are actually a man and a woman with the same name, in the dataset.

MERGE When two entities are matched, and recognized to be the same, they are merged, and the same has to be done to their IDs.

In the following we present a real word example where the propagation of Identity Decision Revision (IDR) is important.

5.2.3.1 *Wikipedia history.*

We can see Wikipedia as a living information repository about entities. Therefore, it is possible to use it as a scenario of IDR propagation. Entities in Wikipedia have their own pages which can grow over time. It is also true that creation, split, and merge operations appear in this collection. New pages are created, articles about similar concepts are merged together, and articles which grow too much are usually split in two (or more) different ones. It is then easy to see the need that other pages linking to a merged/split page have of knowing about this decision. Links have to be changed and in Wikipedia one approach is to use the disambiguation pages: pages about a general concept leading to all the alternative uses of that term.

5.2.4 *Models for Propagating Identity Decision Revisions*

When we want to propagate the information of IDRs there are two main possibilities of doing it: by pushing or by pulling such information.

In the *push* model the ENS server notifies all the clients of an IDR. In this scenario the ENS must be aware of all the clients using a given URI. It is easy to see that this solution is not feasible for all clients on the Semantic Web[2]. It is impossible for the ENS to keep track of which

2 The users on the Web are estimated to be 1.5 billions in 2008. Source: http://www.internetworldstats.com/stats.htm

client uses which ID. More, in this case there would certainly be a problem of network traffic overload: a lot of communication would be needed for propagating IDRs.

In the *pull* model a client has to ask the ENS about changes for a given ID. In this case it is a duty of the client to decide when it is time to request for an update of the ID. That is, the client has to understand when an ID is obsolete. For example, if the client receives an e-mail which contains the ID of an entity, it has to discover that it is a newer version of an ID it already has (e.g., as a result of a split operation on the ENS). In this case it is necessary to query the ENS for an up-to-date version the ID.

It is clear that the trade-off between the two models is set by how often the IDs change. If there are few IDRs, then the push approach is more efficient as all the effort is centralized.

5.3 CONCEPT DEFINITIONS FOR IDENTITY REVISION MANAGEMENT

In this section we define the concepts and relative notation which are needed for describing the approaches presented in Section 5.4.

In this chapter we assume that the ENS, named in the following as *server*, is one central machine only. It will be future work to consider the effects of a distributed ENS platform on the identity management.

Additionally, several applications using entity identifiers are present and are asynchronously submitting entity resolution requests to the server. These can be end-user application like, e.g., e-mail clients or batch processes as web crawlers. We refer to applications that submit requests to the server as *clients*. The server will respond to requests using the knowledge available in the server's *central repository*: a container of all the up-to-date entities and identifiers. When more than one ID is up-to-date for a given deprecated ID, the ENS has to use *entity matching* [3] techniques to disambiguate between those candidates. Moreover, each client has its own *local repository* containing only the identifiers known by the client, that is, all the entities encountered up to now.

Clients are exchanging information about entities and their IDs as well. This means that clients are encountering new IDs as they receive new information from other clients.

As presented in Section 5.2.3, split and merge operations are possible on entity identifiers. We define as *Identity Revision Graph* the directed acyclic graph representing the history of one or more entity identifiers. Using the part of such a graph that it knows, a client can detect deprecated identifiers with no need of sending a request to the server.

3 Also known as record linkage, or deduplication.

5.4 LIMITING THE NUMBER OF UPDATE REQUESTS

A quick computation shows that if 50,000 clients request updates to the server for 3,000 IDs only once a day, it represents more than 1,500 requests per second, which is already significant, and it would be interesting to be able to limit this number of request. We propose to do so by trying to update only the IDs which most need to, with the idea that, as long as one ID in a client's local repository of entities refer to only one entity in this repository, it does not need an update since local uniqueness would be preserved. This is true whether the ID in question has been deprecated (split or merged) on the ENS server.

In the following, we first present the Prime Numbers Labeling Scheme for Directed Acyclic Graphs (PLSD) proposed by Wu et al. in [112] which we use in Section 5.4.2 to show how to reduce the number of update requests by annotating IDs. Then in Section 5.4.4 we present how we can add a label to further reduce the number of update requests by allowing clients update certain labels without even contacting the server. In Section 5.4.3 we present a technique equivalent to the lineage preserving ID labeling introduced in Section 5.4.2 but simpler. And finally, for completion, we present the baseline used later in the experiments, and how it compares to the lineage preserving labeling schemes.

5.4.1 *PLSD: Prime Numbers Labeling Scheme for Directed Acyclic Graphs*

Wu et al.[112] proposed a labeling scheme for transitive closure computation on DAGs. Computing a transitive closure in a graph is used for identifying all ancestors (or descendants) of a given node in the graph. We describe hereafter the Lite version of Prime Numbers Labeling Scheme for DAGs (PLSD) proposed by Wu et al.

5.4.1.1 *Prime number factorisation.*

PLSD is based on the well-known fundamental theorem of arithmetic (or unique prime factorisation theorem), from number theory, which states that any natural number greater than 1 can be written as unique product of prime numbers. The idea of PLSD is then to assign a unique prime number ID to each vertex in the DAG, and label each vertex with the product of its ancestors' prime number IDs. Thus, given a vertex label, by performing a prime number decomposition, we can retrieve the IDs of all its ancestors. More formally this gives:

Definition 15 *Let* $G = (V, E)$ *be a DAG, with* V *a set of vertex and* E *a set of edges. We define the bijective function* $p : V \to \mathbb{N}$ *such that* $p(v)$ *is prime, for* $v \in V$.

Definition 16 *Let* $G = (V, E)$ *be a DAG. We define the label of a vertex* $v \in V$ *as* $L(v) = (c[v])$ *where*

$$c[v] = p(v) \cdot \begin{cases} \prod_{v' \in \mathrm{parents}(v)} c[v'], & \mathrm{in-degree}(v) > 0 \\ 1, & \mathrm{in-degree}(v) = 0 \end{cases}$$

with $\mathrm{parents}(v)$ *being the set of vertex parents of* v'.

Now, given a vertex v and its label $c[v]$, the fundamental theorem of arithmetic assures that there exists a unique prime factorisation of $c[v]$ such that

$$c[v] = p(v) \cdot \prod_{v' \in \mathrm{ancestors}(v)} p(v')^{m_{v'}} \tag{5.1}$$

where $\mathrm{ancestors}(v)$ is the set of all the ancestors of v, for some $m_{v'} \in \mathbb{N}$; which gives us v's ancestors' IDs. An example is shown in Figure 5.1.

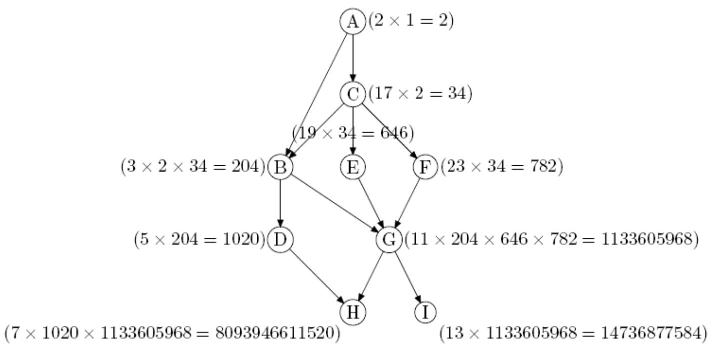


Figure 5.1: PLSD-Lite on a DAG (from [112])

5.4.1.2 *Optimization.*

As we can see in Figure 5.1, the ancestor labels $c[v]$ are growing exponentially due to the $m_{v'}$'s from Equation 5.1 which are in general greater than 1. This requires a storage space such that it is not much more space-efficient than simply storing the ancestors' IDs as a list for each vertex. To avoid this, Wu et al.[112] proposed to force the $m_{v'}$'s to be 1. This can be done by modifying the computation of $c[v]$ described in Definition 16 as follows:

$$c[v] = p(v) \cdot \begin{cases} \mathrm{lcm}\left(c[v'_1], \dots, c[v'_n]\right) & \text{if } \mathrm{in-degree}(v) > 0 \\ & \text{and } v'_1, \dots, v'_2 \in \mathrm{parents}(v) \\ 1 & \text{if } \mathrm{in-degree}(v) = 0 \end{cases} \tag{5.2}$$

where $lcm(a_1, a_2, ..., a_n)$ is the *least common multiple* of the natural numbers $a_1, a_2, ..., a_n$, with the special definition of $lcm(a) = a$. Figure 5.2 shows how the DAG depicted in Figure 5.1 is labeled using ancestor labels as defined in Equation 5.2.

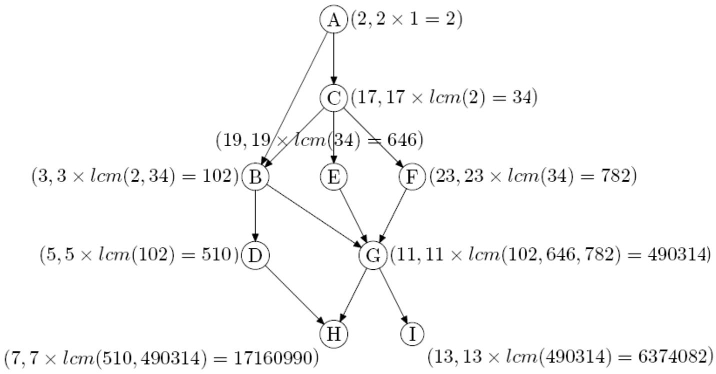


Figure 5.2: PLSD-Lite on a DAG using least common multiple (modified from [112])

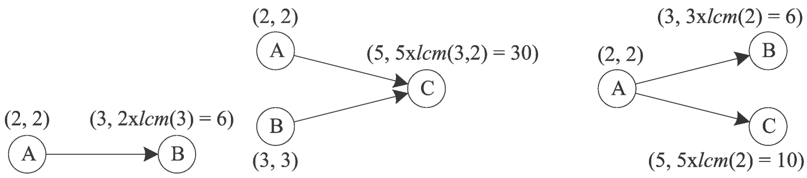
Wu et al. also present a topological sort before labeling a DAG, and define a PLSD-Full labeling scheme for the computation of a vertex's parents, but we omit their description here since they are not useful or applicable in our situation. The version we presented above is referred to as PLSD-Lite. See their paper[112] for more details.

