Data Integration with Dependent Sources

Anish Das Sarma
Yahoo Research
anishdas@yahoo-inc.com

Xin Luna Dong
AT&T Labs–Research
lunadong@research.att.com

Alon Halevy
Google Inc.
halevy@google.com

ABSTRACT

Data integration systems offer users a uniform interface to a set of data sources. Previous work has typically assumed that the data sources are independent of each other; however, in scenarios involving large numbers of sources, such as the Web or large enterprises, there is an *eco-system* of *dependent* sources, where some sources copy parts of their data from others.

This paper considers the new optimization problems that arise while answering queries over large number of dependent sources. These are the (1) *cost-minimization problem*: what is the minimum *cost* we must incur to get all answer tuples, (2) *maximum-coverage problem*: given a bound on the cost, how can we get the maximum possible coverage, and (3) the *source-ordering problem*: for a set of data sources, what is the *best* order to query them so as to retrieve answer tuples as fast as possible.

We consider these optimization problems under several cost models and we show that, in general, they are intractable. We describe effective approximation algorithms that enable us to solve these problems in practice. We then identify the causes of the high complexity and show that for restricted classes, the optimization problems can be solved in polynomial time.

1. INTRODUCTION

Data integration has received significant research attention and recently enjoyed commercial success [11, 12, 13]. Data integration systems offer users a uniform interface to a set of data sources. The user formulates a query over a mediated schema, and the system uses a set of semantic mappings to reformulate the query over the relevant set of data sources. The data integration system then combines the answers from the sources appropriately.

Data integration systems typically assume data sources are independent of each other. However, in scenarios involving large numbers of data sources, such as the Web or large enterprises, there is an *eco-system* of *dependent* sources, where some sources copy parts of their data from others [3]. The copying sources may be aggregators or have some data sharing agreement with the original source. A data integration system can benefit significantly from being aware of dependencies between its data sources. For instance, the system can save resources by not querying data sources that are unlikely to add many new answers to the query, or order the access to the sources to maximize the distinct answers it fetches early on. As another example, searching for "France Capital" on Google returns the answer "Paris" with a total number of data sources that corroborate this fact (without knowing whether mentions of these facts on the sources are independent); by considering dependencies between these data sources, a more authoritative result could be returned. This paper considers the new optimization problems that arise when answering queries over collections of dependent data sources.

EXAMPLE 1.1. *We extracted information about computer science books provided by searching* AbeBooks.com, *a listing-service website that integrates information from online bookstores. The collection includes data from 877 bookstores (sources), and by applying techniques in [6], we found copying between 465 pairs of sources. There were 314 copiers, and among them, 202 copy from a single source, 26 copy all tuples provided by the original sources, 30 copy over 90% of their data from other sources, and 100 copy at least half of their data from others.*

Now consider deciding the correct list of authors for each book. As different sources can provide conflicting information, along with each answer we would like to return the number of sources that support the answer, and take a vote. To avoid bias, we want to count only sources that independently support the answer and ignore the copied data. Hence, we can issue a query that returns all independently provided tuples.

When we answer such a query, we can ignore copiers that copy all data from others without changing the results; we can in addition ignore sources that copy most of their data, so further improve the efficiency without sacrificing the accuracy of the results much. In an online query answering system where we return answers as they are generated, we may wish to order the sources such that copiers with little independent contributions are queried last. □

1.1 The IDS System

We are building a system called IDS (Integrating Dependent Sources), for integrating a large number of data sources, where dependencies may exist between sources. We briefly describe the components in the architecture of the IDS system (depicted in Fig.1) and identify key problems we need to solve to build such a system.

Upon receiving a user query, IDS answers the query in three steps. First, the **Source Selection** component picks an *optimal* subset of sources to visit for returning all answers with the minimal cost, or the maximum number of answers with the given resource limit. This is because in an IDS system, it is often not necessary or feasible to visit all sources, as a subset of sources may have already covered all answers or we have only limited resources. Second, the **Source Ordering** component orders the sources (either

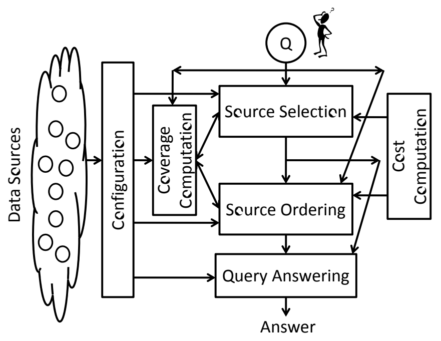

Figure 1: **Architecture for IDS (Integrating Dependent Sources).**

all sources or the ones returned by source selection) in a way such that the system can obtain answers most quickly. This step can be skipped in offline query answering, but would be critical in an online query answering scheme where we return answers to the user as they arrive from the sources, suitable for many applications such as vertical search. Third, the **Query Answering** component visits the selected sources in the specified order, takes the union of the answers, and returns to the users.

