Online Ordering of Overlapping Data Sources

[#]Mariam Salloum [φ]Xin Luna Dong[*] [α]Divesh Srivastava [#]Vassilis J. Tsotras
[#]UC Riverside [φ]Google Inc. [α]AT&T Labs-Research
[#](msalloum, tsotras)@cs.ucr.edu [φ] lunadong@google.com
[α] divesh@research.att.com

ABSTRACT

Data integration systems offer a uniform interface for querying a large number of autonomous and heterogeneous data sources. Ideally, answers are returned as sources are queried and the answer list is updated as more answers arrive. Choosing a good ordering in which the sources are queried is critical for increasing the rate at which answers are returned. However, this problem is challenging since we often do not have complete or precise statistics of the sources, such as their coverage and overlap. It is further exacerbated in the Big Data era, which is witnessing two trends in Deep-Web data: first, obtaining a full coverage of data in a particular domain often requires extracting data from thousands of sources; second, there is often a big variation in overlap between different data sources.

In this paper we present **OASIS**, an **O**nline query **A**nswering **S**ystem for overlapp**I**ng **S**ources. **OASIS** has three key components for source ordering. First, the **Overlap Estimation** component estimates overlaps between sources according to available statistics under the *Maximum Entropy* principle. Second, the **Source Ordering** component orders the sources according to the new contribution they are expected to provide, and adjusts the ordering based on statistics collected during query answering. Third, the **Statistics Enrichment** component selects critical missing statistics to enrich at runtime. Experimental results on both real and synthetic data show high efficiency and scalability of our algorithm.

1. INTRODUCTION

The Big Data era requires not only the capability of managing and querying a large volume of data, but also the capability of querying data from a large number of sources. Dalvi et al. [3] recently presented interesting observations on Deep-Web data. First, obtaining a full coverage of data in a particular domain often requires extracting data from thousands of sources. Second, there is often a big overlap between sources. For example, with strong head aggregators such as *yelp.com*, collecting homepage URLs for 70% restaurants that are mentioned by some websites required only top-10 sources; however, collecting URLs for 90% restaurants required top-1000 sources, and collecting URLs for 95% restaurants required top-5000 sources.

[*]This work was done while the author was working at AT&T Labs-Research.

Proceedings of the VLDB Endowment, Vol. 7, No. 3

Whereas it is often desirable to build a central repository that aggregates data from all sources, this approach is not often feasible because of both the large number of sources and the continuous updates of data in some domains. Therefore, many data integrators query relevant Deep-Web sources, as well as other aggregators, at runtime upon receiving a user query. Since there is often some resource restriction, such as bandwidth for data retrieval and CPU cycles for answer processing, an integrator cannot query all sources and process all answers at the same time, but can only do so sequentially or semi-sequentially (*e.g.*, each time querying a few sources in parallel). Current integrators, such as *KAYAK.com* and *tripadvisor.com*, typically query only tens to hundreds of sources rather than all available sources. Even so, retrieving all answers can take a long time, so some of them display query results in an online fashion: once the integrator retrieves answers from some sources, it returns the partial answers to users; then, it dynamically updates the answer list as new answers arrive.

In general, it is desirable to return as many answers as possible right from the beginning, especially when we wish to query thousands of sources. Hence, choosing a good ordering in which the sources are queried is critical for increasing the rate at which answers are returned. If all sources that an integrator queries return distinct answers, source ordering is simple: sources are queried in decreasing order of their coverage. However, typically different sources contain a lot of overlapping data, and such overlaps can make ordering much more complex.

EXAMPLE 1.1. *Consider five relevant sources A, B, C, D, and E shown in Figure 1 for a given query Q. Each area in the Venn diagram is disjoint from others and represents the answers returned by certain sources but not others. Each area contains a number that represents the number of answer tuples for Q that reside in that area. For example, there are 4 answers returned by source C but not provided by any other source. Similarly, there is 1 answer covered by both C and E.*

Among the sources, A and B each returns the largest number of answers (14 and 13 respectively); however, they have a high overlap: there are 9 answers in common. If we query them first, we obtain 18 distinct answers after querying 2 sources. But if we query A and C first, we obtain 21>18 distinct answers. □

As the example shows, overlap between sources is an important factor to consider in source ordering. In this paper, we consider Select-Project queries within each individual source, which are typical for queries posed on Deep-Web sources. The coverage and overlap of sources vary for different queries; thus, we need to estimate the statistics in a query specific manner using (i) query independent coverage and overlap of sources, (ii) statistics in the log for answering other queries, and (iii) query specific selectivity information, much like cardinality and selectivity statistics are

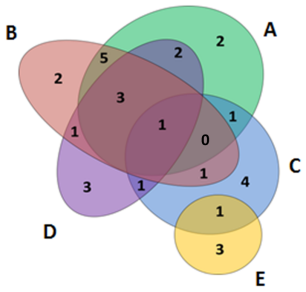

Figure 1: Venn diagram for the five sources in Example 1.1.

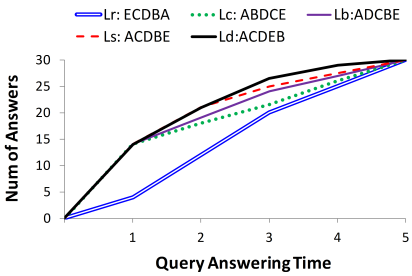

Figure 2: Area-under-the-curve for various orderings.

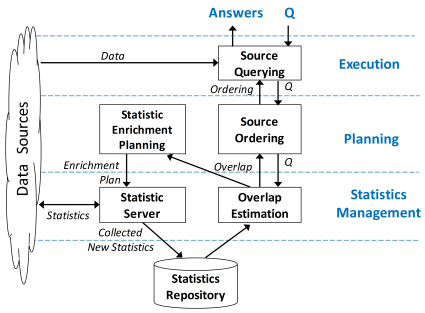

Figure 3: OASIS system architecture.

estimated in query optimization [9] (details skipped since it is orthogonal to this paper). The problem of ordering a large number of sources efficiently at runtime is non-trivial. The number of overlaps between a subset of sources is exponential in the number of the sources and we seldom know all of these overlaps in practice. Even the statistics we do have may not be precise for the given query. Such incomplete and imprecise information about overlaps makes source ordering even more challenging.

In this paper, we propose a system called **OASIS** (**O**nline query **A**nswering **S**ystem for overlapp**I**ng **S**ources). The key idea of **OASIS** is to order the sources according to their coverage, overlap, and querying time (or access and transfer cost) at runtime such that answers are returned at a high rate. **OASIS** solves the problem of incomplete and imprecise information about overlaps in three ways. First, it estimates missing overlaps from the given statistics by applying the *Maximum Entropy* principle, which does not introduce correlations that are unknown *a priori* [12]. Second, at the time of querying, it collects statistics regarding the queried sources, such as their precise coverage and the size of their union with respect to the specific query, and adjusts source ordering dynamically. Third, it also identifies critical missing overlaps, the knowledge of which may lead to a better source ordering, and computes them from already obtained results. We design each component of the system carefully such that it is efficient and scalable. Note that a good caching strategy would improve query answering performance but is orthogonal to this paper.

In this paper, we consider how to order thousands of sources at runtime, which enables leveraging the potential of the rich information on the Web. In particular, we make the following contributions.

1. We present an algorithm that estimates overlaps not known *a priori* using the Maximum Entropy principle. Moreover, our algorithm considers an overlap only if it is likely to have a non-zero value, so significantly reduces the number of overlaps we need to consider and scales to thousands of sources.
2. We propose a *dynamic* source-ordering strategy that orders sources according to their coverage, cost, and overlaps.
3. We propose an algorithm that selects critical missing statistics for inquiry at querying time for better source ordering.
4. We experimented on real and synthetic data sets and empirically show that our source ordering algorithm is scalable in the number of sources and can significantly increase the speed at which answers are returned.

The rest of the paper is structured as follows. Section 2 discusses related work. Section 3 describes the architecture of **OASIS**. Sections 4-6 describe key technical components of **OASIS**, including overlap estimation, source ordering, and statistics enrichment. Section 7 presents experimental results, and Section 8 concludes.

2. RELATED WORK

There has been a lot of work that considers leveraging the overlap or redundancy information for source selection and ordering. Florescu et al. [5] assumed each source is categorized into one or more *domains* (*e.g.*, DB papers vs. AI papers) and leveraged probabilistic information about domain overlaps to select top-k useful sources for query answering. Roth et al. [10] considered both source overlaps and source connections in a peer data management system (PDMS) to select a subset of sources for query rewriting. Bleiholder et al [1] considered selecting sources for answering `Join` queries on life science data; instead of *minimizing* duplicates, they tried to *maximize* duplicates because they assume that different sources would provide data on different aspects of an entity. All of the above works assume *a priori* knowledge about coverage and overlap statistics.

Chokshi et al. [2] leveraged overlap statistics to answer queries over the most relevant set of text documents; the system learns overlaps between collections of documents via sampling but focuses on pair-wise overlaps. Vassalos et al. [14] discussed source selection based on source overlaps and outlined the challenges in computing the exponential number of source overlaps and utilizing those statistics for query answering. Whereas [14] provides useful insights, it did not propose a concrete system to estimate source-level overlaps.

The StatMiner system [8] considers coverage and overlap in *static* source ordering and is most relevant to our work. It assumes that both sources and queries are characterized into "class hierarchies"; based on sample data, it learns coverage and overlap statistics between *classes* (rather than data sources) and decides the best query plan accordingly. There are three major differences between StatMiner and **OASIS**. (1) StatMiner learns statistics for *all* overlaps; this strategy works for a small number of *classes* but does not scale for a large number of *sources*. **OASIS** provides an algorithm to estimate missing overlaps thus can handle a large number of (*e.g.*, thousands of) sources. (2) StatMiner requires a sufficient sample data set to learn overlaps; **OASIS** can deal with stale, inaccurate, and missing statistics by collecting new statistics at query answering time and thus enriching critical statistics. (3) StatMiner learns overlaps off-line; **OASIS** is able to generate overlap estimates quickly at runtime, thus improve source ordering dynamically as more statistics are collected.