5.4.2 *Lineage Preserving ID Labeling (LPID)*

Wu et al. proposed in [112] a labeling scheme for transitive closure computation on Directed Acyclic Graphs (DAGs). Computing a transitive closure in a graph is used for identifying all ancestors (or descendants) of a given node in the graph. We describe hereafter the Lite version of Prime Numbers Labeling Scheme for DAGs (PLSD) proposed by Wu et al.

Considering the entity IDs evolution as a DAG, we can use the PLSD labeling scheme described in [112] to locally detect if an ID has been deprecated. As in Figure 5.3a, if an ID A is deprecated by another ID B, then there exists a directed edge $A \to B$. As a consequence the *merge* and *split* operations mentioned in Section 5.2.3 are modeled as illustrated in Figure 5.3b and 5.3c respectively.

We annotate each node of the DAG such built with a pair of natural numbers: the self-label, and the ancestors-label. The self-label is a unique prime number (i.e., from all the self-labels, a self-label appears once and only once), and the ancestors-label of a vertex is the product of its self-label with the least common multiple of the ancestors-labels of its ancestors (i.e., its in-component). Formally this gives:



(a) A is *deprecated* by B. (b) A and B are *merged* into C. (c) A is *split* into B and C.

Figure 5.3: Lineage annotation with respect to the *deprecate*, *merge* and *split* operations.

Definition 17 *Let* $G = (I, D)$ *be a DAG with* I *a set of vertices representing IDs, and* D *a set of edges representing deprecation. We identify a vertex* $i \in I$ *by a pair* $(i_{self}, i_{ancestors})$ *with* $i_{self}, i_{ancestors} \in \mathbb{N}^+$*, where*

i) i_{self} *is a prime number such that there exists a bijection between* I *and the set* $I_{self} \equiv \{j_{self} | j \in I\}$

ii) $i_{ancestors} = i_{self} \cdot \mathrm{lcm}(a^1_{self}, \ldots, a^n_{self})$ *with* $a^1, \ldots, a^n$ *the ancestors (in-component) of* i

We call $i = (i_{self}, i_{ancestors})$ *the Lineage Preserving ID (LPID),* i_{self} *the self-label of* i*, and* $i_{ancestors}$ *its ancestors-label.* □

Lemma 2 shows that, given two LPIDs as defined in Definition 17, the fact that the self-label of one LPID divides the ancestors-label of the other LPID implies that the former is deprecated.

Lemma 2 *Let* $i = (i_{self}, i_{ancestors})$ *and* $j = (j_{self}, j_{ancestors})$ *be two LPIDs.* $j_{ancestors}/i_{self} \in \mathbb{N}$ *implies that* i *is deprecated.* □

We can now formalize in Algorithms 2, 3 and 4 the three operations defined in Section 5.2.3: `create`, `merge` and `split` respectively.

Algorithm 2 `create`

function CREATE(I) ▷ I the set of attributed LPIDs of the form $i = (i_{self}, i_{ancestors})$
  create the LPID $j = (l, l)$ with l the lowest prime such that $l \notin \{r | r = i_{self}, i \in I\}$
  add j to I
  return j

Definition 18 *Let* R *be the set of possible entity representations. A PLSD URI* $i_r = (p(r), c(r))$ *is defined to be a pair of two natural numbers* $p(r)$ *and* $c(r)$*.* $p : R \to \mathbb{N}$ *is a bijection such that* $p(r)$ *is a prime number* $\forall r \in R$*, and* $c(r)$ *is defined as*

$$c(r) = p(r) \cdot \begin{cases} 1 & \text{if } r = \mathtt{create}() \\ \mathrm{lcm}\left(c(m_1), \ldots, c(m_n)\right) & \text{if } r = \mathtt{merge}(m_1, \ldots, m_n) \\ c(s) & \text{if } r \in \mathtt{split}(s, n) \text{ with } n \in \mathbb{N} \end{cases}$$

Algorithm 3 merge

1: **function** MERGE(M) $\triangleright$ $M \subset I$, with I the set of attributed LPIDs of the form $i = (i_{self}, i_{ancestors})$
2: create the LPID $j = (l, l \cdot \mathrm{lcm}(m_1, \ldots, m_n))$ to the lowest prime l such that $l \notin \{r | r = i_{self}, i \in I\}$ with $\{m_1, \ldots, m_n\} \equiv M$
3: add j to I
4: **return** j

Algorithm 4 split

1: **function** SPLIT(j,n) $\triangleright$ $j \in I$, $n \geqslant 1 \in \mathbb{N}$, with I the set of attributed LPIDs of the form $i = (i_{self}, i_{ancestors})$
2: create empty set S
3: **for** $k = 1$ to n **do**
4: create the LPID $s_k = (l, l \cdot i_{ancestors})$ to the lowest prime l such that $l \notin \{r | r = i_{self}, i \in I\}$
5: add s_k to I
6: add s_k to S
7: **return** S

with create, merge *and* split *the possible operations resulting in a new entity representation.*

For using the proposed PLSD-Lite version presented in Section 5.4.1, we must make sure that the considered directed graph is actually acyclic. The acyclic nature of the graph naturally follows from Algorithm 3 line 2, and Algorithm 4 line 4 where incoming edges are added only to newly created vertices which did not belong to the set of vertices I before the respective operation is executed. This implies that, using the create, merge and split operations, a $\mathtt{deprecated}_{by}$ edge $d = (i, j)$ can only be created if $i \in I$ and $j \notin I$, i.e. $\forall i \in I$ no new incoming edge can ever be created. Thus, the graph $G = (I, D)$ is acyclic.

5.4.3 *List Labeling*

The LPID labeling uses an integer to encodes the ancestors' self-labels in an integer, product of those self-labels. This has the advantage to require only one division to find out whether an ID is an ancestor of another ID. On the other hand, it has the disadvantage to require the use of prime numbers as self-labels, which is costly to generate when the number of IDs is high. The obvious alternative is to use any natural number as a self-label, and store the ancestors simply in a list, separated by commas for example.

The difference with Definition 17, is that self-labels are ordinary natural numbers, and not restricted to prime numbers. And the definition of the ancestors-label is changed to a set of self-labels:

Definition 19 *Let* $G = (I, D)$ *be a DAG with* I *a set of vertices representing IDs, and* D *a set of edges representing deprecation. We label a vertex* $i \in I$ *by a pair* $(i_{self}, i_{ancestors})$ *with* $i_{self} \in \mathbb{N}^+$ *and where*

i) i_{self} *is such that there exists a bijection between* I *and the set* $I_{self} \equiv \{j_{self} | j \in I\}$

ii) $i_{ancestors} = \{a^1_{self}, \ldots, a^n_{self}\}$ *is set of the self-labels of* $a^1, \ldots, a^n$*, the ancestors (in-component) of* i

We call $i = (i_{self}, i_{ancestors})$ *the List Lineage Preserving ID Labeling (LPID),* i_{self} *the self-label of* i*, and* $i_{ancestors}$ *its ancestors-label.*

Since this labeling is equivalent to the LPID labeling, Lemma 2 can be slightly modified for list labeling:

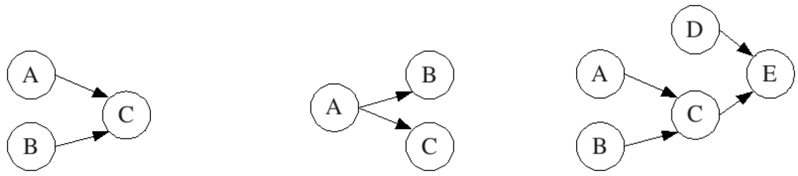
Lemma 3 *Let* $i = (i_{self}, i_{ancestors})$ *and* $j = (j_{self}, j_{ancestors})$ *be two List Labelings.* $i_{self} \in j_{ancestors}$ *implies that* i *is deprecated.*

The list labeling being equivalent to the LPID labeling in terms of selection of the identifiers to ask the ENS server for an update, the only difference between the two labelings is the storage space and and the selection performance.

5.4.4 *Merge Epochs Labeling*

The LPID and List Labelings allow to limit identifier update requests to some identifiers being in conflict. Once the identifiers to be updated have been selected, the client has no other choice to contact the ENS server to do so. This is because the client does not know whether there was a split or not between an identifier and its descendant. Consider Figure 5.4b. The client has in its repository only identifier A, and he receives identifier C. Since A has been split, the client has to ask the ENS server whether to replace A with B or with C. However, if the IDR graph is the one in Figure 5.4a and if the client knew that A has been merged with another identifier into C, it could replace A with C directly, without contacting the ENS server. In this section, we introduce an additional label, the *merge-label*, allowing the client to know whether a split occurred between any identifier and any of its descendant.

As with the ancestors-label, recording which of the ancestors of an identifier can be replaced by it without asking the ENS server can be done using the lcm of the prime number self-labels of the ancestors, or simply the list of the ancestors. Thus, the Definition 17 can be completed to include the merge-label as follows: the label for an identifier i is changed to $(i_{self}, i_{ancestors}, i_{merge})$, and the following definitions are added:



(a) A split is between A and C (b) A merge is between A and C (c) C has been deprecated

Figure 5.4: IDR graphs. In a the client can replace A with C without contacting the ENS server. In b, however, the client has to ask the ENS server to find out whether to replace A with B or C. And in c if the client uses the merge-label to replace A with C, it would use C which is deprecated. Whereas if the client contacted the ENS server, it would now use E which is the most up-to-date identifier for the entity considered.

iii) $i_{merge} = lcm(m^1_{self}, \dots, m^n_{self})$ with $m^k \in \{m^1, \dots, m^s\} \equiv M$, the ancestors of i that are such that all identifiers between i (excluded) and m^k (included) have been merged with other identifiers.

Definition 20 *Let $G = (I, D)$ be a DAG with I a set of vertices representing IDs, and D a set of edges representing deprecation. We identify a vertex $i \in I$ by a triple $(i_{self}, i_{ancestors}, i_{merge})$ with $i_{self}, i_{ancestors}, i_{merge} \in \mathbb{N}^+$, where*

i) i_{self} is a prime number such that there exists a bijection between I and the set $I_{self} \equiv \{j_{self} | j \in I\}$

ii) $i_{ancestors} = i_{self} \cdot lcm(a^1_{self}, \dots, a^n_{self})$ with $a^1, \dots, a^n$ the ancestors (in-component) of i

iii) $i_{merge} = lcm(m^1_{self}, \dots, m^n_{self})$ with $m^k \in \{m^1, \dots, m^s\} \equiv M$, the ancestors of i that are such that all identifiers between i (excluded) and m^k (included) have been merged with other identifiers.