Three components are required for the above process. First, the **Coverage Computation** component computes the fraction of answers a set of sources cover for a particular query. Second, the **Cost Computation** component computes the cost of querying a set of sources, taking into consideration the number of sources in the set, the size of each source, connection cost, latency, and so on. Finally, the **Configuration** component identifies dependencies between data sources and is also responsible for traditional data integration configuration tasks such as schema mapping.

This paper addresses the following fundamental challenges that are needed to build the services described above, and lays the theoretical foundation for building the IDS system.

1. *Coverage:* what is the fraction of the overall set of answers that can be computed by a subset of sources.
2. *Cost minimization:* what is the minimal set of sources from which we can retrieve all the answers to a query.
3. *Maximum coverage:* given a resource bound, what is the set of sources for obtaining the maximum set of answers.
4. *Source ordering:* what is the best ordering of the data sources that provides more answers quickly.

1.2 Summary of Contributions

To address the above problems, we model dependency between sources with a directed acyclic graph whose nodes are the data sources. An edge from source S_1 to S_2 indicates that S_1 copies from S_2 (S_1 may have additional data that is not in S_2). We distinguish two versions of the underlying copy model: the first assumes that tuples are copied at random and applies when we have no information about how the sources are related; the second assumes more information such as a selection predicate over the copied source. We assume that dependencies are given as input. There are multiple ways for obtaining such information; for example, [6] shows how one can discover dependency between sources, decide the copying direction, and compute percentage of data that are copied, and provenance annotation may contain information such as selection predicates in copying.

The goal of query answering in this context is to find all *independent* answers. Roughly, this means we do not want to get the same answer from S_1 and S_2 if S_1 copied the data from S_2.

We begin by identifying the *coverage* problem as a core building block for the optimization problems. The coverage of a subset of sources $\mathcal{T}$ is the expected percentage of the overall set of independent answers that can be computed from $\mathcal{T}$. We establish the following results about coverage. In the case of the random-copying scenario, we show that coverage is, in general, #P-complete[1] in the total number of input data sources. We describe a randomized algorithm that yields an arbitrarily accurate estimate of the coverage in polynomial time, and we identify a subclass of the problem that gives an exact polynomial-time solution. In the scenario where we have more information on how tuples were copied, we show that the complexity of the coverage problem is lower.

We consider the *cost-minimization* problem and the *maximum-coverage* problem under multiple cost models, including the *linear cost model*, which counts the sizes of the accessed sources, and the *number-of-sources cost model*, which does not distinguish between the sources' sizes. We show that in the general case both problems are intractable under these cost models, and show that we can find an approximation in polynomial time. Moreover, we show that the number of data sources that a source can copy from and the ability to copy a fraction of the data are critical to the complexity of the above problems. In fact, if each data source can copy either all or no tuples and from at most one other data source, both problems are PTIME for the number-of-sources model. For the *source-ordering* problem we show that there is an efficient 2-approximation of the optimal ordering.

Note that although our primary motivation for this work was data integration, the problems and techniques we study are also relevant for other applications, such as query answering over partially-replicated data [1, 19]. Partial replication among data sources can be captured using dependencies, and we are interested in finding sets of sources that provide required portion of the data.

For the rest of the paper, Section 2 formally defines the problem. Section 3 studies the coverage problem. Section 4 considers the cost-minimization and maximum-coverage problems. Section 5 discusses the source-ordering problem. Section 6 shows how our results extend to more complex queries. Proofs for all results are presented in Appendix A.

2. PROBLEM DEFINITION

We begin by formally defining the dependency model and the optimization problems we consider. Consider a set of data sources $\mathcal{S} = \{S_1, \ldots, S_n\}$. We refer to the contents of sources as tuples, each modeling an object (*e.g.*, books, movies, job listings) and providing values for a set of attributes. The names of the attributes can vary from one source to another; we assume that we have already reconciled heterogeneity with schema-matching techniques [22] and query answering starts with query reformulation.