Source ordering has also been studied in the presence of varying source accuracy and copying relationships between sources [7, 11]. **OASIS** differs from them in three aspects: (1) their goal is to improve *correctness* of returned answers, while **OASIS** considers scenarios where data is fairly clean and focuses on *coverage* of returned answers; (2) they assume *a priori* knowledge of source accuracy and copying relationships, while **OASIS** is tolerant to impre-

cise and incomplete statistics on source overlaps; (3) they conduct source ordering offline, while **OASIS** conducts source ordering at runtime and can adjust ordering based on updated statistics from already queried sources. We leave consideration of both source overlap and source accuracy for future work.

Recently [9] applied the Maximum Entropy principle in estimating the cardinality of a new query based on the cardinality of a fixed set of queries; [13] adapted the capture-recapture method in estimating the cardinality of a query in the crowdsourcing environment. We tackle a different problem in this paper.

3. OASIS OVERVIEW

We start with an overview of the **OASIS** system and define the problems we need to solve in building such a system.

3.1 Motivation

Consider a query Q and a set of data sources Ω. We wish to answer Q on each source in Ω and return the union of the results, denoted by $Q(\Omega)$. The main goal in building **OASIS** is to maximize the rate at which answers are returned. While other performance metrics can be adopted, the rate of the retrieved answers is best measured by the *area-under-the-curve*, where the X-axis is the query-answering time, the Y-axis is the number of available answers, and the curve plots the total number of *distinct* answers available at each time during query answering [11].

Most domains have thousands of data sources, thus sources cannot be queried at the same time because of resource restrictions such as bandwidth limitation. When sources are queried semi-sequentially, the order in which they are queried can significantly affect the area-under-the-curve, as we show next.

EXAMPLE 3.1. *Continue with Example 1.1. From the 120 possible orderings of these sources, we consider five of them: L_r : $ECDBA$, L_c : $ABDCE$, L_b : $ADCBE$, L_s : $ACDBE$, and L_d : $ACDEB$. In later sections, we will correlate these orderings to different ordering strategies. Figure 2 shows the curve for each ordering, assuming we query the sources sequentially and retrieving data from each source takes one time unit. We observe that L_d returns answers most quickly and L_r returns answers most slowly. Indeed, L_d has the largest area-under-the-curve (118), and L_r has the smallest one (95).* □

Determining the ordering of the sources would benefit from three types of knowledge. First, sources that have a high *coverage* would return more answers. Second, we wish to choose a source that has a low *overlap* with the already queried sources: such a source will contribute more *new* answers than another source with the same coverage. Third, a source with low access and transfer time, which we refer to as *cost*, will yield answers faster. We next formally define these three measures considered in source ordering.

DEFINITION 3.2 (COVERAGE). *Let $S \in \Omega$ be a source and Q be a query. The* coverage *of S w.r.t. Q is defined as $P_Q(S) = \frac{|Q(S)|}{|Q(\Omega)|}$, where $|Q(S)|$ denotes the number of answers returned by S and $|Q(\Omega)|$ denotes the total number of answers.* □

DEFINITION 3.3 (OVERLAP). *Let $\widehat{S} \subseteq \Omega$ denote a subset of sources and Q be a query. The* overlap *of $\widehat{S}$ w.r.t. Q is defined as $P_Q(\cap \widehat{S}) = \frac{|\cap_{S \in \widehat{S}} Q(S)|}{|Q(\Omega)|}$, where $|\cap_{S \in \widehat{S}} Q(S)|$ denotes the number of answers returned by every source in $\widehat{S}$ and $|Q(\Omega)|$ denotes the total number of answers.* □

The coverage of a source can be thought of as the probability that an answer is provided by the source, and similarly for overlap between sources (this justifies the use of the $P_Q(...)$ notation). Note that information on coverage of unions of sources can benefit source ordering as well. We skip union coverage for simplicity because our framework and techniques can be easily extended to directly incorporate such union information (see Section 4.2).

DEFINITION 3.4 (COST). *Let $S \in \Omega$ be a source and Q be a query. The* cost *of querying S (in terms of time) w.r.t. Q is defined as $Cost_Q(S) = CC(S) + TC(S) \times |Q(S)|$, where $CC(S)$ denotes the connection (or access) cost of S, $TC(S)$ denotes the per-tuple transfer overhead time of S, and $|Q(S)|$ denotes the total number of answers returned by S.* □

We can now formally define the key problem in building **OASIS**, the *source-ordering problem*. For simplicity, our definition assumes that we query relevant sources sequentially; our techniques can be easily extended to querying a few sources in parallel and our experiments show the effectiveness of this extension (Section 7.2).

DEFINITION 3.5 (SOURCE ORDERING). *Let Ω denote a set of sources and Q be a query. The input to the* source-ordering *problem includes (1) the coverage of each source $\mathbf{C} = \{P_Q(S) | S \in \Omega\}$, (2) a subset of overlaps $\mathbf{O} = \{P_Q(\cap \widehat{S}) | \widehat{S} \in \widehat{\mathbf{S}}\}$ where $\widehat{\mathbf{S}} \subseteq 2^{\Omega}$, and (3) the cost of querying each source $\mathbf{Cost} = \{Cost_Q(S) | S \in \Omega\}$. The output is an ordering of the sources such that querying Ω in this order obtains the largest area-under-the-curve.* □

3.2 Basic solution and challenges

The source-ordering problem has been studied in [11]. A greedy algorithm, GREEDY, was proposed: each time we select the source with the highest ratio of its *residual contribution* over cost until all sources are exhausted. Given a set of already queried sources $\widehat{S}$, the residual contribution of $S \in \Omega \setminus \widehat{S}$ with respect to $\widehat{S}$ is formally defined as $P_Q(S \setminus \widehat{S}) = |Q(\widehat{S} \cup \{S\})| - |Q(\widehat{S})|$. It is easy to derive that

$$P_Q(S \setminus \widehat{S}) = \sum_{i=0}^{|\widehat{S}|} (-1)^i \sum_{\widehat{S}_0 \subseteq \widehat{S}, |\widehat{S}_0| = i} |S \cap \widehat{S}_0|. \quad (1)$$

This greedy approach does not necessarily generate the optimal ordering: given three sources $S_1 : \{t_1, \ldots, t_{50}\}$, $S_2 : \{t_{51}, \ldots, t_{100}\}$ and $S_3 : \{t_{25}, \ldots, t_{75}\}$, each with the same querying cost, the optimal ordering is S_1, S_2, S_3 (area-under-the-curve is 50+100+100=250), whereas the greedy solution would generate an ordering of S_3, S_2, S_1 (area-under-the-curve is 51+76+100=227). However, GREEDY has a constant approximation bound 2; that is, the area-under-the-curve of the solution by GREEDY is at least half of that of the optimal solution. We re-state the relevant results from [11] as follows.

THEOREM 3.6. *Assume a polynomial-time oracle for residual-contribution computation. (1) The decision problem form of the source-ordering problem (*can a specific area-under-the-curve of A be achieved*) is in NP. (2)* GREEDY *finds a 2-approximation solution in polynomial time.* □

When we apply GREEDY, we face the challenge of computing the residual contribution for each source from the often very limited statistics: (1) we may not know some of the overlaps; in other words, Definition 3.5 requires only $\widehat{\mathbf{S}} \subseteq 2^{\Omega}$ as input; (2) we may only *estimate* the coverage and overlap w.r.t. Q according to the overall coverage of sources and overlap between sources (not specific to the particular query), statistics in the log for answering other queries, and some selectivity information. Such limited information makes it impossible to precisely compute residual contribution and so makes it even harder to find the optimal ordering.

EXAMPLE 3.7. *Continue with the motivating example. If we know all coverage and overlaps precisely,* GREEDY *generates the optimal ordering* $L_d : ACDEB$. *However, assume our input is as follows, where we have (precise) overlap information for only* a few *subsets of sources.*

$P_Q(A)$	$= 0.47$	$P_Q(\cap\{A, B\})$	$= 0.30$
$P_Q(B)$	$= 0.43$	$P_Q(\cap\{A, D\})$	$= 0.20$
$P_Q(C)$	$= 0.30$	$P_Q(\cap\{A, B, C, D\})$	$= 0.03$
$P_Q(D)$	$= 0.37$		
$P_Q(E)$	$= 0.13$		

Only considering coverage will generate ordering $L_c = ABDCE$ *while a careless use of existing overlaps (we give details in Section 7 for this baseline approach) will generate ordering* $L_b = ADCBE$; *both are sub-optimal according to Figure 2.* □

3.3 The OASIS Architecture

OASIS solves the afore-mentioned incomplete-information problem in three ways. First, to better estimate residual contribution, **OASIS** estimates the unavailable overlaps from the given **C** (coverage) and **O** (overlap) by applying the *Maximum Entropy* principle.

Second, instead of ordering the sources upfront, **OASIS** orders the sources in a dynamic fashion: as **OASIS** queries a source S, it updates statistics related to S, including coverage of S and union of already queried sources including S, and takes the new information into account in selecting the next source to query. Such statistics are easy to collect along with query answering.

Third, **OASIS** finds critical missing statistics for improving ordering and computes them from queried sources on-the-fly.

The architecture of **OASIS** (Figure 3) contains three layers.

1. The bottom layer, the *Statistics management* layer, contains two components. The **Statistics Server** component takes charge of statistics collection. *Offline* it collects source coverage and overlap information provided by sources (or from the log); *online* it collects such statistics for the given query according to answers from the queried sources. The collected statistics are stored at the **Statistics Repository**. Since the stored statistics may be incomplete, the **Overlap Estimation** component estimates upon receiving a query the overlap statistics that are unavailable.
2. The middle layer, the *Planning* layer, also contains two components. The **Source Ordering** component decides the ordering in which we query the sources according to statistics provided by the *Statistics management* layer. The **Statistics Enrichment** component selects a set of overlaps to enrich and sends the request to the **Statistics Server**.
3. The top layer, the *Execution* layer, contains the **Source Querying** component, which receives a user query, queries the sources according to the ordering decided by **Source Ordering**, and returns the retrieved answers along the way.

The rest of the paper describes the key components of **OASIS** in more detail: **Overlap Estimation** (Section 4), **Source Ordering** (Section 5), and **Statistics Enrichment Planning** (Section 6).