We call $i = (i_{self}, i_{ancestors}, i_{merge})$ the List Lineage Preserving ID Merge Labeling (LPID Merge), i_{self} the self-label of i, $i_{ancestors}$ its ancestors-label, and i_{merge} its merge-label.

The List Labeling defined in Definition 19 can be similarly extended with a merge list:

Definition 21 *Let $G = (I, D)$ be a DAG with I a set of vertices representing IDs, and D a set of edges representing deprecation. We label a vertex $i \in I$ by a pair $(i_{self}, i_{ancestors}, i_{merge})$ with $i_{self} \in \mathbb{N}^+$ and where*

i) i_{self} is such that there exists a bijection between I and the set $I_{self} \equiv \{j_{self} | j \in I\}$

ii) $i_{ancestors} = a^1_{self}, \dots, a^n_{self}$ is set of $a^1, \dots, a^n$, the ancestors (in-component) of i

iii) $i_{merge} = \{m^1_{self}, \ldots, m^n_{self}\}$ *is set of self-labels of identifiers* $m^k \in \{m^1, \ldots, m^s\} \equiv M$*, the ancestors of* i *that are such that all identifiers between* i *(excluded) and* m^k *(included) have been merged with other identifiers.*

We call $i = (i_{self}, i_{ancestors}, i_{merge})$ *the List Merge Labeling,* i_{self} *the self-label of* i*,* $i_{ancestors}$ *its ancestors-label, and* i_{merge} *its merge-label.*

Lemma 2 and 3 still hold for LPID Merge and List Merge labelings, and in addition:

Lemma 4 *Let* $i = (i_{self}, i_{ancestors}, i_{merge})$ *and* $j = (j_{self}, j_{ancestors}, j_{merge})$ *be two LPID Merge Labelings.* $i_{self}/j_{merge}/in\mathbb{N}$ *implies that* i *can be replaced with* j *without risking to attribute an identifier to an entity which this identifier doesn't refer to.*

Lemma 5 *Let* $i = (i_{self}, i_{ancestors}, i_{merge})$ *and* $j = (j_{self}, j_{ancestors}, j_{merge})$ *be two List Merge Labelings.* $i_{self} \in j_{merge}$ *implies that* i *can be replaced with* j *without risking to attribute an identifier to an entity which this identifier doesn't refer to.*

5.4.4.1 *Drawbacks.*

As already mentioned, the merge labeling presents the advantage to allow the client, in some cases, to replace an identifier with its descendant without contacting the ENS server. This comes at the price of two things: 1) the additional merge-label require more space to store the labeling of an identifier, and 2) the client might use deprecated identifiers, where if it contacted the ENS server, it would use the most up-to-date identifier. See Figure 5.4c for an illustration of the second point.

5.4.5 *Baseline: No Labeling*

As far as we know, since the ENS is a relatively new concept, the problem considered in this chapter has not been approached before. For this reason, the baseline method which we use for comparing the above-presented techniques is the most straight forward one; that is, no labeling at all. In this case, when the client receives a description on an entity referred to by an identifier unknown to the client, the only way to make sure the new identifier does not conflict with another identifier in its repository is to simply ask the ENS server for each and every ID in its local repository.

This obviously would generate a lot of update requests to the ENS server, but it also has advantages over the labeling techniques we proposed. The first advantage is that a bigger part of the identifiers used by the client are up-to-date, since it more often asks for identifier updates. This implies that less identifiers in its repository refer to two or more entities in the world, and that less entities in its repository have more than one identifier. The second advantage is that since no labeling at all is used, the space requirement is less than when using lineage labeling.

5.5 EXPERIMENTS

We conducted experiments aiming at comparing the different proposed labeling approaches to the baseline where no labeling is used. We compared the different approaches with respect to two main characteristics: network traffic and quality of the identifiers.

The network traffic is measured in terms of number of update requests a client addresses to the ENS server, and the size of the labeling metadata. The quality of the identifiers answers the question: how unique are the identifiers? Ideally the identifiers are globally unique, i.e. for each entity there exists exactly one identifier, and each identifier identifies exactly one entity. To measure this we used at the client level: the number of deprecated identifiers, the repository size, the number of entities per identifier, and the number of identifiers per entity.

5.5.1 *Scenario*

In our experiments, we considered the situation where a client receives semantic annotations describing different kinds of entities. As the client encounters new identifiers, it first selects the identifiers in its repository which need update, according to the considered labeling scheme. The labeling scheme is one of the ones presented in Section 5.4: LPID, LPID Merge, List, List Merge, or the baseline without labeling. In each case it does its best to request update for all identifiers the scheme detects as deprecated. In the baseline case, since it does not have any information on which identifiers are outdated or not, it requests updates for all identifiers in its repository, plus the newly received identifiers. Once no more identifier is detected as outdated by the considered labeling scheme, it waits for a new arrival of identifiers, and repeats the process.

Before requesting for identifier updates, the merge labeling schemes perform the local replacements when applicable. Once the ENS receives an update request for identifier i, it returns one random[4] leaf identifier of the IDR graph which is a descendant of i. However, for each identifier i to update, the returned up-to-date identifier is the same independently of the labeling scheme evaluated; this is important for the results to be comparable. The sets of identifiers added

4 In practice the ENS performs a matching operation, but since this is not the focus of this chapter, we chose the identifier randomly, for the sake of simplicity.

to the client are the same and in the same order for all evaluated labeling schemes as well.

5.5.2 *Evaluation*

In the remainder of this section we will use the following abbreviations for referring to the labeling schemes:

LPID Lineage Preserving IDentifier labeling (see Definition 17, Section 5.4.2)

LPIDMerge Lineage preserving identifier labeling with Merge labeling (Definition 20, Section 5.4.4)

List List identifier lineage labeling (Definition 19, Section 5.4.3)

ListMerge List identifier lineage labeling with Merge Labeling (Definition 21, Section 5.4.4)

Classic Baseline without labeling (Section 5.4.5)

5.5.2.1 *Measuring Uniqueness.*

When identifiers are unique, it means there is a bijection between the set of identifiers and the set of entities. Informally this means there is exactly one identifier for each entity, and only one entity referred by each identifier. As we see unicity depends on the set of identifiers and the set of entities. We will talk about *local uniqueness* when considering the set of entities in a client's local repository and the set of identifiers related to those entities. And we will talk about *global uniqueness* when considering the set of entities in ENS server's repository and the set of identifiers related to those entities.

We will focus in this chapter on global uniqueness. The reason is that we started to consider the problem of how to transmit identity decision revisions from the ENS server to the client, taking the point of view of the server, that is, global. We will then assume that each identifier in the client's repository refers to exactly one entity in the same repository, and devise a measure for how globally unique the identifiers in the client's repository are. This means we want to know how many entities in the world an identifier used in the client's repository refers to; and how many distinct identifiers there is in the client's repository for one entities in that repository. Those are the measures we will use in those experiments to compare the impact on uniqueness quality of the evaluated labeling schemes. We call them *number of entities per ID* and *number of IDs per entity*.

5.5.2.2 *Measuring Network Traffic and ENS Server Workload.*

Identifier update requests makes use of two resources of concern in our scenario: 1) network bandwidth, and 2) ENS server workload.

Each update request from a client needs to be sent over the network, including the identifiers to update, and, if required, the description of the entities whose identifier is to be updated. For each identifier update request the ENS server receives, if there is more than one leaf descendant, the server has to perform an entity matching process to find out which is the present up-to-date identifier for the requested entity.

In this chapter we are not concerned with the details of the process the ENS server follows to return the up-to-date identifier. However we know that both network bandwidth consumption and ENS server workload are functions on the number of identifiers whose update is requested. This is what we will call in the remainder *number of update requests*. Note that, in practice, all identifiers needed to be updated might be sent to the server in one bulk update request. In this chapter we assume that one update request concerns one identifier only.

In addition, impacting on the network bandwidth consumption, the *size of the lineage labeling metadata* will be reported.

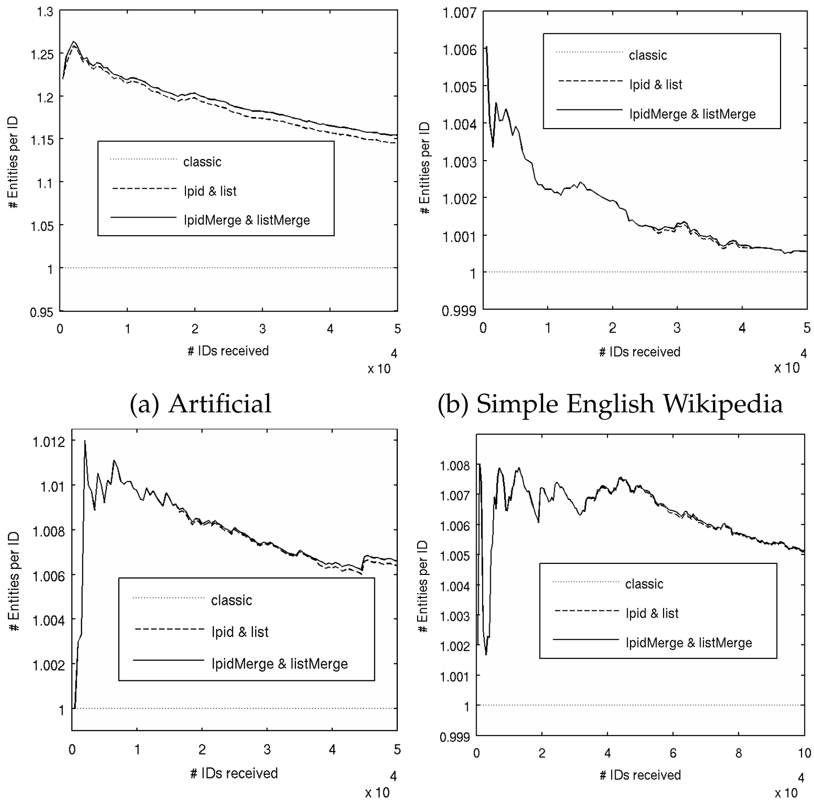
5.5.2.3 *Measuring Performance.*

As already mentioned, the reason why we introduced the identifier lineage labeling is to allow to reduce the number of request updates, while keeping a reasonable uniqueness quality. This is mainly a performance concern, and is therefore a central measure in this chapter.