Dependency between sources: Our goal is to capture the fact that, in addition to having original data of their own, data sources often copy data from others. In general, a copier may copy data by performing a query over another source and adding the result of that query to its database; in practice, however, one may not know the queries used to copy data and can only estimate the fraction of tuples that are copied. We use the following *dependency DAG* to record the copying relationships between sources.

DEFINITION 2.1 (DEPENDENCY DAG). *The dependencies between the set of data sources $\mathcal{S} = \{S_1, \ldots, S_n\}$ are given by a DAG $G(\mathcal{S}) = (\mathbf{V}, \mathbf{E})$ where*

- *for every source $S_i \in \mathcal{S}$, there is a node in $\mathbf{V}$, associated*

[1]#P-completeness corresponds to the complexity class of hard counting problems [23].

with a number $n(S_i)$ specifying the number of tuples independently added by S_i[2]*, and*

- *a directed edge $S_i \rightarrow S_j$ denotes that S_i copies tuples from S_j, and the edge is associated with an "annotation" describing tuples copied by S_i from S_j.* □

There are at least three kinds of annotations for copying edges and accordingly we have three types of dependency graphs: (1) *Fraction-copying DAG*: the annotation on $S_i \rightarrow S_j$ is a fraction $f_{i,j}$ (called *selectivity*) denoting the fraction of tuples copied by S_i from S_j; (2) *Select-copying DAG*: the annotation on $S_i \rightarrow S_j$ is a select condition composed of predicates of the form $A\ op\ a$, where A is an attribute in S_j, a is a constant, and op is one of $=, <, \leq, >, \geq$, and all tuples in S_j satisfying the select condition are copied into S_i; (3) *Histogram-copying DAG*: the annotation on $S_i \rightarrow S_j$ is a histogram specifying the copying fraction for each range of possible attribute values. Results in this paper can be easily extended for a hybrid case with different types of annotations in the dependency graph, and also for the case where a copier copies by individual values rather than by tuples. Note that we leave projection in copying for future work, as estimating the size of projection results is known to be hard because of the duplicate-elimination problem [5].

When the graph has an edge $S_i \rightarrow S_j$, we refer to S_i as a *copier*. We say that S_i is a *full-copying* copier if it copies *all* data from the original sources whenever it copies anything. In this case, we call the dependency graph a *full-copying DAG*. We say that S_i is a *single-source* copier, if it is a full-copying copier and copies from a single source. We assume no-loop copying (common in practice) and copying direction has been given as input (from provenance information or by applying techniques in [6]), thus restrict ourselves to a DAG.

We assume that each tuple is annotated with the source from which it was copied, or marked as independently added. That is, *tuples* are of the form (t, S) where t is the tuple value, and S is the source that independently provided t. We assume sources are *sets* of tuples. Hence, even if (t, S) is obtained by copying from multiple sources, only one copy of the tuple is retained in the source.

The total number of tuples in a source S_i, denoted by $|S_i|$, can be estimated by its dependencies and independently added tuples; however, as we show shortly, this estimation is non-trivial.

EXAMPLE 2.2. *Fig.2(a) shows an example dependency graph for 6 sources. Among the sources, S_1 and S_2 each independently provides 100 tuples. S_3 and S_4 each copies 0.5 fraction of tuples from S_1 and also independently provides 50 tuples each. Sources S_5 and S_6 copy from multiple sources (S_5 from S_1 and S_2, and S_6 from S_2, S_3, and S_4) without independently providing data.* □

Query answering: For most of the paper we first present our solutions for one prototypical query: find all the tuples from the sources, denoted $Q(\mathcal{S})$. This prototypical query already unveils many challenges that arise in our context, and isolates the complexity of our problems from that of answering Q. In practice, the majority of queries tend to ask for all tuples that satisfy certain predicates. Section 6 describes an extension of our solutions for queries with select, project, join predicates.

We define the semantics of a query as the union of answers from all sources. Formally, given a source S_i, we denote by $Q(S_i)$ the set of answers from S_i (either independently added or copied). We define $Q(\mathcal{S})$ as $\cup_{S_i \in \mathcal{S}} Q(S_i)$, where $\cup$ is the set union of $Q(S_i)$. Recall that each tuple, (t, S), is annotated by the source that independently provides it. Hence, if tuple t is independently provided by S_1 and S_2, both (t, S_1) and (t, S_2) will be in the answer.

[2]Changed values are also considered as independently provided.

Our goal is to take advantage of the dependency between sources to compute $Q(\mathcal{S})$ efficiently. Hence, we try to answer the query (or get a nearly complete answer) from a subset of sources. We denote by $|Q(\mathcal{T})|$ the total number of answer tuples returned by a subset $\mathcal{T} \subseteq \mathcal{S}$ of sources. When Q is the identity query, we use $\langle \mathcal{T} \rangle$ and $|Q(\mathcal{T})|$ interchangeably.