4. OVERLAP ESTIMATION

This section studies the overlap-estimation problem; that is, based on the given coverage and overlap information, how to estimate the rest of the overlaps. We first formalize the problem (Section 4.1), then present our basic solution by applying the *maximum entropy* principle (Section 4.2), and finally describe an iterative algorithm that is scalable to thousands of sources (Section 4.3).

4.1 Problem definition

Given a set of sources Ω, each query answer must be provided by a particular subset $\widehat{S} \subseteq \Omega$ of sources; we thus consider $2^{|\Omega|}$ events, each representing an answer being provided by all and only sources in $\widehat{S}$. We associate each event with a Ω-*tuple* where a *positive expression* **S** indicates that $S \in \widehat{S}$ and a *negative expression* S' indicates that $S \notin \widehat{S}$. Interchangeably, we denote the tuple by $V_{\widehat{S}}$. We say that the *cardinality* of $V_{\widehat{S}}$ is $|\widehat{S}|$ and call $V_{\widehat{S}}$ a $|\widehat{S}|$-*pos tuple*. For example, tuple $V_{\{A,B\}} = \mathbf{AB}C'D'E'$ is a 2-pos tuple and denotes the event that A, B provide an answer but C, D, E do not. We denote by $P(T)$ the probability of the event represented by tuple T. In the rest of the paper, we abuse notation and omit P for simplicity; that is, $V_{\{A,B\}}$ denotes a 2-pos variable representing the probability of only A, B providing an answer. We denote the set of all variables by $\mathbf{V}(\Omega)$; their sum is 1.

Our goal in overlap estimation is to estimate the value of each Ω-tuple variable based on the input statistics.

DEFINITION 4.1 (OVERLAP ESTIMATION). *Let* Ω *be a set of sources and* Q *be a query. Given* **C** *and* $\mathbf{O} = \{P_Q(\cap\widehat{S})|\widehat{S} \in \widehat{\mathbf{S}}\}$, *the* overlap-estimation problem *estimates the value for each* Ω-*tuple variable* $v \in \mathbf{V}(\Omega)$. □

Note that there is an exponential number of variables in $\mathbf{V}(\Omega)$ so the output also has an exponential size. Instead, we only output estimates for variables whose values are not close to 0; the number of such variables is typically much smaller and bounded by $|Q(\Omega)|$, because there are at most $|Q(\Omega)|$ answers. Note also that the variables in $\mathbf{V}(\Omega)$ do not directly represent the overlaps of subsets of sources, but we can infer the overlaps from them. Actually, as we show in the next section, we can infer the residual contributions directly from $\mathbf{V}(\Omega)$ more efficiently than from overlaps.

4.2 Basic solution

We can formulate the coverage of a source S as a linear constraint with $2^{|\Omega|-1}$ variables where S is "positively" expressed. For example, given five sources A through E, the coverage of A can be formulated as

$P_Q(A) = \mathbf{A}B'C'D'E' + \mathbf{A}B'C'D'\mathbf{E} + \mathbf{A}B'C'\mathbf{D}E' + \mathbf{A}B'C'\mathbf{DE} + \mathbf{A}B'\mathbf{C}D'E' + \mathbf{A}B'\mathbf{C}D'\mathbf{E} + \mathbf{A}B'\mathbf{CD}E' + \mathbf{A}B'\mathbf{CDE} + \mathbf{ABC}'D'E' + \mathbf{ABC}'D'\mathbf{E} + \mathbf{ABC}'\mathbf{D}E' + \mathbf{ABC}'\mathbf{DE} + \mathbf{ABC}D'E' + \mathbf{ABC}D'\mathbf{E} + \mathbf{ABCD}E' + \mathbf{ABCDE}$.

Similarly, we can formulate the overlap of sources in $\widehat{S}$ as a linear constraint with $2^{|\Omega|-|\widehat{S}|}$ variables where the sources in $\widehat{S}$ are "positively" expressed (similar for union). For example,

$P_Q(\cap\{A, B\}) = \mathbf{AB}C'D'E' + \mathbf{AB}C'D'\mathbf{E} + \mathbf{AB}C'\mathbf{D}E' + \mathbf{AB}C'\mathbf{DE} + \mathbf{ABC}D'E' + \mathbf{ABC}D'\mathbf{E} + \mathbf{ABCD}E' + \mathbf{ABCDE}$.

There are often many more variables than constraints, so many solutions can be generated that satisfy the constraints. Among them, we choose the solution with the *Maximum Entropy*, which essentially corresponds to the distribution with the highest likelihood taking into account the provided statistics and not assuming any extra correlation between the sources [12]. Formally, we estimate the overlaps by solving the following optimization problem, which we call MAXENT. We can apply existing technology (such as [4]) to solve the optimization problem.

$$\text{maximize} \quad -\sum_{v \in \mathbf{V}(\Omega)} v \log v$$

$$\text{subject to} \quad (1) \sum_{v \in \mathbf{V}(\Omega)} v = 1$$

(2) constraints derived from **C** and **O**

Table 1: Estimates of variables in Example 3.7. by the basic approach.

Variable	Value
$\boldsymbol{A}\ B'\ C'\ D'\ E'$	.07
$A'\ \boldsymbol{B}\ C'\ D'\ E'$	.05
$A'\ B'\ \boldsymbol{C}\ D'\ E'$	.19
$A'\ B'\ C'\ \boldsymbol{D}\ E'$	.16
$A'\ B'\ C'\ D'\ \boldsymbol{E}$	.05
$\boldsymbol{A}\ \boldsymbol{B}\ C'\ D'\ E'$	.27
$\boldsymbol{A}\ B'\ C'\ \boldsymbol{D}\ E'$	.10
$A'\ \boldsymbol{B}\ \boldsymbol{C}\ \boldsymbol{D}\ \boldsymbol{E}$	.08
$\boldsymbol{A}\ \boldsymbol{B}\ \boldsymbol{C}\ \boldsymbol{D}\ E'$	.03

EXAMPLE 4.2. *Continue with Example 3.7. There are five sources so $2^5 = 32$ variables. We maximize $-\sum_{v\in\mathbf{V}(\Omega)} v \log v$ under constraint $\sum_{v\in\mathbf{V}(\Omega)} v = 1$ and the 8 other constraints given in Example 3.7. Table 1 shows the estimates for the variables whose values are larger than .01 (we consider that others have a value of 0). Observe that although 32 variables were used to define the set of constraints, only 9 were determined to be non-zero.* □

4.3 A scalable algorithm

The basic solution considers $2^{|\Omega|}$ variables and thus with thousands of sources, the number can quickly become unmanageable. To make it scalable, we wish to reduce the number of variables. We observe that (1) as the number of sources in a subset increases, the overlap quickly decreases, and (2) if there are N answers (N can be estimated applying cardinality estimation techniques), at most N variables have observed values corresponding to non-empty sets. Thus, a natural approach is to exclude variables that would have very small values in the solution.

However, before we solve the optimization problem, we cannot know which variables would have small values. We thus start with considering variables with low cardinality, and more likely to have large values *a priori*, and iteratively add high-cardinality variables. In particular, we expand the set of variables as follows.

1. In the first iteration, we start with variables $\bar{V}$, containing only (1) those whose positive sources correspond to sources in the given coverage and overlaps, and (2) the 0-pos variable (denoting the probability that no source provides the answer). We assume other variables all have value 0.
2. In the k-th ($k > 1$) iteration, for each variable $V_{\widehat{S}}$ added to $\bar{V}$ in the $(k-1)$-th iteration, we examine the variables that indicate one more provider than $\widehat{S}$; that is, $\{V_{\widehat{S}\cup\{S\}} | S \in \Omega \setminus \widehat{S}\}$. We say that $V_{\widehat{S}}$ is a *parent* variable of $V_{\widehat{S}\cup\{S\}}$. We add a variable $V_{\widehat{S}'}$ to $\bar{V}$ only if it satisfies two conditions. First, the sum of its parents' values is not too small. The intuition is that if $V_{\widehat{S}'}$ is actually "large" but we force $V_{\widehat{S}'} = 0$, its value will "flow" to its parent variables in our solution, and so their sum should be large. Formally, we require $\sum_{V_{\widehat{S}}\in\bar{V}, \widehat{S}\subset\widehat{S}', |\widehat{S}'|-|\widehat{S}|=1} V_{\widehat{S}} \geq \theta$, where θ is a threshold. We set θ to $\frac{1}{N}$ in our implementation because a given variable must be greater than or equal to $\frac{1}{N}$ if it holds at least one answer tuple. Second, existing statistics should not indicate that the value of $V_{\widehat{S}'}$ must be low. Formally, for any overlap (similar for coverage) $P_Q(\cap\widehat{S}) \in \mathbf{O}$ and $P_Q(\cap\widehat{S}) < \theta$, we require $\widehat{S} \not\subseteq \widehat{S}'$.
3. For any existing variable in $\bar{V}$, we remove it if its value is below θ, since we deem it too small to affect source ordering.
4. We continue until there are no more variables to add. The k-th iteration would have considered all k-pos variables so the process would terminate in at most $|\Omega|$ iterations.

Table 2: Solution for each iteration in Ex. 4.3. The table includes only variables that are considered in some iteration. A dash '-' indicates that the variable was not specified in any constraint for that particular iteration.