We report two durations. The first is the time it takes the client, using a given labeling scheme, to select which identifiers it needs to ask the ENS server for update. We call this the *selection time*, which is null for the baseline, since all identifiers are always updated. To be able to compare performance between the baseline and the labeling schemes, we thus need to consider also the total time it takes a client to update it's identifiers. This comprises the selection time, plus the transmission time (of the requests), plus the possible entity matching time of the ENS server, plus the transmission time back. We approximated the whole to be the selection time, plus some time constant per identifier update request. We chose arbitrarily 200 ms for this time constant, which is not an uncommon round trip time for an intercontinental internet connection. It makes it a low constant for our case, considered that it also includes the time the server needs to process the request. We call it *total matching time*.

5.5.2.4 *Comparing the Labeling Schemes.*

Since most of the above proposed measures will yield different values for the different labeling schemes (and the baseline), we need some measure common to all of them to be able to compare them. For this we chose the the number of new identifiers added to the client.



(a) Artificial (b) Simple English Wikipedia

(c) NL Wikipedia 50,000 (d) NL Wikipedia 100,000

Figure 5.5: Number of Entities per Identifier for each of the Datasets

5.5.3 *Datasets*

We experimented with 3 different IDR graphs. The two first based on the revisions history of the Dutch and the Simple English Wikipedia datasets, and the third one artificially generated. As introduced in Section 5.2.3, an IDR graph is defined by a sequence of three events: identifier creations, merges and splits.

Here is how we interpreted the Wikipedia revision history in terms of those three actions. When a Wikipedia page is created, it obviously corresponds to an identifier creation. The first revision of a Wikipedia page where a redirect appears is considered as a merge between the two identifiers corresponding to the Wikipedia page and its redirect target page. And the first revision of a page where a disambiguation tag appears is interpreted as a split for this page's identifier to the ones corresponding to its targets.

The artificial IDR graph was generated with the following probability distributions. At each iteration, we first choose what action to perform[5]: creation ($p = 0.77$), split ($p = 0.08$) and merge ($p = 0.15$). If split or merge was picked, the identifier(s) to which the action applies are picked at random (uniform distribution); and the number of iden-

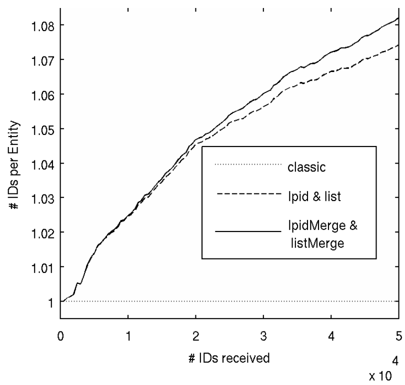
5 Probabilities are estimated using statistics from Wikipedia articles operations.

tifiers into which the identifier is split, or the number of identifiers to merge follow a Zipf distribution with $\sigma = 2$, a maximum of 5, and a minimum of 2.

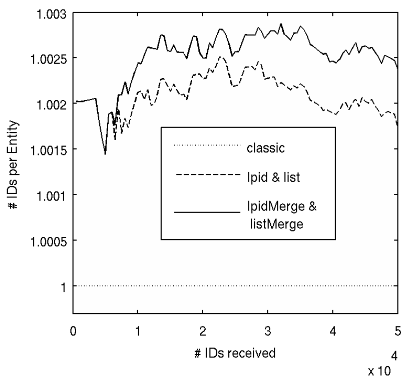
5.5.4 *Experiment Results*

We present in this section the results of our experiments. When significant difference is mentioned, this has been confirmed with a one-way ANOVA[6] with 5% of significance. For most of the graphs we present the results for each of for IDR graphs: artificially generated, simple English Wikipedia, and Dutch Wikipedia. For each we added to the client 100 times 500 (= 50,000) identifiers, and since the Dutch Wikipedia is the only having enough entities to allow it, we also tried 200 times 500 (= 100,000) identifiers for this IDR graph. Which gives us 4 different simulation for each scheme, which we note 'artificial', 'simplewiki', 'nlwiki100' and 'nlwiki200' respectively.

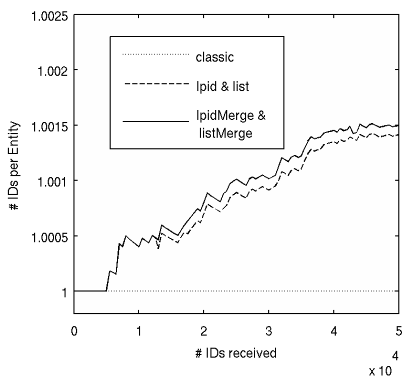
5.5.4.1 *Identifiers Uniqueness.*



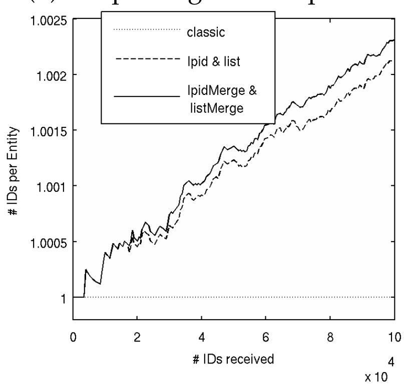
(a) Artificial



(b) Simple English Wikipedia



(c) NL Wikipedia 50,000

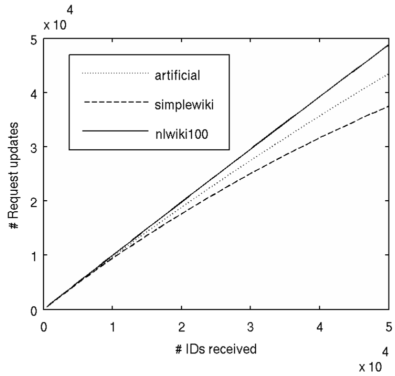


(d) NL Wikipedia 100,000

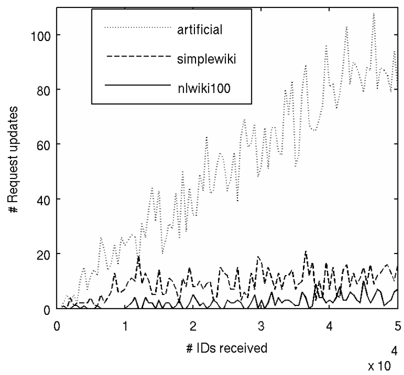
Figure 5.6: Number of Identifiers per Entity

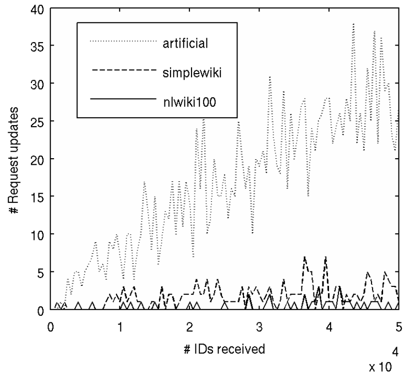
6 Roughly, one-way ANalysis Of VAriance (ANOVA) is similar to the T-Test, but for more than two series of observations. See [85] for more details.

We show in Figure 5.5 the average number of entities per identifier at different steps (number of identifiers received by the client) in the simulation for each of the datasets. The value for the classic scheme is one, as expected. For all datasets, the difference between using or not the merge-label happens to be non significant; which is a good sign since this means that the merge labels allow to decrease the number of update requests while keeping a uniqueness quality similar as if no merge labels were used. The difference between the classic and the labeling schemes is visibly significant.



(a) Classic





(b) LPID & List (c) LPIDMerge & ListMerge

Figure 5.7: Number of ID update requests per iteration of 500 IDs added to the client.

In Figure 5.6 we present the average number of identifiers per entity for each iteration step. Whereas for the classic scheme we have the expected values of one, it is significantly different from the others, but merge-labeling makes a significant difference only for the IDR graphs of Simple English and Dutch Wikipedia on 100,000 identifiers.

We note that the number of entities per identifier tends to decrease over time, which is a good sign in the sense that it converges to the ideal case of one entity existing on the ENS server per identifier in the client's repository. However the number of identifiers per entity tends to augment over time, and this more rapidly while using merge-labels as not using them. This is problematic, even though it is easier to detect duplicate entities in the client's repository than reverse an accidental merge of two distinct entities because they erroneously shared a same identifier. Note however that, even if the differences are significant, the values are still quite low, which indicates that the problem would occur rarely.

5.5.4.2 *Network Traffic and ENS Server Workload.*

We present in Figure 5.7 the number of identifiers the client is requesting the ENS server for update for all the labeling schemes. Note that the number of update requests are the same for the LPID and List schemes, both with Merge epochs or not. Because the values for the baseline are so huge compared to the labeling schemes, we present the plots per scheme instead of per dataset as for the other measures. In addition to the obvious significant explosion of updates of the classic scheme compared to using lineage labeling—approximately 50 to 500 more than with lineage labeling—the one-way ANOVA revealed that using merge-labels or not does not change significantly the number of update requests, even though the merge labels allow to decrease the number of update requests by a factor 2 approximately.

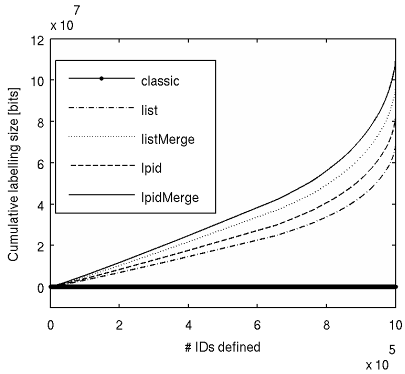


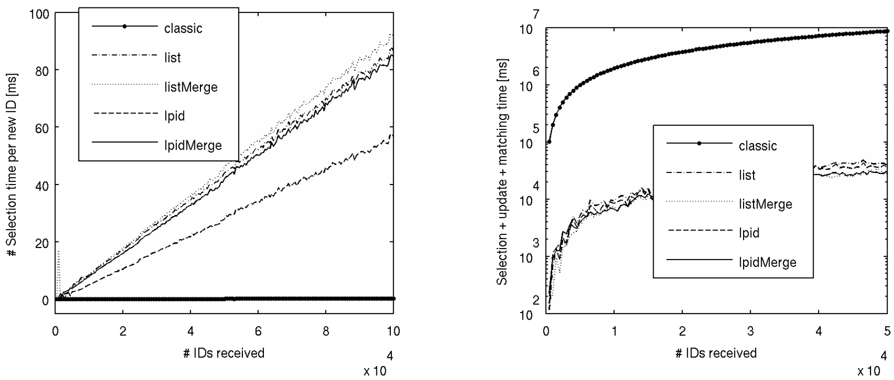
Figure 5.8: Size of the labeling as the number of IDs grows on the Artificial IDR graph.