Cost models: Given a set $\mathcal{T}$ of data sources, we consider the following variations on the cost model and show that they have subtle effects on the complexity results. Our results can be easily extended to the case when we need to combine the models (*e.g.*, querying each source incurs a constant connection cost and a cost proportional to the size of the data).

1. **Linear Cost Model (LCM):** We denote by $|S_i|$ the number of tuples in S_i. The cost of querying $\mathcal{T}$ is $c(\mathcal{T}) = \sum_{S_i \in \mathcal{T}} |S_i|$. This model applies when data are already stored locally and thus performing the union can be done in near-linear time in the size of the answers returned from each data source (and certainly in linear time in the number of I/Os) using either a hash table or an ordered index; it can also capture the bandwidth usage in case data are stored at each source.
2. **Number-of-Sources Cost Model (NSCM):** We denote by $|\mathcal{T}|$ the number of sources in $\mathcal{T}$ and $c(\mathcal{T}) = |\mathcal{T}|$. Such a model applies when the system is being charged for every query over any of the sources.
3. **Arbitrary Source Cost Model (ASCM):** Here we assume each source S_i is associated with an arbitrary cost c_i incurred in querying it. Hence, $c(\mathcal{T}) = \sum_{S_i \in \mathcal{T}} c_i$. This model applies when the system is charged on different sources differently.

Coverage: Given a subset of sources $\mathcal{T}$, we would like to define the *coverage* of $\mathcal{T}$ w.r.t. $\mathcal{S}$ as the expected value of the fraction of answers to $Q(\mathcal{S})$ that we can obtain from $\mathcal{T}$.

DEFINITION 2.3 (COVERAGE PROBLEM). *Given a set $\mathcal{S}$ of data sources, a dependency graph $G(\mathcal{S})$, and a subset $\mathcal{T} \subseteq \mathcal{S}$ of sources, compute the expected value of $\frac{|Q(\mathcal{T})|}{|Q(\mathcal{S})|}$.* □

In certain cases, such as when we know only copying fractions (so we have a fraction-copying DAG), $\frac{|Q(\mathcal{T})|}{|Q(\mathcal{S})|}$ cannot be uniquely determined by $G(S)$, and hence we are interested in obtaining the expected value of $\frac{|Q(\mathcal{T})|}{|Q(\mathcal{S})|}$. The coverage problem will play an important role in the other problems we consider in the paper.

Optimization problems: We now formally define our optimization problems, given a cost model c.

The *cost-minimization problem* tries to find a minimal set of sources that still yields all the answers to the query:

DEFINITION 2.4 (COST MINIMIZATION PROBLEM (CMP)). *Given a query Q, a set $\mathcal{S}$ of data sources, and a dependency graph $G(\mathcal{S})$, find a subset $\mathcal{T} \subseteq \mathcal{S}$ such that*

1. *$Q(\mathcal{T}) = Q(\mathcal{S})$;*
2. *for any $\mathcal{T}' \subseteq \mathcal{S}$, if $Q(\mathcal{T}') = Q(\mathcal{S})$, then $c(\mathcal{T}') \geq c(\mathcal{T})$.* □

The *maximum-coverage problem* tries to find the best answer (measured in number of tuples) that can be obtained with a fixed cost limit.

DEFINITION 2.5 (MAXIMUM COVERAGE PROBLEM (MCP)). *Given a query Q, a set $\mathcal{S}$ of data sources, a dependency graph $G(\mathcal{S})$, and allowed cost C_{max}, find a subset $\mathcal{T} \subseteq \mathcal{S}$ such that*