Variable	Iter. 1	Iter. 2	Iter. 3	Iter. 4
$A'\ B'\ C'\ D'\ E'$	0	-	-	-
$\boldsymbol{A}\ B'\ C'\ D'\ E'$	0	-	-	-
$A'\ \boldsymbol{B}\ C'\ D'\ E'$	.13	.04	.04	.04
$A'\ B'\ \boldsymbol{C}\ D'\ E'$	.21	.15	.23	.23
$A'\ B'\ C'\ \boldsymbol{D}\ E'$	.12	.14	.12	.13
$A'\ B'\ C'\ D'\ \boldsymbol{E}$	.13	.03	.03	.03
$\boldsymbol{A}\ \boldsymbol{B}\ C'\ D'\ E'$	.21	.27	.27	.27
$\boldsymbol{A}\ B'\ C'\ \boldsymbol{D}\ E'$	.16	.17	.17	.17
$\boldsymbol{A}\ \boldsymbol{B}\ \boldsymbol{C}\ \boldsymbol{D}\ E'$	.06	.03	.03	.03
$A'\ \boldsymbol{B}\ \boldsymbol{C}\ D'\ E'$	-	.04	0	-
$A'\ \boldsymbol{B}\ C'\ D'\ \boldsymbol{E}$	-	.05	.09	.06
$A'\ B'\ \boldsymbol{C}\ \boldsymbol{D}\ E'$	-	.03	0	-
$A'\ B'\ \boldsymbol{C}\ D'\ \boldsymbol{E}$	-	.05	0	-
$\boldsymbol{A}\ B'\ \boldsymbol{C}\ D'\ E'$	-	0	-	-
$\boldsymbol{A}\ B'\ C'\ D'\ \boldsymbol{E}$	-	0	-	-
$A'\ \boldsymbol{B}\ C'\ \boldsymbol{D}\ E'$	-	0	-	-
$A'\ B'\ C'\ \boldsymbol{D}\ \boldsymbol{E}$	-	0	-	-
$\boldsymbol{A}\ \boldsymbol{B}\ \boldsymbol{C}\ D'\ E'$	-	0	-	-
$\boldsymbol{A}\ \boldsymbol{B}\ C'\ \boldsymbol{D}\ E'$	-	0	-	-
$\boldsymbol{A}\ \boldsymbol{B}\ C'\ D'\ \boldsymbol{E}$	-	0	-	-
$\boldsymbol{A}\ B'\ \boldsymbol{C}\ \boldsymbol{D}\ E'$	-	0	-	-
$\boldsymbol{A}\ B'\ C'\ \boldsymbol{D}\ \boldsymbol{E}$	-	0	-	-
$\boldsymbol{A}\ \boldsymbol{B}\ \boldsymbol{C}\ \boldsymbol{D}\ \boldsymbol{E}$	-	0	-	-
$A'\ B'\ \boldsymbol{C}\ \boldsymbol{D}\ \boldsymbol{E}$	-	-	.03	0
$A'\ \boldsymbol{B}\ \boldsymbol{C}\ \boldsymbol{D}\ E'$	-	-	.01	-
$\boldsymbol{A}\ B'\ \boldsymbol{C}\ D'\ \boldsymbol{E}$	-	-	0	-
$A'\ \boldsymbol{B}\ C'\ \boldsymbol{D}\ \boldsymbol{E}$	-	-	0	-
$A'\ \boldsymbol{B}\ \boldsymbol{C}\ D'\ \boldsymbol{E}$	-	-	0	-
$A'\ \boldsymbol{B}\ \boldsymbol{C}\ \boldsymbol{D}\ \boldsymbol{E}$	-	-	-	.04

EXAMPLE 4.3. *Reconsider Example 3.7.*

Iteration 1: *In the first iteration, we start with variables that correspond to coverage and overlap constraints. Five variables $\boldsymbol{A}B'C'D'E'$, $A'\boldsymbol{B}C'D'E'$, $A'B'\boldsymbol{C}D'E'$, $A'B'C'\boldsymbol{D}E'$, $A'B'C'D'\boldsymbol{E}$ are defined for $P_Q(A)$, $P_Q(B)$, $P_Q(C)$, $P_Q(D)$, $P_Q(E)$, respectively. Three variables $\boldsymbol{AB}C'D'E'$, $\boldsymbol{A}B'C'\boldsymbol{D}E'$, and $\boldsymbol{A\,B\,C\,D}\,E'$ are defined for $P_Q(\cap\{A,B\})$, $P_Q(\cap\{A,D\})$, $P_Q(\cap\{A,B,C,D\})$, respectively. Finally, variable $A'B'C'D'E'$ represents the probability of an answer not provided by any source. Thus, in the first iteration, we start with 9 variables (5 for coverage, 3 for overlaps, and 1 0-pos variable). We formulate the following constraints.*

$P_Q(A) = \boldsymbol{A}B'C'D'E' + \boldsymbol{AB}C'D'E' + \boldsymbol{A}B'C'\boldsymbol{D}E' + \boldsymbol{ABCD}E'$
$P_Q(B) = A'\boldsymbol{B}C'D'E' + \boldsymbol{AB}C'D'E' + \boldsymbol{ABCD}E'$
$P_Q(C) = A'B'\boldsymbol{C}D'E' + \boldsymbol{ABCD}E'$
$P_Q(D) = A'B'C'\boldsymbol{D}E' + \boldsymbol{A}B'C'\boldsymbol{D}E' + \boldsymbol{ABCD}E'$
$P_Q(E) = A'B'C'D'\boldsymbol{E}$
$P_Q(\cap\{A,B\}) = \boldsymbol{AB}C'D'E' + \boldsymbol{ABCD}E'$
$P_Q(\cap\{A,D\}) = \boldsymbol{A}B'C'\boldsymbol{D}E' + \boldsymbol{ABCD}E'$
$P_Q(\cap\{A,B,C,D\}) = \boldsymbol{ABCD}E'$
$1.0 = \boldsymbol{A}B'C'D'E' + A'\boldsymbol{B}C'D'E' + A'B'\boldsymbol{C}D'E' + A'B'C'\boldsymbol{D}E' + A'B'C'D'\boldsymbol{E} + \boldsymbol{AB}C'D'E' + \boldsymbol{A}B'C'\boldsymbol{D}E' + \boldsymbol{ABCD}E' + A'B'C'D'E'$

We set the threshold for considering variable expansion and reduction as $\theta = \frac{1}{N} = 0.03$. Table 2 shows the solution for each iteration of the scalable approach. Note that because of rounding, the probabilities of each column may not sum up to 1.

At the end of the first iteration, we expand variables by adding 14 variables. Consider $A'B'C'D'\boldsymbol{E}$ as an example. We examine its four children variables: $\boldsymbol{A}B'C'D'\boldsymbol{E}$, $A'\boldsymbol{B}C'D'\boldsymbol{E}$, $A'B'\boldsymbol{C}D'\boldsymbol{E}$,

and $A'B'C'\mathbf{DE}$. Every variable is expanded since the sum of its parent variables is above θ. In addition, we discard 2 variables $A'B'C'D'E'$ and $\mathbf{A}B'C'D'E'$ as their values are below θ.

Iteration 2: *The problem is re-solved in the second iteration, and 10 variables are removed and 5 variables are added. Observe that variable $\mathbf{AB}C'\mathbf{DE}$ is not added because the sum of its parent variables $\mathbf{AB}C'\mathbf{D}E'$, $\mathbf{AB}C'D'\mathbf{E}$, $A'\mathbf{B}C'\mathbf{DE}$ and $\mathbf{A}B'C'\mathbf{DE}$ (another parent A' $\mathbf{B}$ C' $\mathbf{DE}$ is not considered in this iteration) is below θ.*

Iteration 3: *The problem is re-solved in the third iteration; 1 additional variable is added and 7 variables are removed.*

Iteration 4: *The problem is re-solved once more; the algorithm converges since no additional variables are added.* □

Since we use only a subset of variables (*i.e.*, forcing the rest of the variables to be 0), sometimes, especially in early iterations, we cannot find a solution that satisfies all constraints. We do the following to guarantee that the problem is solvable.

1. Instead of using equality constraints, we use upper- and lower-bound constraints; that is, if the constraint has an equality form of $\cdots = p$, we change it to $p - \delta \leq \cdots \leq p + \delta$ (again, the problem can be solved by existing technology [4]). In the first iteration we start with a very small δ. While the problem is not solvable, we double δ and retry. In each of the later iterations, we start with the δ used in the previous iteration, and reduce it by half for the next round if we find feasible solutions without increasing it.
2. If after increasing δ several times and the problem is still unsolvable, we expand variables as described, without any pruning.
3. In the convergence round, we continue reducing δ by half and re-solving the problem until δ is close to 0.

EXAMPLE 4.4. *Reconsider Example 4.3. Observe that the solution generated by the first iteration in Table 2 does not add up to one, in fact the sum of the variables is equal to 1.02. A δ=0.03 was added to form the upper and lower bounds of the constraints since the problem was infeasible under the given constraints expressed with only nine variables. With δ=0.03 a feasible solution was generated in the first iteration, and the delta was reduced by half. The second iteration used δ=0.015, while the third iteration used δ=0.0075. At the end of the fourth iteration, we reduce δ by half and re-solve the problem until δ is close to zero.*

Algorithm OVERLAPESTIMATION (Algorithm 1) gives the framework for our algorithm. It proceeds in five steps.

1. First, generate the initial set of variables (Line 1).
2. Solve the maximum entropy problem accordingly. (Line 5).
3. If the problem is not solvable, double δ (Line 13) or expand the variables (Lines 9-11).
4. Otherwise, add new variables and remove variables with small values (Lines 16-20). If the problem is solvable without increasing δ, reduce δ by half (Lines 21-22).
5. Repeat Steps 2-4 until there is no new variable to expand and δ is reduced back to a small value (δ_0) (Line 4).

PROPOSITION 4.5. *Let s be the largest size of $\bar{V}$ in an iteration and $\theta_{infeasible}$ be the number of times we double δ before variable expansion.* OVERLAPESTIMATION *converges and solves the maximum entropy problem at most $\theta_{infeasible} \cdot |\Omega|$ times, each time with at most s variables and $|\mathbf{O}| + |\mathbf{C}| + 1$ constraints.* □

Algorithm 1: OVERLAPESTIMATION($\Omega, \mathbf{C}, \mathbf{O}$)

Input:
Ω */* Sources */* $\mathbf{C}, \mathbf{O}$ */* Existing statistics */*
Output:
$\bar{V}$ */* Values for the valid variables */*

```
1  V̄ ← InitiateVariables(Ω, C, O); ΔV̄ ← V̄;
2  δ = δ0; /* δ0 is a constant close to 0 */
3  n_infeasible = 0;/* number of allowed infeasible solutions */
4  while δ > δ0 OR ΔV̄ ≠ ∅ do
5      MaxEntropy(V̄, C, O, δ);//Solve the maximum entropy
       problem with variables in V̄
6      if infeasible solution then
7          n_infeasible + +;
8          if n_infeasible > θ_infeasible then
9              ΔV̄ ← ExpandVars(V̄, ΔV̄, C, O);
10             V̄ ← V̄ ∪ ΔV̄;
11             n_infeasible ← 0;
12         else
13             δ ← δ * 2;
14         end
15     else
16         if ΔV̄ ≠ ∅ then
17             ΔV̄ ← ExpandVars(V̄, ΔV̄, C, O);
18             RemoveVars(V̄);
19             V̄ ← V̄ ∪ ΔV̄;
20         end
21         if n_infeasible = 0 then
22             δ ← δ/2;
23         end
24         n_infeasible = 0;
25     end
26 end
```

Table 3: Number of variables in MaxEnt, removed and added in each iteration of the scalable solution for Example 3.7.