Figure 5.8 present the cumulative size of the lineage labeling for all the identifier schemes on the artificial IDR graph. For example, the total size needed to store the lineage labeling for one million IDs using the LPIDMerge scheme is 13 MB (109,124,363 bits). Since the baseline is not using any labeling, its size is zero for all datasets. Additionally, we notice that the list labeling scheme uses less space than the LPID one, and naturally, adding merge-labels augments the size of the labeling. All differences are statistically significant, and similar for other IDR graphs.

More interestingly, the size of the lineage labeling schemes augments almost exponentially. In our case the biggest labeling is of 4,500 bits for one identifier in the artificial IDR graph, which has a bit more than one million identifiers. This is still reasonable, but this size always increases over time, and this will be even more pronounced as new identifiers will be created and new split and merge operations will occur.

5.5.4.3 *Performance.*

Figure 5.9a shows the selection time. The main observation we can make regarding those figures, is that LPID selects the identifiers to be updated significantly faster than LPIDMerge, and the two List labelings. The classic is trivially different from the others since it is null, and all other labeling perform similarly. This analysis is confirmed by one-way ANOVA.



(a) Selection time per number of ID added to the client's repository on the Dutch Wikipedia IDR graph and 100,000 identifiers added to the client.

(b) Total matching time per number of identifier update request on the artificial IDR graph.

Figure 5.9: Performance measurements

In Figure 5.9b, we see the total matching times, i.e. the total time needed to update a clients repository, in average, for one identifier to update. It shows clearly that the classic scheme is much slower. This is mainly due to the very high number of identifier update requests it generates. Also, there is not a significant difference between the other schemes. This is confirmed by a one-way ANOVA.

5.6 Related work

Quite some work have been done in the area of data provenance [101, 34] and lineage retrieval [14]. However, today new challenges are present: with the growing size of the Semantic Web and with the spreading of semantic-aware user applications managing identity, the Web has to face the problem of scalability and efficiency. In this chapter we analyze how to optimize identity management on the Web. [101], for example, cites the following applications of data lineage:

DATA QUALITY: Lineage can be used to estimate data quality and data reliability based on the source data and transformations [70]. It can also provide proof statements on data derivation [35].

AUDIT TRAIL: Provenance can be used to trace the audit trail of data [83], determine resource usage [64], and detect errors in data generation [52].

REPLICATION RECIPES: Detailed provenance information can allow repetition of data derivation [in scientific research], help maintain its currency [83], and be a recipe for replication [49].

ATTRIBUTION: Pedigree can establish the copyright and ownership of data, enable its citation [70], and determine liability in case of erroneous data.

Our work would fit best in the Data Quality category, however, both the application and techniques used are quite different from our approach. [35] defines a notion of information provenance called "knowledge provenance" and proposes a system to provide to the end-users information about how a question answering system arrived at its answers. [70], a SIGMOD workshop report, outlines the importance of data provenance in life science research allow the update of derived information when the base information changes, but does not proposes concrete solution. [34] formally defines and propose algorithms for the data lineage problem in general data warehouses: "tracing warehouse data items back to the original source items from which they were derived." Again, this is quite different from our approach: it proposes a general data lineage solution, however, we are addressing a particular problem that we solve with a specific partial data lineage information.

[105] presents "a fully decentralized model of *collaborative* data sharing, in which participants publish their data on an ad hoc basis and simultaneously *reconcile* updates with those published by others." Each participant thus maintain their own state of the information. To allow participants to perform this reconciliation, all transactions modifying an entity are communicated to each participants, thus also communicating lineage. However, identifiers are assumed to be stable, so that, unlike the scenario considered in this chapter, the identifiers are never merged nor split. The object and usage of the lineage is therefore significantly different than the one we are considering in our work.

5.7 CONCLUSIONS AND FURTHER WORK

We proposed to use identifier lineage labeling, and experiments confirmed that they allow to reduce the number of identifier update requests to the ENS server, while keeping an acceptable quality of uniqueness at the level of client's local repository. We also proposed merge-labels to further reduce the number of update requests by allowing the clients to replace an identifier by one of its descendants when this does not decrease the local uniqueness quality of the identifiers on the client's repository, which was also confirmed by the experiments. In the future we need to address the exponential growth of the identifier labeling. To reduce the size of the labeling, we intend to investigate the use bloom filters [13] to summarize list of IDs. As well, avoiding to duplicate an ID in the merge-label and in the ancestors-label should also allow to reduce the size of merge-labeling, without solving the problem of exponential growth. The use of time-to-live (TTLs) associated to IDs indicating to the client when to request an update seems to be a promising approach as well.

6 CONCLUSIONS & FUTURE WORKS

In this dissertation we presented three works addressing challenges presented by highly heterogeneous semi-structured web data. We presented SQuaScheD, a hierarchical schema discovery technique based on the information theoretic Minimum Description Length principle and spectral graph clustering. We presented our DSCD algorithm which leverages the inter-dependency between duplicate entity descriptions and duplicate attributes in order to facilitate query formulation and processing in such heterogeneous environments. And finally, we presented the Lineage Preserving Identifier (LPID) labeling scheme which helps to detect and resolve conflicts due to evolving entities, allowing pay-as-you-go information integration decisions to be propagated and identifiers to be updated only if necessary.

A natural continuation of this work includes studying:

- How to disambiguate attributes and values, and therefore also entities. Deduplication, also known as record linkage or data matching, is a well-known and studied problem in information integration. The problem of *information disambiguation*, however, is more recent and constitute the dual problem of deduplication. It occurs naturally and frequently in information produced by and for humans, due to their capacity at considering context while interpreting information representations; and here lies, we believe, the key to information disambiguation.

- How to leverage *values* of the attributes for schema discovery information disambiguation. The value of an attribute being part of the representation of an object's property as introduced in Definition 3 of Chapter 2, it is natural that it is a valuable feature for information integration and disambiguation, as well as for schema discovery.

- How to leverage information deduplication and disambiguation to improve schema discovery and vice versa. Knowing which instances of an attribute represent properties of which kind of objects (disambiguation), and which instances of different attributes represent similar properties of similar objects (deduplication) is obviously valuable for better schema discovery. Conversely, knowing what kinds of objects are described with the instances of a given attribute is useful for discerning ambiguous meanings of the attributes (disambiguation), or spotting different attributes with similar meaning (deduplication). We believe this inter-dependency can be leveraged to profit to the solution of both problems.

In addition to the three core publications related in detail in this dissertation, I also contributed significantly to the two following related publications, among others:

[82] Zoltán Miklós, Nicolas Bonvin, Paolo Bouquet, Michele Catasta, Daniele Cordioli, Peter Fankhauser, Julien Gaugaz, Ekaterini Ioannou, Hristo Koshutanski, Antonio Maña, Claudia Niederée, Themis Palpanas, and Heiko Stoermer. From Web Data to Entities and Back. *CAiSE*, pages 302–316, June 2010. URL http://dl.acm.org/citation.cfm?id=1883784.1883817

[37] Gianluca Demartini, Julien Gaugaz, and Wolfgang Nejdl. A Vector Space Model for Ranking Entities and Its Application to Expert Search. *ECIR*, 5478:189–201, 2009. doi: 10.1007/978-3-642-00958-7. URL http://www.springerlink.com/content/x2w826v168677434/

[82] describes the system developed in the OKKAM IP European project (FP7-ICT-2007-215032) mentioned in Chapter 5. I significantly contributed in this project by the design and implementation of the *Generic Matching Module* described in Section 3.4 of the publication, representing one important software contribution of my institute to the project. The *Generic Matching Module* is the precision-optimized entity search component of the Entity Name System (ENS) developed in the OKKAM project and also mentioned in Chapter 5.

I also contributed in [37] which models experts or other entities as linear combinations of the textual documents relating those entities, such that the entities are in the same term vector space as the documents, allowing to naturally include various features in the model, such as authoritativeness or time-related decay of the documents and/or the entities.

Beside those contributions to research articles, I also contributed significantly in the project proposal for the LivingKnowledge IP European project (FP7-ICT-2007-3-231126) about "Facts, Opinions and Bias in Time", and the proposal for Prof. Nejdl's ERC project Alexandria (ERC-339233) about not only storing Web Archives, but also indexing, retrieving and exploring them efficiently and meaningfully.

I also had the pleasure to assist Prof. Nejdl in the teaching of three courses: Artificial Intelligence I and II, and later also Web Science. In the Artificial Intelligence I course I taught the basics of Prolog programming as part of the practical sessions of the course, as well as supported the students for the preparation of the examinations, and assisted Prof. Nejdl in the redaction and correction of the latter. In "Web Science", an advanced course for Master students, I assisted Prof. Nejdl in the organization of the introductory lectures to the research topics studied in the course, and organized and supported the students in the preparation of the presentation of state-of-the-art papers in one of the offered research topics.

A
Scientific and Academic Curriculum Vitae

2013 to present Data Scientist, XING AG, Hamburg

2006 to 2012 PhD Student, L3S Research Center, Fakultät für Elektrotechnik und Informatik, Leibniz Universiät Hannover

2000 to 2005 ing. sys. com. dipl. EPF (equivalent to Dipl.-Ing. in Communication Systems in Germany), École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Aug 2002 to Sep 2003 Erasmus exchange student, Kungliga Tekniska Högskola (KTH), Stockholm, Sweden

1998-2000 3 semesters of Sociology, Université de Genève, Switzerland

BIBLIOGRAPHY

[1] SPARQL 1.1 Query Language. Technical report, W3C, 2013. URL http://www.w3.org/TR/sparql11-query.

[2] Marco D. Adelfio and Hanan Samet. Schema extraction for tabular data on the web. *PVLDB*, 6(6):421–432, 2013. URL http://www.vldb.org/pvldb/vol6/p421-adelfio.pdf.

[3] Charu C. Aggarwal and Chandan K. Reddy, editors. *Data Clustering: Algorithms and Applications*. CRC Press, 2014. ISBN 978-1-46-655821-2. URL http://www.charuaggarwal.net/clusterbook.pdf.