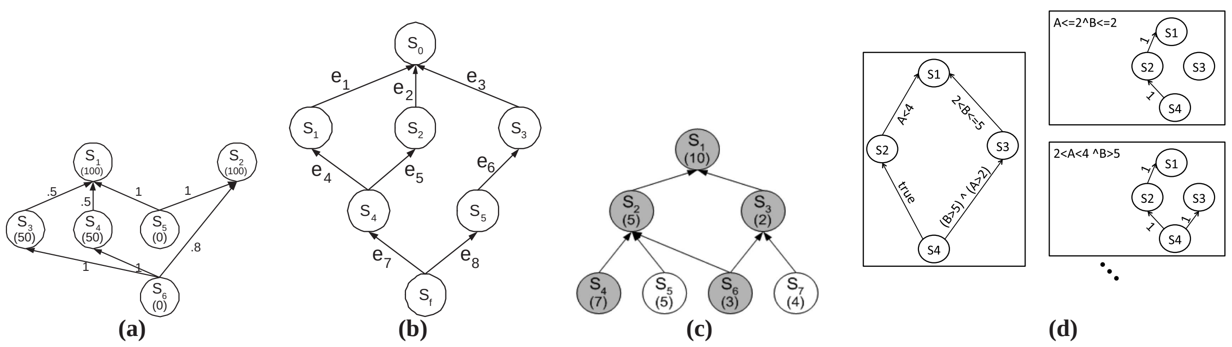

Figure 2: (a) An example dependency graph. Each node is marked with the source it represents and the number of independent tuples added by that source; each edge is marked with the fraction of data being copied. (b) An example of Limited Coverage Problem. (c) Input dependency graph in Example 3.8: each edge is associated with a fraction of 1 and so we omit the fractions; the marked nodes are those considered in computing coverage of $\{S_4, S_6\}$. (d) An example dependency graph with selection predicates on each node. The graphs on the right show two graphs constructed by restricting to specific combinations of attribute values.

1. *$c(\mathcal{T}) \leq C_{max}$;*
2. *for any $\mathcal{T}' \subseteq \mathcal{S}$, if $c(\mathcal{T}') \leq C_{max}$, then $|Q(\mathcal{T}')| \leq |Q(\mathcal{T})|$.* □

The *source-ordering problem* tries to find the optimal order of sources in which to execute the query, so we return query answers as quickly as possible. Intuitively, if we plot the curve of the number of tuples returned as we query more sources, we want to maximize the area under the curve. Formally, let Π be a permutation of the l data sources, where $\Pi(j)$ denotes the jth source in the permutation Π. We define the area below a curve that represents answering the query with respect to permutation Π as

$$A_Q(\Pi) = \sum_{i=1}^{l} c(\{S_{\Pi(i)}\}) \cdot |Q(\cup_{j=1}^{i}\{S_{\Pi(j)}\})|.$$

The source ordering problem can be defined as follows.

DEFINITION 2.6 (SOURCE ORDERING PROBLEM (SOP)). *Given a query Q, a set of sources $\mathcal{S} = \{S_1, \ldots, S_l\}$, a dependency graph $G(\mathcal{S})$, find a permutation Π_{opt} of $\{1, \ldots, l\}$ such that for any other permutation Π, we have $A_Q(\Pi_{opt}) \geq A_Q(\Pi)$.* □

EXAMPLE 2.7. *Consider the dependency graph in Fig.2(a). For the cost minimization problem, an optimal solution is $\{S_3, S_4, S_5\}$ w.r.t. the linear cost model (with cost 100 + 100 + 200=400). But this solution is not optimal w.r.t. the number-of-sources cost model, where the optimal solution is $\{S_5, S_6\}$ instead.*

Now assume we can query at most one source (the number-of-sources model) for the maximum coverage problem. Querying $\{S_6\}$ is the optimal solution, returning 255 tuples in expectation. Finally, w.r.t. the number-of-sources model the optimal permutation of sources is $S_6 \rightarrow S_5$ and then the rest of the sources (which do not add new tuples). □

3. THE COVERAGE PROBLEM

We begin by considering the coverage problem, which is fundamental to all of the other three problems. In Section 3.1, we consider the case when all we know is the fraction of tuples being copied between sources. In Section 3.2 we study the case when we know more about the specific set of tuples being copied, specified by a selection query or a histogram. Table 1 summarizes the results we establish in this section.

3.1 Copying a fraction of tuples

In practice we often do not have a-priori knowledge of which tuples are more likely to be copied by which sources. In such cases we cannot compute a precise coverage, since two copiers S_1 and S_2 may copy a fraction of the data of a source S_3, and we do not know the overlap between the data they copied. Hence, we *estimate* the coverage assuming each tuple is equally likely to be copied.

We establish three main results. First, we show that in general, the coverage problem is #P-complete in the number of sources (Section 3.1.1). Second, we show that a PTIME randomized algorithm yields arbitrarily good approximations of the coverage (Section 3.1.2). Finally, we show that the hardness of the problem comes from allowing a copier to copy only a fraction of the data from the original source (Section 3.1.3).