	# Vars	# Vars Removed	# Vars Added
Iteration 1	9	2	14
Iteration 2	21	10	5
Iteration 3	16	7	1
Iteration 4	10	1	0

EXAMPLE 4.6. *Consider applying* OVERLAPESTIMATION *to Example 3.7. Table 3 shows the number of inserted variables and deleted variables in each iteration when $\theta = .03$ (details in Table 2). We observe that (1)* OVERLAP *considers at most 21 variables in each iteration rather than all of the 32 variables (when we have more sources, the difference is typically much larger); and (2)* OVERLAPESTIMATION *obtains fairly similar results to the basic solution–the average absolute difference between the basic solution (Table 1) and the scalable solution (Table 2) is 0.028.* □

As shown in our experiments (Section 7.3), our overlap estimation algorithm significantly improves efficiency and scalability, while still being able to find a good ordering of the sources.

5. SOURCE ORDERING

With the estimated overlap information, we can order the sources for obtaining the largest area-under-the-curve. This section describes two ordering schemes: *static ordering* orders the sources once upfront; *dynamic ordering* collects new statistics along with query answering and adjusts the ordering accordingly.

Algorithm 2: STATIC($\mathcal{S}, \mathbf{O}, \mathbf{C}, Q$)

Input:
Ω /* *Sources* */ $\mathbf{C}, \mathbf{O}$ /* *Existing statistics* */
Q /* *Query* */
Output:
$\bar{A} = Q(\Omega)$ /* *Answer tuples* */

1 $\bar{V} \leftarrow$ OVERLAPESTIMATION($\Omega, \mathbf{C}, \mathbf{O}$);
2 $\vec{S} \leftarrow \emptyset$;
3 **while** $|\vec{S}| < |\Omega|$ **do**
4 Add SOURCESELECTION($\Omega, \vec{S}, \bar{V}$) to $\vec{S}$; /* *Choose the source with the largest ratio of residual contribution over cost* */
5 **end**
6 Answer Q in the ordering of $\vec{S}$ and return answers;

Algorithm 3: SOURCESELECTION($\Omega, \widehat{S}, \bar{V}$)

Input:
Ω /* *Sources* */ $\widehat{S}$ /* *Already selected sources* */
$\bar{V}$ /* *Estimated variable values* */
Output:
S /* *Next source to query* */

1 $RC[S] \leftarrow 0$ for each $S \in \Omega \setminus \widehat{S}$;
2 **foreach** $V_{\widehat{S}'} \in \bar{V}$ **do**
3 **if** $\widehat{S} \cap \widehat{S}' = \emptyset$ **then**
4 **foreach** $S \in \widehat{S}'$ **do**
5 $RC[S] \leftarrow RC[S] + V_{\widehat{S}'}$;
6 **end**
7 **end**
8 **end**
9 **return** $\arg\max_S \frac{RC[S]}{Cost[S]}$;

5.1 Static ordering

Once we know all overlap information, we can order the sources greedily: start with the source with the highest ratio of coverage over cost, and subsequently, select the source with the largest ratio of residual contribution over cost to query next. Algorithm STATIC gives the details. The key in this process is to compute residual contribution efficiently, which we describe next.

Let $\widehat{S}$ be the already queried sources and S be a remaining source. Recall that the residual contribution of S w.r.t. $\widehat{S}$ is defined as $P_Q(S \setminus \widehat{S}) = |Q(\widehat{S} \cup \{S\})| - |Q(\widehat{S})|$ and can be computed from the overlap information by Eq.(1). A naive evaluation of this equation would access $2^{|\widehat{S}|}$ overlaps and take exponential time. In addition, computing overlaps from $\mathbf{V}(\Omega)$ is also expensive. However, we can actually compute residual contributions directly from $\bar{V}$ (recall that $\bar{V} \subseteq \mathbf{V}(\Omega)$ contains variables with non-zero estimated value) by adding up the variables where the considered source S is positively expressed and every source in $\widehat{S}$ is negatively expressed.

$$P_Q(S \setminus \widehat{S}) = \sum_{V_{\widehat{S}'} \in \bar{V}, \widehat{S} \cap \widehat{S}' = \emptyset, S \in \widehat{S}'} V_{\widehat{S}'}. \quad (2)$$

Algorithm SOURCESELECTION finds the next source to probe according to $\bar{V}$. For each variable that expresses every source in $\widehat{S}$ negatively, the algorithm adds its (estimated) value to the residual contribution of each of its positively expressed sources (Lines 3-7). As a result, its running time is $O(|\bar{V}| \cdot |\Omega|)$. As we noted previously, typically the set of $|\bar{V}|$ is small, so we can compute each residual contribution very efficiently.

Table 4: Solution for dynamic ordering (solution shown for variables whose estimates are above θ).

Variables	Src A	Src C $P_Q(\cup\{A,C\}) = .7$	Src D $P_Q(\cup\{A,C,D\}) = .83$
A' ***B*** C' D' E'	.04	.06	-
A' B' ***C*** D' E'	.23	.19	.23
A' B' C' ***D*** E'	.13	.10	.13
A' B' C' D' ***E***	.03	.10	.04
A ***B*** C' D' E'	.27	.27	.27
A B' C' ***D*** E'	.17	.17	.17
A ***B*** ***C*** ***D*** E'	.03	.03	.03
A' ***B*** C' D' ***E***	.06	-	.08
A' ***B*** ***C*** ***D*** E'	-	.04	.04
A' ***B*** ***C*** ***D*** ***E***	.04	.03	-

PROPOSITION 5.1. *Let f_{est} be the complexity for overlap estimation. Algorithm* STATIC *takes time $O(f_{est} + |\bar{V}| \cdot |\Omega|^2)$.* □

EXAMPLE 5.2. *Consider selecting sources according to the estimated overlaps in Example 4.3. We first select source A, as it has the largest coverage (recall that we assume unit cost of all sources). The residual contribution of the other sources, given that source A has been probed, is computed as follows:*
$P_Q(B \setminus A) = A'\mathbf{B}C'D'E' + A'\mathbf{BC}\,\mathbf{DE} + A'\mathbf{B}\,C'\,D'\mathbf{E} = 0.14$
$P_Q(C \setminus A) = A'B'\mathbf{C}D'E' + A'\mathbf{BC}\,\mathbf{DE} = 0.27$
$P_Q(D \setminus A) = A'B'C'\mathbf{D}E' + A'\mathbf{BCDE} = 0.17$
$P_Q(E \setminus A) = A'B'C'D'\mathbf{E} + A'\mathbf{BCDE} + A'\,\mathbf{B}C'D'\mathbf{E} = .13$

Among them, C has the largest residual contribution and we select it to query next. Following this approach, we generate the ordering $L_s : ACDBE$. Note however that this ordering is different from the optimal ordering $L_d : ACDEB$. □

5.2 Dynamic ordering

As Example 5.2 shows, the statistics provided as input may be imprecise and incomplete and can often lead to sub-optimal ordering. When we query a source, we can collect some more precise statistics with very little additional cost. Such statistics include the coverage of the source and the union of all queried sources. DYNAMIC ordering improves over STATIC ordering by incorporating such new statistics to re-estimate the overlaps and the new results may improve the ordering of the unprobed sources.

EXAMPLE 5.3. *Continue with Example 5.2. Recall that the ordering according to the given statistics is $ACDBE$ while the optimal ordering is $ACDEB$. This is because none of the given statistics indicates that E is disjoint from A, B and D while B overlaps with D significantly.*

In DYNAMIC *ordering, we still first query sources A and C. At this point, we collect additional information $P_Q(\cup\{A,C\}) = .7$ (the union of A and C). As shown in Table 4, this additional statistic changes our estimates of the variables; however, D still has the highest residual contribution and is selected to query next. After probing D, we know in addition $P_Q(\cup\{A,C,D\}) = 0.83$. Note that according to estimates in the first round, we would compute that $P_Q(\cup\{A,C\}) = .74$ and $P_Q(\cup\{A,C,D\}) = 0.87$.*

With the knowledge that each of the two unions is actually smaller, we would infer a much lower $A'\mathbf{B}C'D'E'$ (dropping from the original .04 to 0) and a slightly higher $A'B'C'D'\mathbf{E}$ (increasing from the original .03 to .04). At this point, we compute the residual

Algorithm 4: DYNAMIC($\Omega, \mathbf{O}, \mathbf{C}, Q, N$)

Input:
Ω /* *Sources* */ $\mathbf{C}, \mathbf{O}$ /* *Existing statistics* */
Q /* *Query* */ N /* *Expected number of answers* */
Output:
$\bar{A} = Q(\Omega)$ /* *Answer tuples* */

1 $\overline{S} \leftarrow \oslash$; $\bar{A} \leftarrow \oslash$;
2 $S \leftarrow \arg\max_{S \in \Omega} \frac{P_Q(S)}{Cost(S)}$;
3 **while** $|\overline{S}| < |\Omega|$ **do**
4 $\quad \bar{A} \leftarrow \bar{A} \cup Q(S)$;
5 $\quad$ Update $\mathbf{C}$ with $P_Q(S) = \frac{|Q(S)|}{N}$;
6 $\quad$ Add $P_Q(\widehat{S} \cup S) = \frac{|\bar{A}|}{N}$ to $\mathbf{O}$;
7 $\quad \bar{V} \leftarrow$ OVERLAPESTIMATION($\Omega, \mathbf{C}, \mathbf{O}$);
8 $\quad S \leftarrow$ SOURCESELECTION($\Omega, \overline{S}, \bar{V}$);
9 **end**

contribution of B as 0.08 and that of E as 0.12, so decide to query E next instead of B. This results with the optimal ordering $L_d : ACDEB$. □

Algorithm DYNAMIC proceeds in four steps.