[4] Sihem Amer-Yahia, SungRan Cho, and Divesh Srivastava. Tree Pattern Relaxation. In Christian Jensen, Simonas Šaltenis, Keith Jeffery, Jaroslav Pokorny, Elisa Bertino, Klemens Böhn, and Matthias Jarke, editors, *EDBT*, volume 2287 of *Lecture Notes in Computer Science*, pages 89–102. Springer Berlin / Heidelberg, 2002. ISBN 978-3-540-43324-8. URL http://dx.doi.org/10.1007/3-540-45876-X_32.

[5] Sihem Amer-Yahia, Laks V. S. Lakshmanan, and Shashank Pandit. FleXPath. In *SIGMOD*, page 83, New York, New York, USA, June 2004. ACM Press. ISBN 1581138598. doi: 10.1145/1007568.1007581. URL http://dl.acm.org/citation.cfm?id=1007568.1007581.

[6] Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroki Arimura, Hiroshi Sakamoto, and Setsuo Arikawa. Efficient Substructure Discovery from Large Semi-structured Data. In Robert L Grossman, Jiawei Han, Vipin Kumar, Heikki Mannila, and Rajeev Motwani, editors, *SDM*. SIAM, 2002. ISBN 0-89871-517-2. URL http://www.appliedmathematician.net/proceedings/datamining/2002/dm02-10AsaiT.pdf.

[7] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-76297-3.

[8] Wolf-Tilo Balke and Matthias Wagner. Through Different Eyes. In *WWW Alt.*, page 196, New York, New York, USA, May 2004. ACM Press. ISBN 1581139128. doi: 10.1145/1013367.1013400. URL http://dl.acm.org/citation.cfm?id=1013367.1013400.

[9] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open Information Extraction from the Web. In Manuela M Veloso, editor, *IJCAI*, pages 2670–2676, January 2007. URL http://ml.cs.washington.edu/www/media/papers/tmpYZBSTp.pdf.

[10] Andrew R Barron, Jorma Rissanen, and Bin Yu. The Minimum Description Length Principle in Coding and Modeling. *IEEE Trans. Inf. Theory*, 44(6):2743–2760, 1998. URL http://dblp.uni-trier.de/rec/bibtex/journals/tit/BarronRY98.

[11] Klaus Berberich, Srikanta J Bedathur, Thomas Neumann, and Gerhard Weikum. A time machine for text search. In *SIGIR*, pages 519–526, 2007.

[12] A. Bilke and F. Naumann. Schema Matching Using Duplicates. In *ICDE*, pages 69–80. IEEE, 2005. ISBN 0-7695-2285-8. doi: 10.1109/ICDE.2005.126. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1410107.

[13] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13:422–426, 1970. URL http://dx.doi.org/10.1145/362686.362692.

[14] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, 2005.

[15] Paolo Bouquet, Heiko Stoermer, and Barbara Bazzanella. An Entity Name System (ENS) for the Semantic Web. In *Semant. Web Res. Appl. Proc. ESWC2008.*, volume Volume 502 of *Lecture Notes in Computer Science*, pages 258–272. Springer Berlin / Heidelberg, June 2008.

[16] Paolo Bouquet, Heiko Stoermer, Claudia Niederée, Antonio Ma\ Na, Claudia Niederee, and Antonio Mana. Entity Name System: The Backbone of an Open and Scalable Web of Data. In *ICSC*, number CSS-ICSC 2008-4-28-25, pages 554–561. IEEE Computer Society, August 2008. doi: DOI10.1109/ICSC.2008.37. URL http://doi.ieeecomputersociety.org/10.1109/ICSC.2008.37http://dx.doi.org/10.1109/ICSC.2008.37.

[17] Markus M Breunig, Hans P Kriegel, Peer Kröger, and Jörg Sander. Data bubbles: quality preserving performance boosting for hierarchical clustering. In *SIGMOD '01 Proc. 2001 ACM SIGMOD Int. Conf. Manag. data*, pages 79–90, New York, NY, USA, 2001. ACM. ISBN 1-58113-332-4. doi: http://dx.doi.org/10.1145/375663.375672. URL http://dx.doi.org/10.1145/375663.375672.

[18] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Computer Networks and ISDN Systems*, pages 107–117. Elsevier Science Publishers B. V., 1998.

[19] A. Z. Broder and A. C. Ciccolo. Towards the Next Generation of Enterprise Search Technology. *IBM Syst. J.*, 43(3):451–454, 2004. ISSN 0018-8670. doi: 10.1147/sj.433.0451. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5386744.

[20] Ingo Brunkhorst, Paul-Alexandru Alexandru Chirita, Stefania Costache, Julien Gaugaz, Ekaterini Ioannou, Tereza Iofciu, Enrico Minack, Wolfgang Nejdl, and Raluca Paiu. The Beagle++ Toolbox: Towards an Extendable Desktop Search Architecture. *Semant. Deskt. Work. ISWC*, November 2006. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.125.2754.

[21] Michael J Cafarella, Dan Suciu, and Oren Etzioni. Navigating Extracted Data with Schema Discovery. In *WebDB*, 2007. URL http://web.eecs.umich.edu/~michjc/papers/cafarella_webdb07.pdf.

[22] Michael J Cafarella, Alon Y Halevy, Daisy Z Wang, Eugene Wu, and Yang Zhang. WebTables: Exploring the power of tables on the web. *VLDB*, 1(1):538–549, 2008. URL http://www.vldb.org/pvldb/1/1453916.pdf.

[23] Michael J Cafarella, Alon Y Halevy, and Nodira Khoussainova. Data Integration for the Relational Web. *PVLDB*, 2(1):1090–1101, 2009. URL http://www.vldb.org/pvldb/2/vldb09-576.pdf.

[24] MJ Cafarella, AY Halevy, and Y Zhang. Uncovering the relational web. In *WebDB*, 2008. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.5666&rep=rep1&type=pdf.

[25] David Carmel, Yoelle S. Maarek, Matan Mandelbrod, Yosi Mass, and Aya Soffer. Searching XML Documents via XML Fragments. In *SIGIR*, pages 151–158, New York, New York, USA, July 2003. ACM Press. ISBN 1581136463. doi: 10.1145/860435.860464. URL http://dl.acm.org/citation.cfm?id=860435.860464.

[26] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/320434.320440. URL http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=0C573FDE4B5BEB3BAD612436DBF66C01?doi=10.1.1.123.1085&rep=rep1&type=pdf.

[27] Sergey Chernov, Gianluca Demartini, and Julien Gaugaz. L3S Research Center at TREC 2006 Enterprise Track. *Text Retr. Conf. Proc.*, 2006.

[28] Paul-Alexandru Chirita, Select Stefania Costache, Julien Gaugaz, and Wolfgang Nejdl. Desktop Context Detection Using Implicit Feedback. *Work. Pers. Inf. Manag. SIGIR*, 2006.

[29] C. Chow and C. Liu. Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Trans. Inf. Theory*, 14(3):462–467, May 1968. ISSN 0018-9448. doi: 10.1109/TIT.1968.1054142. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1054142.

[30] Wesley W. Chu, Hua Yang, Kuorong Chiang, Michael Minock, Gladys Chow, and Chris Larson. CoBase: A Scalable and Extensible Cooperative Information System. *J. Intell. Inf. Syst.*, 6(2-3):223–259, June 1996. ISSN 0925-9902. doi: 10.1007/BF00122129. URL http://www.springerlink.com/content/hq6362080t854235/.

[31] E. F. Codd. A relational model of data for large shared data banks. *CACM*, 13(6):377–387, 1970.

[32] Stefania Costache, Julien Gaugaz, Ekaterini Ioannou, Claudia Niederée, and Wolfgang Nejdl. Detecting Contexts on the Desktop Using Bayesian Networks. *Deskt. Search Work. SIGIR2010*, 2010.

[33] Nick Craswell, Gianluca Demartini, Julien Gaugaz, and Tereza Iofciu. L3S at INEX 2008: Retrieving Entities Using Structured Information. In Shlomo Geva, Jaap Kamps, and Andrew Trotman, editors, *INEX*, volume 5631 of *Lecture Notes in Computer Science*, pages 253–263. Springer, 2008. ISBN 978-3-642-03760-3.

[34] Yingwei Cui and Jennifer Widom. Lineage tracing for general data warehouse transformations. *VLDB J.*, 12(1):41–58, 2003. URL http://dblp.uni-trier.de/db/journals/vldb/vldb12.html#CuiW03.

[35] Paulo Pinheiro da Silva, Deborah L. McGuinness, and Rob McCool. Knowledge provenance infrastructure. *IEEE Data Eng. Bull.*, 26(4):26–32, 2003. URL http://sites.computer.org/debull/A03dec/paulo.ps.

[36] Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. Mining Satabase Structure; or, How to Build a Data Quality Browser. In *SIGMOD*, page 240, New York, New York, USA, June 2002. ACM Press. ISBN 1581134975. doi: 10.1145/564691.564719. URL http://dl.acm.org/citation.cfm?id=564691.564719.

[37] Gianluca Demartini, Julien Gaugaz, and Wolfgang Nejdl. A Vector Space Model for Ranking Entities and Its Application to Expert Search. *ECIR*, 5478:189–201, 2009. doi: 10.1007/978-3-642-00958-7. URL http://www.springerlink.com/content/x2w826v168677434/.

[38] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling Schemas of Disparate Data Sources. *SIGMOD Rec.*, 30(2): 509–520, June 2001. ISSN 01635808. doi: 10.1145/376284.375731. URL http://dl.acm.org/citation.cfm?id=376284.375731.

[39] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Y Halevy. Learning to Map between Ontologies on the Semantic Web. In *WWW Conf.*, page 662, New York, New York, USA, May 2002. ACM Press. ISBN 1581134495. doi: 10.1145/511446.511532. URL http://dl.acm.org/citation.cfm?id=511446.511532.

[40] Xin Dong and Alon Y Halevy. Malleable Schemas: A Preliminary Report. In *WebDB*, pages 139–144, Baltimore, Maryland USA, 2005. URL http://www2.research.att.com/~lunadong/publication/malleable_webdb.pdfhttp://webdb2005.uhasselt.be/webdb05_eproceedings.pdf.

[41] Ahmed K Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S Verykios. Duplicate Record Detection: A Survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, January 2007. ISSN 1041-4347. doi: 10.1109/TKDE.2007.9. URL http://doi.ieeecomputersociety.org/10.1109/TKDE.2007.9.

[42] Hazem Elmeleegy, Jayant Madhavan, and Alon Y. Halevy. Harvesting relational tables from lists on the web. *PVLDB*, 2(1): 1078–1089, 2009. URL http://www.vldb.org/pvldb/2/vldb09-325.pdf.