3.1.1 The limited coverage problem

Our results rely on identifying a limited version of the coverage problem and relating it to the computation of the probability of a boolean formula. In the limited-coverage problem we have a single *original source* that independently provides data, a set of sources that copy from the original source directly or transitively, and a single *sink* source that is not copied by any other source (see Fig.2(b) for an example).

DEFINITION 3.1 (LIMITED COVERAGE PROBLEM). *The limited coverage problem considers a set of data sources $\mathcal{S}$ with dependency graph $G(\mathcal{S}) = (\mathbf{V}, \mathbf{E})$ that satisfies the following properties:*

- *there exists just one source $S_0 \in \mathcal{S}$ whose node $v_0 \in \mathbf{V}$ has no outgoing edges,*
- *there exists just one source $S_f \in \mathcal{S}$ whose node $v_f \in V$ has no incoming edges, and*
- *only S_0 independently adds tuples.*

The limited coverage problem is to compute $\frac{|S_f|}{|S_0|}$ assuming equal probability of a tuple being copied, where $|S|$ is the number of tuples in S. □

We reduce the limited coverage problem to the problem of finding the probability of a boolean formula $\mathcal{F}$ in DNF form constructed as follows:

- There is a boolean variable in $\mathcal{F}$ for every edge in G. Each variable is independent of the others. For the variable corresponding to $e_{ij} = (v_i, v_j) \in \mathbf{E}$, its probability of being *true* is $f_{i,j}$, where $f_{i,j}$ is the fraction associated with the edge.
- For each distinct path from S_f to S_0 in G (there must exist such a path) consisting a sequence of edges $\{e_1, \ldots, e_k\}$, we add a conjunct $(e_1 \wedge \ldots \wedge e_k)$ to $\mathcal{F}$.

For Fig. 2(b), the coverage problem can be reduced to computing probability of the following DNF

$$(e_1 \wedge e_4 \wedge e_7) \vee (e_2 \wedge e_5 \wedge e_7) \vee (e_3 \wedge e_6 \wedge e_8)$$

Table 1: Summary of results for the coverage problem for various copy models. Let N be the number of nodes in the input dependency graph, E be the number of edges, k be the number of attributes on which selection predicates or histograms are present, b be the maximum number of constants in predicates for each attribute on each edge when selection predicates are present and the maximum number of buckets for each attribute on each edge when histograms are present, and $L = \frac{\log \frac{1}{\delta}}{\epsilon^2}$. For some copying models we consider two cases: attributes are independent or correlated.

	Fraction-copying	Full-copying	Select-copying	Histogram-copying
Exact Solution	#P-complete	$O(N+E)$	**Attr. Dep:** $O((2bE)^k(N+E))$ **Attr. Indep:** $O(bkE(N+E))$	#P-complete
(ϵ, δ)**-approx**	$O(LNE)$	N/A	N/A	**Attr. Dep:** $O((bE)^k LNE)$ **Attr. Indep:** $O(bkLNE^2)$

The probability of a boolean formula is defined as the sum of the probabilities of all its satisfying assignments. The following lemma provides the key result that we will use next by relating the coverage problem and the probability of $\mathcal{F}$.

LEMMA 3.2. *The probability of $\mathcal{F}$ constructed as described is equal to $\frac{|S_f|}{|S_0|}$.* □

The lemma below shows that even the limited-coverage problem is #P-hard. It is proved by a reduction from the #P-complete problem of counting the number of satisfying assignments in a bipartite monotone 2-DNF formula [21].

LEMMA 3.3. *The limited coverage problem is #P-hard.* □

Finally, we establish the #P-completeness of the general version of the coverage problem.

THEOREM 3.4. *Given a set $\mathcal{S}$ of data sources, a dependency graph $G(\mathcal{S})$, and a subset $\mathcal{T} \subseteq \mathcal{S}$, the associated coverage problem is #P-complete in the number of sources in $\mathcal{S}$.* □

3.1.2 Approximating coverage

We now show that we can approximate the coverage problem with a monte-carlo based algorithm. We will establish this claim in two steps. First, we show that we can give an arbitrarily accurate estimate for the limited coverage problem. We then show that we can solve the general coverage problem by solving a linear number of limited coverage problems.