1. Select the source with the highest ratio of coverage over cost to query first (Line 2).
2. As the source is queried, output new answers, compute its coverage and the union of this source with already probed other sources, and update the statistics (Lines 4-6).
3. Re-estimate overlaps and then select the source with the largest residual contribution to query next by invoking SOURCESELECTION (Lines 7-8).
4. Repeat Steps 2-3 until all sources are queried (Line 3).

Note that originally $\mathbf{O}$ contains only overlap information. As sources are queried, the newly collected union information is added to $\mathbf{O}$; however, the framework OVERLAPESTIMATION is also applicable when $\mathbf{O}$ contains union information. Also note that the coverage and union are estimated against the expected number of answers, denoted by N. This number is taken as an input; it can be estimated according to either a-priori domain knowledge or the given statistics. Finally, note that Algorithm STATIC has similar structure as DYNAMIC except that it skips Lines 5-6 in DYNAMIC and estimates overlap only once upfront.

PROPOSITION 5.4. *Let f_{est} be the complexity for overlap estimation. Algorithm* DYNAMIC *takes time $O(f_{est} \cdot |\bar{V}| \cdot |\Omega|^2)$.* □

Although statistics collection and overlap estimation when we query each source introduces overhead, as shown in our experiments, the extra statistics can significantly improve source ordering and improve the rate of answer returning.

6. STATISTICS ENRICHMENT

As was shown, collecting additional statistics from each newly queried source often leads to better orderings. Thus, it may be desirable to collect more unknown statistics for even better ordering. We can compute the overlap between already probed sources or compute overlap for unprobed sources via sampling. If possible, we can also use statistics from some third parties; an example of third-party servers is *data markets* that provide profiles of data sources. Our experiments show that even statistics only from queried sources can already improve source ordering, and more statistics about unqueried sources can further improve the ordering.

In this process we need to deal with resource restrictions: (1) computing overlaps and retrieving statistics can take time (both from our end and from third-parties) and thus will increase query-answering time; (2) third parties may charge for each required statistic. Thus, we wish to carefully select only a small portion of statistics for enrichment.

EXAMPLE 6.1. *Re-consider the sources in Figure 1. To illustrate the benefit of statistics enrichment, we assume that we know only source coverage and one overlap statistic, $P_Q(\cap\{A, D\})$. The dynamic ordering according to this input is $ABDCE$, which is sub-optimal.*

Although we can collect information for all remaining overlaps ($2^5 - 1 - 1 = 30$) and accordingly obtain the optimal ordering, the overhead can be significant. Indeed, just knowing in addition the statistic $P_Q(\cap\{A, B\})$ can already lead to an optimal ordering if we apply DYNAMIC. *On the other hand, the additional statistics for $P_Q(\cap\{A, E\})$ would generate a dynamic ordering $ACBDE$, which cannot improve the ordering much. This shows the importance of statistics selection.* □

This section first describes how we enhance the static ordering and dynamic ordering with enriched statistics; we call the resulting algorithms STATIC+ and DYNAMIC+ respectively (Section 6.1). Then, it describes how we select the set of statistics with the highest impact to enrich (Section 6.2).

6.1 Static+ and Dynamic+

Figure 4 shows the control-flow diagram of static ordering with statistics enrichment, which we call STATIC+. We started with overlap estimation and then select a set of statistics to enrich. With the new statistics, we re-do overlap estimation and apply static ordering, essentially running STATIC.

Figure 5 shows the control-flow diagram of DYNAMIC+, dynamic ordering with statistics enrichment. Dynamic ordering has the benefit that instead of deciding the ordering upfront, each time it selects only one source to query next. This makes it possible to parallelize source querying and statistics enrichment. DYNAMIC+ uses two threads. Thread 1 selects a new source, queries it, and collects statistics from it, essentially the same as DYNAMIC. Meanwhile, Thread 2 iteratively selects an unknown overlap and obtains information for it. Once Thread 1 finishes, Thread 2 terminates.

In summary, Figure 6 shows the relationships of the four ordering algorithms we have presented.

6.2 Statistics selection

To save both time and cost for statistics enrichment, we assume we select at most K statistics, so we need to wisely select the statistics to enrich. We focus on overlaps, since other statistics such as unions can be inferred from coverage and overlaps. Intuitively, we wish to enrich the overlap that would change the current estimated variables most. First, such overlaps are likely to cover a lot of variables; for example, the overlap of sources A and B covers many more variables than that of sources $A, B, \ldots, E$, and so can affect the estimation of more variables. Second, such overlaps are likely to be fairly big so the real value for a variable it covers can be quite different from the estimate following the Maximum Entropy principle (assuming no extra correlation). For example, if we know $P_Q(\cap\{A, B\}) = .8$ and $P_Q(\cap\{A, C\}) = .01$, the estimated overlap of sources A, B, D (at most .8) can be more different from the real one than that of A, C, D (at most .01), so overlap $P_Q(\cap\{A, B\})$ has a higher impact.

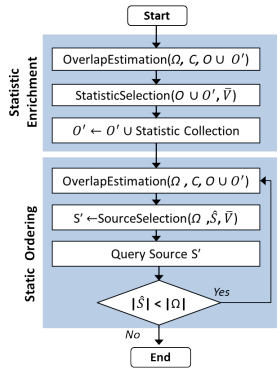

Figure 4: Control-flow diagram for STATIC+. Initially $O' = \emptyset$.

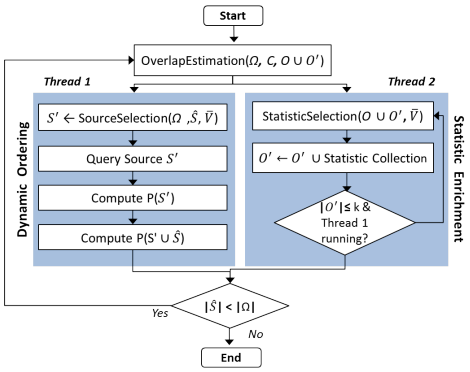

Figure 5: Control-flow diagram for DYNAMIC+. Initially $O' = \emptyset$.

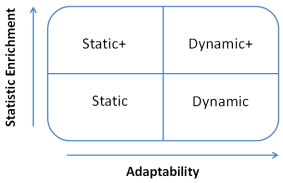

Figure 6: Comparison of algorithms.

Table 5: Statistics for data sets.

	Book I		*Book II*			
	Cov	2-Way Overlap	Cov	2-Way Overlap	Conn. Cost	Per-tuple Cost
Avg.	0.021	0.123	0.138	0.323	478	0.035
Stdev.	0.005	0.036	0.018	0.042	187	0.041
Min	0.001	0	0.001	0	291	0.005
Max	0.562	0.665	0.562	0.886	834	0.149

When we solve the Maximum Entropy problem, we can compute the *sensitivity* of a variable with little additional cost. Sensitivity is defined as the amount of change allowed in the variable to obtain the same value of the objective function; thus, the variables whose values are less certain are likely to have higher sensitivity and those with more certain values are likely to have lower sensitivity. Consistently, an overlap with higher accumulated sensitivity tends to cover more variables and the associated estimated value is often big. Our empirical study shows that such overlaps can have higher impact and our knowledge of them can lead to better ordering.

Specifically, we select statistics in three steps. First, we start with 2-way overlaps and compute the sensitivity of an overlap as the accumulated sensitivity from its contained variables in $\bar{V}$. Second, for each overlap of $\widehat{S}$ we iteratively compute sensitivity for $\widehat{S}$'s supersets; here we use a threshold θ_{sen}, which indicates large impact, and skip further traversal once the sensitivity for an overlap is below θ_{sen}. Finally, we order the overlaps first by sensitivity and then by the estimated value, and select the first up to K statistics whose sensitivity is above θ_{sen}.

7. EXPERIMENTAL RESULTS

We proceed with an experimental study of **OASIS** showing that it can significantly speed up query answering in terms of increasing the area-under-the-curve and reducing the time of getting a large percentage (*e.g.*, 90%) of answers.

7.1 Experiment setup

Data: We experimented on two data sets with different characteristics: one with more sources and higher overlaps than the other. The first data set, referred to as *Book I*, was obtained by crawling computer-science books from *AbeBooks.com*, an online aggregator of bookstores. It contains 1028 bookstores (sources), 1256 unique books, and 25347 book records in total. Each source provides from 1 to 630 books, and on average 211 books; 85% of the sources provide fewer than 5% books. We observed that the mean connection time to a website listed in *AbeBooks* is 356 ms and that of transmission of one tuple is 0.3 ms, so we set them as the fixed costs across all sources for this data set to focus the experiments on overlap estimation and source ordering.

The second data set, referred to as *Book II*, is a larger data set derived from *Book I*. It contains all sources in *Book I*. In addition, we randomly selected one third of the data sources for replication once, twice or three times to create new data sources. When we create a new source, we randomly decided a probability $0 < p < 1$ for each source; each record in the original data source was either copied as is to the new source with probability p, or replaced by a randomly selected book record in the data set. This data set contains 2150 data sources, 1256 unique books, and 58232 book records in total. For each source we randomly chose a connection cost in $[250, 850]$ ms, and a per-tuple transfer cost in $[.005, .15]$ ms. Note that *Book II* considers sources with faster per-tuple transfer than *Book I*, as per-tuple transfer cost is likely to be reduced substantially with network speedup without significantly affecting the mean connection time to a website. Table 5 gives statistics for the two data sets.

We computed the coverage of each source in the two data sets and also the overlap between each subset of up to 10 sources. Since in practice we seldom have precise information for coverage and overlap, we perturbed each statistic randomly by increasing or decreasing it by 10-50%. By default we randomly selected 250 overlaps as input; we may also range the number from 20 to 500. In dynamic ordering, we used precise coverage and union information for statistics collected from queried sources.