[43] Hazem Elmeleegy, Jayant Madhavan, and Alon Y. Halevy. Harvesting relational tables from lists on the web. *VLDB J.*, 20(2): 209–226, 2011. URL http://dx.doi.org/10.1007/s00778-011-0223-0.

[44] David W. Embley, Matthew Hurst, Daniel P. Lopresti, and George Nagy. Table-processing paradigms: a research survey. *IJDAR*, 8(2-3):66–86, 2006. URL http://www.ecse.rpi.edu/~nagy/PDF_chrono/2006_Embley_Hurst_Lopresti_GN_IJDAR06_Table_Survey.pdf.

[45] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam. Open information extraction: The

second generation. In Toby Walsh, editor, *IJCAI*, pages 3–10. IJCAI/AAAI, 2011. ISBN 978-1-57735-516-8. URL http://ijcai.org/papers11/Papers/IJCAI11-012.pdf.

[46] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *EMNLP*, pages 1535–1545. ACL, 2011. ISBN 978-1-937284-11-4. URL http://aclweb.org/anthology/D/D11/D11-1142.pdf.

[47] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Mach. Learn.*, 2(2):139–172, September 1987. ISSN 0885-6125. doi: 10.1007/BF00114265. URL http://www.springerlink.com/content/x8552ppn35245112/.

[48] W Tecumseh Fitch. *The evolution of language*. Cambridge University Press, 2010.

[49] Ian T. Foster, Jens-S. Vöckler, Michael Wilde, and Yong Zhao. The virtual data grid: A new model and architecture for data-intensive collaboration. In *CIDR*, 2003. URL http://www-db.cs.wisc.edu/cidr/cidr2003/program/p18.pdf.

[50] Dayne Freitag and Andrew McCallum. Information extraction with HMM structures learned by stochastic optimization. In Henry A. Kautz and Bruce W. Porter, editors, *AAAI/IAAI*, pages 584–589. AAAI Press / The MIT Press, 2000. ISBN 0-262-51112-6. URL http://www.aaai.org/Papers/AAAI/2000/AAAI00-089.pdf.

[51] Norbert Fuhr and Kai Gross. XIRQL: a Query Language for Information Retrieval in XML Documents. In *SIGIR*, pages 172–180, New York, New York, USA, September 2001. ACM Press. ISBN 1581133316. doi: 10.1145/383952.383985. URL http://dl.acm.org/citation.cfm?id=383952.383985.

[52] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Improving data cleaning quality using a data lineage facility. In Dimitri Theodoratos, Joachim Hammer, Manfred A. Jeusfeld, and Martin Staudt, editors, *Proceedings of the 3rd Intl. Workshop on Design and Management of Data Warehouses, DMDW'2001, Interlaken, Switzerland, June 4, 2001*, volume 39 of *CEUR Workshop Proceedings*, page 3. CEUR-WS.org, 2001. URL http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-39/paper3.pdf.

[53] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S Seshadri, and Kyuseok Shim. XTRACT: a system for extracting document type descriptors from XML documents. In *SIGMOD '00 Proc. 2000 ACM SIGMOD Int. Conf. Manag. data*, volume 29, pages 165–176, New York, NY, USA, June 2000. ACM Press.

doi: http://dx.doi.org/10.1145/342009.335409. URL http://dx.doi.org/10.1145/342009.335409.

[54] Minos N Garofalakis, Aristides Gionis, Rajeev Rastogi, S Seshadri, and Kyuseok Shim. DTD Inference from XML Documents: The XTRACT Approach. *IEEE Data Eng. Bull.*, 26 (3):19–25, 2003. URL http://dblp.uni-trier.de/rec/bibtex/journals/debu/GarofalakisGRSS03.

[55] Julien Gaugaz and Gianluca Demartini. Entity identifiers for lineage preservation. In Paolo Bouquet, Harry Halpin, Heiko Stoermer, and Giovanni Tummarello, editors, *Proceedings of the 1st IRSW2008 International Workshop on Identity and Reference on the Semantic Web, Tenerife, Spain, June 2, 2008*, volume 422 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. URL http://ceur-ws.org/Vol-422/irsw2008-submission-7.pdf.

[56] Julien Gaugaz, Stefania Costache, Paul-Alexandru Chirita, Claudiu Firan, and Wolfgang Nejdl. Activity Based Links as a Ranking Factor in Semantic Desktop Search. *Lat. Am. Web Conf.*, pages 49–57, October 2008. doi: 10.1109/LA-WEB.2008.7. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4756161.

[57] Julien Gaugaz, Jakub Zakrzewski, Gianluca Demartini, and Wolfgang Nejdl. How to Trace and Revise Identities. *ESWC*, 5554:414–428, May 2009. doi: 10.1007/978-3-642-02121-3. URL http://dl.acm.org/citation.cfm?id=1561533.1561574.

[58] Julien Gaugaz, Patrick Siehndel, Gianluca Demartini, Tereza Iofciu, Mihai Georgescu, and Nicola Henze. Predicting the Future Impact of News Events. In Ricardo Baeza-Yates, Arjen de Vries, Hugo Zaragoza, B. Barla Cambazoglu, Vanessa Murdock, Ronny Lempel, and Fabrizio Silvestri, editors, *ECIR*, 2012.

[59] Julien Gaugaz, Xuan Zhou, Qingzhong Meng, Claudia Niederée, and Wolfgang Nejdl. Here is the Data. Where is its Schema? SQuaScheD: Unsupervised Schema Discovery for Heterogeneous Data. Technical report, 2015.

[60] Mihai Georgescu, Dang Duc Pham, Claudiu Firan, Julien Gaugaz, and Wolfgang Nejdl. Map to Humans and Reduce Error - Crowdsourcing for Deduplication Applied to Digital Libraries. *CIKM*, 2012.

[61] Parke Godfrey. Minimization in Cooperative Response to Failing Database Queries. *Int. J. Coop. Inf. Syst.*, 6(2):95 – 149, 1997. URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.2305.

[62] Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In Matthias Jarke, Michael J Carey, Klaus R Dittrich, Frederick H Lochovsky, Pericles Loucopoulos, and Manfred A Jeusfeld, editors, *VLDB'97, Proc. 23rd Int. Conf. Very Large Data Bases*, pages 436–445, 1997. URL ftp://db.stanford.edu/www/lore/pubs/dataguide.pdf.

[63] Roy Goldman and Jennifer Widom. Approximate Dataguides. In *Proc. Work. Query Process. Semistructured Data Non-Standard Data Formats*, volume 97, pages 436–445, 1999. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.6111.

[64] Mark Greenwood, CA Goble, Robert D Stevens, Jun Zhao, Matthew Addis, Darren Marvin, Luc Moreau, and Tom Oinn. Provenance of e-science experiments-experience from bioinformatics. In *Proceedings of UK e-Science All Hands Meeting 2003*, pages 223–226, 2003. URL https://scholar.google.de/scholar.bib?q=info:16GsQYq9YQwJ:scholar.google.com/&output=citation&scisig=AAGBfm0AAAAAVNDU5_hSgkCus1wEE6saF_Q3_F3ICxAU&scisf=4&hl=en.

[65] Peter Grünwald. A Tutorial Introduction to the Minimum Description Length Principle. *Adv. Minim. Descr. Length Theory Appl.*, math.ST/04, 2005. URL http://www.cwi.nl/~pdg/ftp/mdlintro.pdf.

[66] Alon Y Halevy and Xin (Luna) Dong. A Platform for Personal Information Management and Integration. In *CIDR*, volume pages, pages 119–130, 2005. URL http://www.cidrdb.org/cidr2005/papers/P10.pdf.

[67] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explor. Newsl.*, 11(1): 10, November 2009. ISSN 19310145. URL http://dl.acm.org/citation.cfm?id=1656274.1656278.

[68] Eran Halperin and Richard M Karp. The minimum-entropy set cover problem. *Theor. Comput. Sci.*, 348(2-3):240–250, 2005. ISSN 0304-3975. doi: DOI:10.1016/j.tcs.2005.09.015. URL http://www.sciencedirect.com/science/article/B6V1G-4H6PP22-2/2/3848090f2d45da7a1bb4ec8b4e8339f5.

[69] Bin He and Kevin Chen-Chuan Chang. Statistical Schema Matching across Web Query Interfaces. In *SIGMOD*, pages 217–228, New York, New York, USA, June 2003. ACM Press. ISBN 158113634X. doi: 10.1145/872757.872784. URL http://dl.acm.org/citation.cfm?id=872757.872784.

[70] H. V. Jagadish and Frank Olken. Database management for life sciences research. *SIGMOD Record*, 33(2):15–20, 2004. doi: 10.1145/1024694.1024697. URL http://doi.acm.org/10.1145/1024694.1024697.

[71] DR Karger, K Bakshi, and David Huynh. Haystack: A Customizable General-Purpose Information Management Tool for End Users of Demistructured Fata. In *CIDR*, 2005. URL http://www.cs.brown.edu/courses/cs295-11/2006/haystack.pdf.

[72] J.M. Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, September 1999. ISSN 0004-5411. doi: 10.1145/324133.324140. URL http://portal.acm.org/citation.cfm?id=324140http://dl.acm.org/citation.cfm?id=324133.324140.

[73] Nick Koudas, Chen Li, Anthony K. H. Tung, and Rares Vernica. Relaxing Join and Selection Queries. In *VLDB*, pages 199–210, September 2006. URL http://dl.acm.org/citation.cfm?id=1182635.1164146.

[74] George Lakoff. *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. University of Chicago Press, 1990. ISBN 9780226468044.

[75] Dongwon Lee. *Query Relaxation for XML Model*. PhD thesis, University of California, Los Angeles, 2002. URL http://pike.psu.edu/publications/dongwon-dissertation.pdf.

[76] Yunyao Li, Cong Yu, and H. V. Jagadish. Schema-Free XQuery. *VLDB*, 30:72–83, August 2004. URL http://dl.acm.org/citation.cfm?id=1316689.1316698.

[77] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *PVLDB*, 3(1):1338–1347, 2010. URL http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/papers/R118.pdf.

[78] David B. Lomet and Elke A. Rundensteiner. Special Issue on Data Transformations. *IEEE Data Eng. Bull.*, 22(1), 1999. URL http://dblp.uni-trier.de/rec/bibtex/journals/debu/C99http://dblp.uni-trier.de/db/journals/debu/C99.html.