To show the first part, we note that although the formula $\mathcal{F}$ constructed in Section 3.1.1 could be of size exponential in $|\mathcal{S}|$, we can apply the following randomized algorithm to compute the probability of $\mathcal{F}$ in time polynomial in $|\mathcal{S}|$.

```
0: Input: Dependency graph $G$ for the limited coverage problem.
   Output: Estimation of $\frac{|S_f|}{|S_0|}$.
1: Topologically sort the nodes in $G$: $S_0$ first and $S_f$ last;
2: Set $C_L = 0$. Repeat $L$ times:
3:        For each edge $e_{i,j}$, include it with probability $f_{i,j}$
          (and omit it with probability $(1 - f_{i,j})$);
4:        Decide in the topological order for each source if it is
          connected to $S_0$;
5:        if ($S_f$ is connected to $S_0$) $C_L$++.
6: return $\frac{C_L}{L}$.
```

Algorithm 1: LIMITEDCOVERAGE Randomized algorithm to solve the limited coverage problem.

Algorithm LIMITEDCOVERAGE (Algorithm 1) proceeds as follows. In every iteration, we adjust edges of the dependency graph. For each original edge with fraction $f_{i,j}$, we include the edge with probability $f_{i,j}$ and remove it otherwise. We count 1 if there exists a path from S_f to S_0, which can be decided in polynomial time.

This procedure is repeated L times and the following theorem shows that we can get an estimation that is arbitrarily close to the correct coverage in polynomial number of iterations. Specifically, for a given allowed error $\epsilon > 0$, we can ensure that the probability of the error exceeding ϵ is at most δ, when the number of iterations is more than $\frac{\log(\frac{1}{\delta})}{\epsilon^2}$. In other words, we can arbitrarily reduce the probability of exceeding a given error bound in polynomial number of iterations. Since each iteration is polynomial in the number of edges $E = |\mathbf{E}|$ of the graph, the total complexity is $O(\frac{E\log(\frac{1}{\delta})}{\epsilon^2})$.

THEOREM 3.5. *If $L > \frac{\log(\frac{1}{\delta})}{\epsilon^2}$ and the randomized algorithm satisfies $\mathcal{F}$ in C_L of the L iterations, then $\Pr(|\Pr(\mathcal{F}) - \frac{C_L}{L}| \geq \epsilon) \leq \delta$, where $\Pr(\mathcal{F})$ is the true probability of $\mathcal{F}$.* □

Next, we show how to solve the general coverage problem using the limited coverage problem. Consider the set $\mathcal{S}$ of sources and a subset $\mathcal{T} \subseteq \mathcal{S}$, and our goal is to estimate $\frac{\langle \mathcal{T} \rangle}{\langle \mathcal{S} \rangle}$. Algorithm COVERAGE first computes the coverage of $\mathcal{T}$ on tuples independently added by each source. Since the contribution of tuples to $\mathcal{T}$ by every source in $\mathcal{S}$ is independent of other sources in $\mathcal{S}$, these contributions are added to obtain the total coverage.

```
0: Input: $\mathcal{S}, \mathcal{T} \subseteq \mathcal{S}$, and dependency graph $G(\mathcal{S})$.
   Output: Estimation of $\frac{\langle \mathcal{T} \rangle}{\langle \mathcal{S} \rangle}$.
1: Set $D = \sum_{S_i \in \mathcal{S}} n(S_i)$ and $C = 0$;
2: foreach (source $S_i \in \mathcal{S}$ where $n(S_i) > 0$ and $S_i$ has at least one
   descendant in $\mathcal{T}$)
3:        Construct the subgraph $I_{G_i}$ of $G$ induced by vertexes
          $S_i$ and $\mathcal{T}$ as follows: with $S_i$ as the root, traverse
          child node to reach all possible descendants of $S_i$
          in $\mathcal{T}$. Add a special node $S_f$ in $I_{G_i}$ with edges to each
          descendant of $S_i$ in $\mathcal{T}$. Set the fractions associated with
          all these added edges to 1;
4:        Compute the coverage $c_i$ of $\{S_f\}$ in $I_{G_i}$ by invoking
          Algorithm LIMITEDCOVERAGE;
5:        $C = C + c_i * n(S_i)$;
6: return $\frac{C}{D}$.
```

Algorithm 2: COVERAGE Randomized algorithm to solve the coverage problem.