When simulating query answering, we perturbed querying cost randomly by 10-50% for unexpected connection delay and imprecise knowledge. By default we assume existence of a third-party statistics server from which we can inquire overlaps between a subset of sources; we also experimented for cases when such servers do not exist. We used perturbed overlaps as statistics returned by such a server. By default we assume it takes 250 milliseconds (ms) to compute one overlap from queried sources or obtain it from the server; we also ranged this number from 50 to 500 ms.

Implementations: We implemented seven ordering algorithms.

- RANDOM: Randomly choose an order of the sources.
- COVERAGE: Order the sources in decreasing coverage order.
- BASELINE: As a very simple way of considering overlaps in source ordering, order the sources by their residual contribution computed as follows.: Let $\hat{S}$ be the set of selected sources and S be a new source. Among the given overlaps with S and at least one source in $\hat{S}$, choose the highest overlap and subtract it from $P_Q(S)$ as S's residual contribution; i.e., $P_Q(S \backslash \hat{S}) = P_Q(S) - \max_{P_Q(\hat{S}') \in \mathbf{O}, \hat{S} \cap \hat{S}' \neq \emptyset, S \in \hat{S}'} P_Q(\hat{S}')$.
- STATIC: Algorithm STATIC.
- DYNAMIC: Algorithm DYNAMIC.

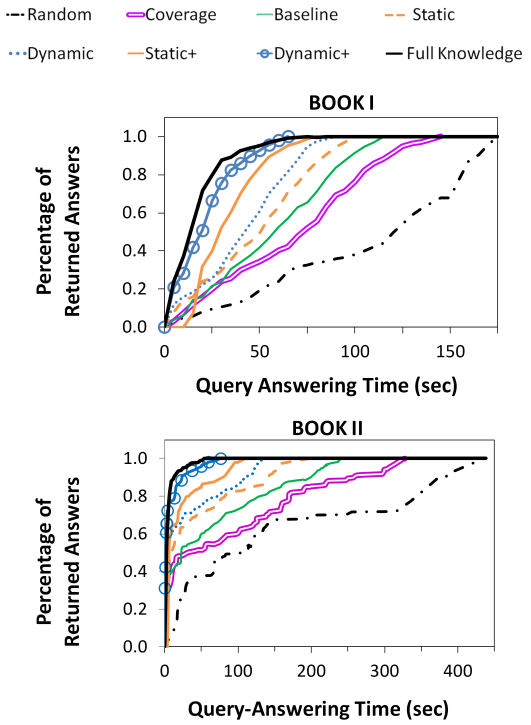

Figure 7: Area-under-the-curve for *Book I* and *Book II*.

- STATIC+: Algorithm STATIC+.
- DYNAMIC+: Algorithm DYNAMIC+.
- FULLKNOWLEDGE: Apply STATIC with precise information for all coverage and overlaps as input. Its performance can be considered as the upper bound we can achieve although it may not obtain the optimal ordering (Theorem 3.6).

In overlap estimation, we set $\theta = \frac{1}{1256} = .0008, \theta_{infeasible} = 5$, $\delta_0 = 0.05$. In DYNAMIC, we assumed knowledge of the total number of answers. In statistics selection, we set $\theta_{sen} = 0.005$; by default we inquired up to $K = 250$ statistics, and we ranged K from 50 to 500.

We implemented all algorithms in Java 1.5, and ran our experiments on a Windows 7 machine with 2.40GHz Intel Core i3 CPU and 4GB of RAM. We used the LINDO non-linear solver [6], which is publicly available, for solving the MAXENT problem.

Measures: We used query `SELECT * FROM Bookstores` and evaluated the performance of our algorithms using two measures: (1) *Area-under-the-curve* is as defined in Section 3, with the only difference that the Y-axis is the percentage of the returned answers over all answers; (2) *Execution time* is the time for returning 90% of the answers. Since we have perturbed coverage and overlap statistics, we expect similar results for other queries. We measured CPU time as real execution time, including overlap estimation, source ordering, statistics selection, statistics inquiry, and retrieving answers from the sources.

7.2 Overall performance

Sequential querying: We first compared various methods under default settings when we query the sources sequentially. Figure 7 shows the percentage of answers returned by each method at each time point using a single thread for source querying on both data sets. We have the following observations.

First, both naive solutions are very slow. RANDOM was the slowest in returning results and had the smallest area-under-the-curve; it spent 2.7 minutes in returning 90% answers on *Book I* and 6.3 minutes on *Book II*. COVERAGE was slightly faster than RANDOM, by 38% and 46% for *Book I* and *Book II*, respectively.

Second, considering overlaps can speed up query answering. BASELINE, which considers overlaps in a naive way, was faster than COVERAGE by 21% and 29%. STATIC, which estimates overlaps following the *Maximum Entropy* principle, was faster than BASELINE despite the overhead from overlap estimation; its speed up is by 15% and 28%.

Third, DYNAMIC, which collects statistics about the queried sources and dynamically selects the next source to query, was faster than STATIC by 21% and 28% for *Book I* and *Book II*, respectively, showing that the extra statistics does help.

Fourth, STATIC+ was significantly faster than STATIC by 54% and 75%, while DYNAMIC+ was 56% and 81% faster compared with DYNAMIC.

Fifth, we observed that although STATIC+ spent 10 seconds upfront selecting statistics and inquiring them (so the curve is flat at the beginning), it quickly caught up after that and was even faster than DYNAMIC (by 13% and 38%), showing the benefit of obtaining a lot of critical statistics upfront. On the other hand, although DYNAMIC+ does not obtain all additional statistics upfront but inquires them along the way, it was still faster than STATIC+ (by 38% and 150% respectively) since (1) it uses another thread for statistics inquiry and has almost no overhead, and (2) it obtains a better ordering, as we shall discuss next.

Finally, DYNAMIC+ is the fastest, spending no more than 1.2 minutes in returning 90% answers on both data sets and was 1.6 and 9.9 times faster than RANDOM for *Book I* and *Book II*, respectively. Furthermore, its performance is also very close to FULLKNOWLEDGE, to obtain full coverage DYNAMIC+ is slower only by 8% and 29% for *Book I* and *Book II*, respectively.

To better understand the results, we also plotted the percentage of returned answers with respect to the number of queried sources on *Book I* (Figure 8). Our observations are consistent with the observations on Figure 7. In particular, STATIC queried 480 sources to obtain 90% of the answers, while DYNAMIC+ queried only 200 sources. Note that at the beginning DYNAMIC+ did not obtain a better ordering than STATIC+, because STATIC+ essentially has more statistics to start with; however, as DYNAMIC+ queried sources, it had more precise estimates of the overlaps and selected more critical statistics to inquire, so after querying 60 sources, it returned more answers than STATIC+. Indeed, DYNAMIC+ queried only 200 sources to obtain 90% of the answers, while FULLKNOWLEDGE queried 160 sources for this purpose.

Table 6: Time of returning 90% answers by DYNAMIC+.

M	1	2	3	4	5
Time (sec)	12.3	11.7	9.2	6.9	8.0

Parallel querying: Our algorithms can be easily adapted when we have multiple threads for querying and source selection. Figure 9 plots query-answering time for returning 90% answers when varying the number of threads from 1 to 12. In DYNAMIC+, we use $M = 1$ thread for statistics enrichment. We observe the same ordering of the various methods. In addition, we observe that (1) the more threads, the less execution time; (2) the speedup is not linear with respect to the number of threads, as we typically observe for parallel systems; and (3) the speedup flattens out when we have more than 5 threads. Indeed, when we have 5 threads, we can return 90% answers from thousands of sources in only a few seconds.

Table 6 shows the time of returning 90% answers by DYNAMIC+ when there are 12 threads and when we vary M from 1 to 5. We

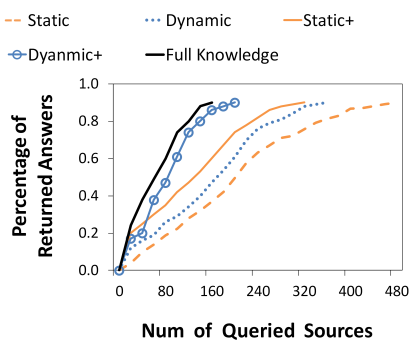

Figure 8: Returned answers vs. # sources queried (*Book I*).

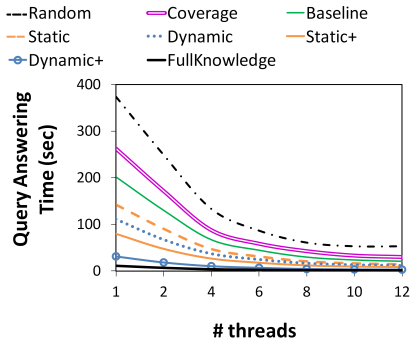

Figure 9: Parallel query answering (*Book II*).

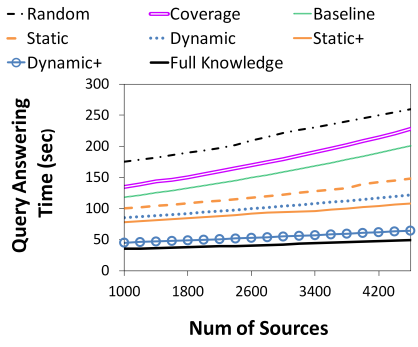

Figure 10: Scalability of various methods (*Book I*).

observe that increasing M from 1 to 4 would reduce execution time, since we get more statistics and so better ordering of the sources. Then continuing to increase M would increase execution time, showing that the benefit of obtaining a better ordering cannot exceed the loss of querying fewer sources at the same time.

Scalability: To evaluate scalability of the algorithms, we generated a pool of sources as follows. We first duplicated the sources three times and duplicated the books two times; we then created 2000 books and assigned them to the sources randomly. We randomly chose 1000 to 4100 sources from this pool for our experiments; as the number of sources increased, we also increased the statistics (#overlaps) linearly and randomly chose new statistics to add. Figure 10 reports the execution time for returning 90% answers when we query sources sequentially. We observed quite smooth linear growth of execution time for various methods; however, the growing rates are different. The rate is the highest for RANDOM, COVERAGE, BASELINE, on average 2.4 second per 100 sources; lower for STATIC, DYNAMIC, on average 1.5 second per 100 sources; and lowest for STATIC+, DYNAMIC+, only .005 second per 100 sources for DYNAMIC+. When the number of sources increased from 1000 to 4100, the execution time for DYNAMIC+ increased only from 45 seconds to 64 seconds.