[79] Jayant Madhavan, Philip A P.A. Bernstein, AnHai Doan, and Alon Y Halevy. Corpus-based Schema Matching. In *ICDE*, pages 57–68. IEEE, 2005. ISBN 0-7695-2285-8. doi: 10.1109/ICDE.2005.39. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1410106.

[80] Mausam, Michael Schmitz, Stephen Soderland, Robert Bart, and Oren Etzioni. Open language learning for information extraction. In *EMNLP-CoNLL*, pages 523–534. ACL, 2012. ISBN 978-1-937284-43-5. URL http://www.aclweb.org/anthology/D12-1048.

[81] Marco Merlini and Gheorghe Lazarovici. Settling Discovery Circumstances, Dating and Utilization of the Tărtăria Tablets. *Acta Terrae Septemcastrensis Journal, Proceedings of the International Colloquium: The Carpathian Basin and its Role in the Neolithisation of the Balkan Peninsula*, VII:111–195, 2008. URL http://eprints.jiia.it:8080/167/.

[82] Zoltán Miklós, Nicolas Bonvin, Paolo Bouquet, Michele Catasta, Daniele Cordioli, Peter Fankhauser, Julien Gaugaz, Ekaterini Ioannou, Hristo Koshutanski, Antonio Maña, Claudia Niederée, Themis Palpanas, and Heiko Stoermer. From Web Data to Entities and Back. *CAiSE*, pages 302–316, June 2010. URL http://dl.acm.org/citation.cfm?id=1883784.1883817.

[83] Simon Miles, Paul T. Groth, Miguel Branco, and Luc Moreau. The requirements of using provenance in e-science experiments. *J. Grid Comput.*, 5(1):1–25, 2007. doi: 10.1007/s10723-006-9055-3. URL http://dx.doi.org/10.1007/s10723-006-9055-3.

[84] Renée J Miller and Periklis Andritsos. On Schema Discovery. *IEEE Data Eng. Bull.*, 26(3):40–45, 2003. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.6526&rep=rep1&type=pdf.

[85] Rupert G Miller and Rupert G Miller Jr. *Beyond ANOVA: Basics of Applied Statistics (Texts in Statistical Science Series)*. Chapman & Hall/CRC, January 1997. ISBN 0412070111. URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0412070111.

[86] Enrico Minack, Raluca Paiu, Stefania Costache, Gianluca Demartini, Julien Gaugaz, Ekaterini Ioannou, Paul-Alexandru Chirita, and Wolfgang Nejdl. Leveraging Personal Metadata for Desktop Search: The Beagle++ System. *Web Semant. Sci. Serv. Agents World Wide Web*, 8(1):37–54, March 2010. ISSN 15708268. doi: 10.1016/j.websem.2009.12.001. URL http://dx.doi.org/10.1016/j.websem.2009.12.001.

[87] Daniel L Moody. Metrics for Evaluating the Quality of Entity Relationship Models. *Concept. Model. – ER '98*, pages 211–225, 1998. URL http://www.springerlink.com/content/l6xd8vxhyncep9pf.

[88] Samer Nassar, Jörg Sander, and Corrine Cheng. Incremental and effective data summarization for dynamic hierarchical clustering. In *Proc. 2004 ACM SIGMOD Int. Conf. Manag. data*, SIGMOD '04, pages 467–478, New York, NY, USA, 2004. ACM. ISBN 1-58113-859-8. doi: http://doi.acm.org/10.1145/1007568.1007621. URL http://doi.acm.org/10.1145/1007568.1007621.

[89] Svetlozar Nestorov, Jeffrey Ullman, Janet Wiener, and Sudarshan Chawathe. Representative objects: Concise representations of semistructured, hierarchical data. In *Proc. Thirteen. Int. Conf. Data Eng.*, pages 79–90, 1997. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.9024.

[90] Régis S Paul, Guillaume Raschia, Noureddine Mouaddib, and Lina Polytech Nantes. General purpose database summarization. In *VLDB '05 Proc. 31st Int. Conf. Very large data bases*, pages 733–744, 2005. URL http://portal.acm.org/citation.cfm?id=1083678.

[91] Charles Sanders Peirce. On a New List of Categories. In *Proceedings of the American Academy of Arts and Sciences*, volume 7, pages 287–298, 1868.

[92] Fuchun Peng and Andrew McCallum. Information extraction from research papers using conditional random fields. *Inf. Process. Manage.*, 42(4):963–979, 2006. URL http://maroo.cs.umass.edu/pdf/IR-329.pdf.

[93] A. W. G. Pike, D. L. Hoffmann, M. García-Diez, P. B. Pettitt, J. Alcolea, R. De Balbín, C. González-Sainz, C. de las Heras, J. A. Lasheras, R. Montes, and J. Zilhão. U-series dating of paleolithic art in 11 caves in spain. *Science*, 336(6087):1409–1413, 2012. doi: 10.1126/science.1219957. URL http://www.sciencemag.org/content/336/6087/1409.abstract.

[94] Leo L Pipino, Yang W Lee, and Richard Y Wang. Data quality assessment. *Commun. ACM*, 45(4):211–218, 2002. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/505248.506010. URL http://portal.acm.org/citation.cfm?id=506010&coll=ACM&dl=ACM&CFID.

[95] Neoklis Polyzotis, Minos Garofalakis, and Yannis Ioannidis. Approximate XML Query Answers. In *SIGMOD*, page 263, New York, New York, USA, June 2004. ACM Press. ISBN 1581138598. doi: 10.1145/1007568.1007599. URL http://dl.acm.org/citation.cfm?id=1007568.1007599.

[96] Jay M. Ponte and W. Bruce Croft. A Language Modeling Approach to Information Retrieval. In *SIGIR*, pages 275–281, New York, New York, USA, August 1998. ACM Press. ISBN

1581130155. doi: 10.1145/290941.291008. URL http://dl.acm.org/citation.cfm?id=290941.291008.

[97] Erhard Rahm, Philip A. Bernstein, and Jayant Madhavan. A Survey of Approaches to Automatic Schema Matching. *VLDB*, 10(4):334–350, September 2001. ISSN 10668888. doi: 10.1007/s007780100057. URL http://www.springerlink.com/index/10.1007/s007780100057http://www.dia.uniroma3.it/~vldbproc/009_049.pdf.

[98] Howard Robinson. Substance. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2014 edition, 2014.

[99] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Commun. ACM*, 18(11):613–620, November 1975. ISSN 00010782. doi: 10.1145/361219.361220. URL http://dl.acm.org/citation.cfm?id=361219.361220.

[100] Arjun N. Saxena. *Invention of Integrated Circuits: Untold Important Facts*. International series on advances in solid state electronics and technology. World Scientific Publishing Company, Incorporated, 2009. ISBN 9789812814463. URL http://books.google.ch/books?id=z7738Wq-j-8C.

[101] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, September 2005. ISSN 0163-5808. doi: 10.1145/1084805.1084812. URL http://dx.doi.org/10.1145/1084805.1084812.

[102] Marios Skounakis, Mark Craven, and Soumya Ray. Hierarchical hidden markov models for information extraction. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 427–433. Morgan Kaufmann, 2003. URL http://pages.cs.wisc.edu/~sray/papers/hhmm.ijcai03.pdf.

[103] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999. URL http://homes.cs.washington.edu/~soderlan/soderland_ml99.pdf.

[104] Partha Pratim Talukdar, Joseph Reisinger, Marius Pasca, Deepak Ravichandran, Rahul Bhagat, and Fernando Pereira. Weakly-supervised acquisition of labeled class instances using graph random walks. In *EMNLP*, pages 582–590. ACL, 2008. URL http://aclweb.org/anthology/D/D08/D08-1061.pdf.

[105] Nicholas E. Taylor and Zachary G. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors,

Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006, pages 13–24. ACM, 2006. ISBN 1-59593-256-9. doi: 10.1145/1142473.1142476. URL http://doi.acm.org/10.1145/1142473.1142476.

[106] Petros Venetis, Alon Y. Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *PVLDB*, 4(9):528–538, 2011. URL http://www.vldb.org/pvldb/vol4/p528-venetis.pdf.

[107] Ke Wang and Huiqing Liu. Schema Discovery for Semistructured Data. In *KDD*, pages 271–274, 1997. URL http://dblp.uni-trier.de/rec/bibtex/conf/kdd/WangL97http://www.aaai.org/Papers/KDD/1997/KDD97-057.pdf.

[108] Ke Wang and Huiqing Liu. Discovering Structural Association of Semistructured Data. *IEEE Trans. Knowl. Data Eng.*, 12(3):353–371, 2000. doi: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=846290. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=846290.

[109] Linda Wetzel. Types and tokens. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2014 edition, 2014.

[110] Wilson Wong, Wei Liu, and Mohammed Bennamoun. Ontology learning from text: A look back and into the future. *ACM Computing Surveys*, 44(4):1–36, August 2012. doi: 10.1145/2333112.2333115. URL http://dx.doi.org/10.1145/2333112.2333115.

[111] Fei Wu and Daniel S. Weld. Open information extraction using wikipedia. In Jan Hajic, Sandra Carberry, and Stephen Clark, editors, *ACL*, pages 118–127. The Association for Computer Linguistics, 2010. ISBN 978-1-932432-66-4, 978-1-932432-67-1. URL http://www.aclweb.org/anthology/P10-1013.

[112] Gang Wu, Kuo Zhang, Can Liu, and Juanzi Li. Adapting Prime Number Labeling Scheme for Directed Acyclic Graphs. In *Database Syst. Adv. Appl.*, pages 787–796. 2006. URL http://dx.doi.org/10.1007/11733836$\delimiter"026E30F$_56.

[113] Cong Yu and H. V. Jagadish. Schema Cummarization. In *Very Large Data Bases*, pages 319–330, Seoul, Korea, 2006. VLDB Endowment. doi: http://dx.doi.org/10.1016/S0169-023X(96)00007-9. URL http://portal.acm.org/citation.cfm?id=1164156http://dx.doi.org/10.1016/S0169-023X(96)00007-9.

[114] Xuan Zhou, Julien Gaugaz, Wolf-Tilo Balke, and Wolfgang Nejdl. Query Relaxation Using Malleable Schemas. *SIGMOD*, pages 545–556, 2007. doi: http://doi.acm.org/10.1145/1247480.1247541. URL http://dx.doi.org/10.1145/1247480.1247541.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:
http://code.google.com/p/classicthesis/