We next examine various aspects of our algorithm using *Book I* assuming sequential query answering.

7.3 Evaluating overlap estimation

Scalable solution vs basic solution: We first compared Algorithm OVERLAPESTIMATION with the basic way of solving the MAXENT problem, which can involve $2^{|\Omega|}$ variables. Since the basic solutions cannot deal with a large number of sources, we randomly chose 30 sources from the data set, which would require 1 billion variables. Figure 11 plots the percentage of returned answers versus the number of queried sources for both methods. We observed that the basic solution did find a slightly better ordering of the sources, although the difference is not big. However, OVERLAPESTIMATION finished in milliseconds on 30 sources, whereas the basic solution spent three days. This experiment shows that our scalable solution for overlap estimation is much more efficient without sacrificing the quality of the ordering too much.

Effect of initial statistics: We next considered the effect of the number of initial overlaps on overlap estimation and source ordering. We varied the number of initial statistics from 20 to 500. Figure 12 shows the relative error between the true value and the estimated value of the variables, and Figure 13 shows the execution time for returning 90% answers when we applied DYNAMIC. (1) As we have more initial statistics, we do have better estimates, but the error curve flattens out after we reach 300 statistics. (2) On the other hand, the fewer errors do not speed up query answering much, if any, when we increase the number of overlaps from 20 to 200; after that the big overhead for overlap estimation slows down query answering. This experiment shows the importance of making a balance between accuracy of estimates and execution time.

7.4 Statistics enrichment

Statistics selection: To show the effectiveness of statistics-selection, we compared various statistics-selection strategies. We considered three options: 1) ordering first by sensitivity and then by estimated value; 2) ordering only by sensitivity; and 3) ordering only by estimated value. Recall that our algorithm adopts the first option. Figure 14 compares these three strategies with DYNAMIC when K ranges from 50 to 500. We have the following observations. (1) Considering both sensitivity and estimated value obtained the best results. (2) Considering sensitivity alone obtained better results than considering estimated size alone. (3) Even considering only one criterion can obtain better results than DYNAMIC, showing the benefit of enriched statistics. (4) As we increased K, the area-under-the-curve for DYNAMIC+ increased at the beginning, but flattened out when K reaches 425, as the additional statistics may not further improve the ordering. This experiment shows the effectiveness of our statistics-selection techniques.

Type of enriched statistics: We next examined the contribution of the chosen statistics. We compared three options: 1) BOTH obtains overlaps between already probed sources and overlaps involving unprobed sources (by inquiring third-party sources); 2) PROBED obtains only overlaps between probed sources; 3) UNPROBED obtains only overlaps involving unprobed sources. Figure 15 compares these three options with DYNAMIC when K is varied from 50 to 500. We have three observations. (1) PROBED improved the performance over DYNAMIC: when $k = 500$, the area-under-the-curve is increased by 20%. (2) UNPROBED has better performance than PROBED; this is not surprising because the former obtains overlaps revealing more information about unprobed sources, therefore it leads to better ordering among them. (3) BOTH improved over PROBED, but only slightly (by 8% when $K = 500$). This experiment shows that even if third-party servers do not exist, enriching statistics can improve performance; existence of such servers would further improve the performance.

Overhead of obtaining statistics: Finally, we examined the effect of the overhead of obtaining statistics on the performance of our algorithms. Figure 16 shows execution time for obtaining 90% of answers when we ranged statistics-requesting time from 50ms to 500ms. Obviously, as the overhead increases, the execution time also increases. The execution time of STATIC+ increased by 60%, with the much larger time for statistics inquiry. The execution time of DYNAMIC+ also increased, but only by 15%, since DYNAMIC+ parallelizes statistics enrichment with source querying; the longer

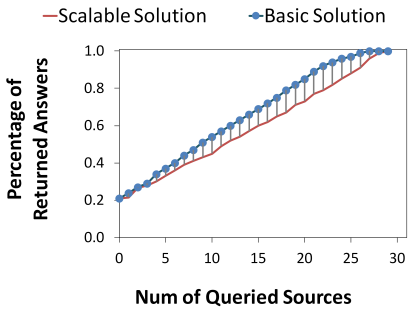

Figure 11: Basic vs. approximate estimation (*Book I*).

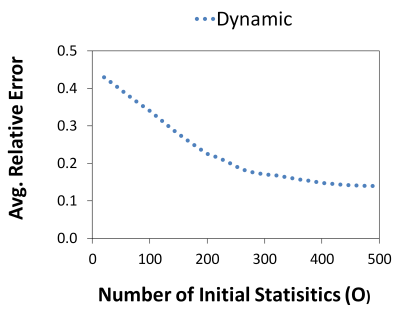

Figure 12: Average relative error vs. # statistics (*Book I*).

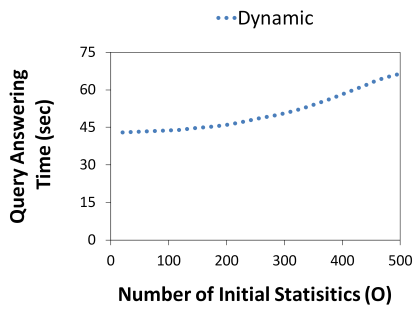

Figure 13: Query answering time vs. # statistics (*Book I*).

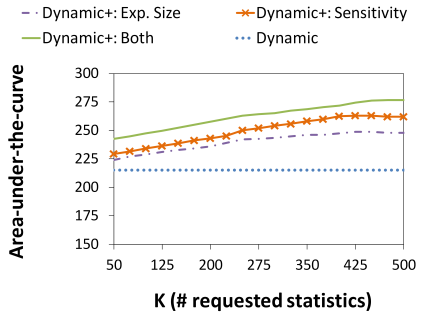

Figure 14: Statistic Selection.

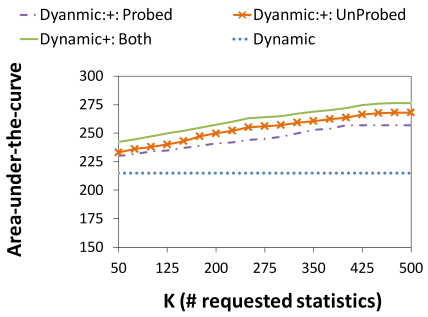

Figure 15: Probed vs. Unprobed Statistics.

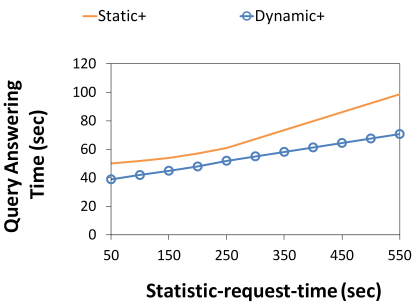

Figure 16: Statistic request time.

execution time is due to obtaining fewer statistics so the quality of ordering is lower. This experiment shows the robustness of DYNAMIC+ with respect to the overhead of statistics inquiry.

7.5 Summary

We summarize our experimental results as follows.

1. Our overlap-estimation algorithm is both scalable and effective; it can find good ordering of the sources to speed up query answering for sequential or semi-sequential querying.
2. DYNAMIC+ is the fastest among all ordering methods, has a low growth rate of execution time when we increase the number of sources and statistics, and is robust against the overhead of statistics inquiry.
3. It is important to make a balance between accuracy and execution time in overlap estimation.
4. Getting more statistics along with query answering and dynamically adjusting source ordering can indeed improve performance. However, having too many random statistics upfront may even slow down query answering because of the overhead for overlap estimation.
5. Our algorithm works well when we query sources in parallel.

8. CONCLUSIONS

We presented **OASIS**, an online query answering system whose core is an efficient and scalable algorithm that orders overlapping sources at runtime such that we can return answers to users fast in an online fashion. We empirically show the advantage of our algorithm over baseline source-ordering techniques on real data sets that contain thousands of sources. Future work includes combining our techniques with those that consider quality measures such as data accuracy and freshness in query answering to return answers both at a high speed and with high quality.

9. ACKNOWLEDGMENTS

This research was partially supported by NSF grants: IIS-0910859 and CNS-1305253.

10. REFERENCES

[1] J. Bleiholder, S. Khuller, F. Naumann, L. Raschid, and Y. Wu. Query planning in the presence of overlapping sources. In *EDBT*, pages 811–828, 2006.

[2] B. Chokshi, T. Hernandez, and S. Kambhampati. Relevance and overlap aware text collection selection. In *ASU TR 06-019*, 2006.

[3] N. Dalvi, A. Machanavajjhala, and B. Pang. An analysis of structured data on the web. *VLDB*, 5(7):680–691, Mar. 2012.

[4] M. Dudik, S. J. Phillips, and R. E. Schapire. Performance guarantees for regularized maximum entropy density estimation. In *the 17th Annual Conf. on Computational Learning Theory*, 2004.

[5] D. Florescu, D. Koller, and A. Y. Levy. Using probabilistic information in data integration. In *VLDB*, pages 216–225, 1997.

[6] Lindo systems: Optimization software. http://www.lindo.com/.

[7] X. Liu, X. L. Dong, B. C. Ooi, and D. Srivastava. Online data fusion. *PVLDB*, 4(11):932–943, 2011.

[8] Z. Nie, S. Kambhampati, and U. Nambiar. Effectively mining and using coverage and overlap statistics for data integration. *TKDE*, 17:638–651, 2005.

[9] C. Ré and D. Suciu. Understanding cardinality estimation using entropy maximization. In *PODS*, 2010.

[10] A. Roth. Completeness-driven query answering in peer data management systems. In *VLDB*, 2007.

[11] A. D. Sarma, X. L. Dong, and A. Halevy. Data integration with dependent sources. In *EDBT*, 2011.

[12] D. Srivastava and S. Venkatasubramanian. Information theory for data management. *PVLDB*, 2(2):1662–1663, 2009.

[13] B. Trushkowsky, T. Kraska, M. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, 2013.

[14] V. Vassalos and Y. Papakonstantinou. Using knowledge of redundancy for query optimization in mediators. In *AAAI Workshop on AI and Info. Integration*, 1998.