THEOREM 3.6. *Let N be the number of sources in $\mathcal{S}$, E be the number of edges in the input fraction-copying DAG $G(\mathcal{S})$, and $L = \frac{\log(\frac{1}{\delta})}{\epsilon^2}$. Algorithm* COVERAGE *(Algorithm 2) can give an arbitrarily accurate estimate for the coverage problem in time $O(LNE)$.* □

3.1.3 A PTIME subclass

Finally, we show that the high complexity of the coverage problem is due to the fact that each copier can copy only a fraction of data from an original source. Algorithm FULLCOPYINGCOVERAGE (illustrated in Ex. 3.8, and described in Algorithm 3) computes the exact coverage of a set of sources when they are all full-copiers: if they copy any data from a source, then they copy all of it.

```
 0: Input: $\mathcal{S}, \mathcal{T} \subseteq \mathcal{S}$, and dependency graph $G(\mathcal{S})$.
    Output: $\frac{\langle \mathcal{T} \rangle}{\langle \mathcal{S} \rangle}$.
 1: $n_s = 0; n_t = 0$;
 2: for each ($S \in \mathcal{S}$) $n_s$+=$n(S)$; //$n(S)$ is #(independent tuples) in $S$.
 3: for each ($T \in \mathcal{T}$)
 4:     $Q = \{T\}$; // the queue to traverse
 5:     while ($Q \neq \emptyset$)
 6:         $N = pop(Q)$;
 7:         if ($N$ is not visited yet)
 8:             $n_t$+=$n(N)$;
 9:             Mark $N$ as visited;
10:             Push $N$'s parents into $Q$;
11: return $n_t/n_s$;
```

Algorithm 3: Algorithm FULLCOPYINGCOVERAGE.

THEOREM 3.7. *Let N be the number of sources in $\mathcal{S}$, E be the number of edges in the input full-copying DAG $G(\mathcal{S})$. Algorithm* FULLCOPYINGCOVERAGE *solves the coverage problem in time $O(N + E)$.* □

EXAMPLE 3.8. *Consider the dependency graph shown in Fig.2(c). To compute the coverage of $\{S_4, S_6\}$, Algorithm* FULLCOPYINGCOVERAGE *traverses them and their ancestors in the order of S_4, S_2, S_1, S_6, S_3. The number of independently added tuples by these nodes is 27. The total number of independent tuples of all sources is 36. So the coverage is $27/36 = .75$.* □

3.2 Select copying

In this section we consider cases in which we have more information. We start with the case where we know the exact selection predicates applied in copying. We then extend our results for a hybrid case, where we have histograms describing the fraction of tuples copied for each bucket.

3.2.1 Conjunctive predicates

When we know the predicates used for copying, we have a select-copying DAG as the input. To solve the coverage problem, we transform the select-copying DAG to a set of *full-copying DAGs*, apply the PTIME algorithm FULLCOPYINGCOVERAGE on each result DAG and then aggregate the results. In the transformation, for each attribute we define a set of disjoint value ranges and each result DAG corresponds to a combination of value ranges for different attributes. We specify the SELECTCOPYINGCOVERAGE algorithm rigorously as follows.

1. Suppose we have predicates on k attributes $A_1, \ldots, A_k$. For each attribute $A_i, i \in [1, k]$, do the following. (1) Collect and order all constants that appear in the predicates $A_i\ op\ a$ on some edge; the results is $\{a_1, \ldots, a_{l_i}\}$, where l_i is the number of distinct constants for A_i and $\forall j \in [1, l_i), a_j < a_{j+1}$. (2) Consider all $2l_i + 1$ possible value ranges: $A_i < a_1$, $A_i = a_1$, $a_1 < A_i < a_2$, $A_i = a_2, \ldots, A_i > a_{l_i}$.
2. Create $P = \prod_{i=1}^{k}(2l_i+1)$ full-copying dependency graphs, $G_1, \ldots, G_P$, each corresponding to a combination of value ranges for the k attributes, denoted by $R(G_i), i \in [1, P]$. Do the following for each G_i. (1) For each edge e, if $R(G_i)$ satisfies the predicate for e in G, associate a fraction of 1; otherwise, remove e. (2) For each source S, update $n(S_i)$ as the number of independently added tuples in $R(G_i)$ (we assume such $n(S_i)$'s are given as input).
3. Solve the coverage problem for each G_i, $i \in [1, P]$, using Algorithm FULLCOPYINGCOVERAGE. Sum up the results as the coverage for G.

EXAMPLE 3.9. *Fig.2(d) shows an example dependency graph with selection predicates on two attributes A and B. Attribute A has two end points, 2 and 4, so has 5 possible ranges, $A < 2$, $A = 2$, $2 < A < 4$, $A = 4$, and $A > 4$. Similarly, there are 5 possible ranges for B: $B < 2$, $B = 2$, $2 < B < 5$, $B = 5$, and $B > 5$. Hence, there are a total of 25 combinations, giving 25 full-copying DAGs. Fig.2(d) shows two of them; in the full-copying DAGs, we have combined multiple ranges (such as $B < 2$ and $B = 2$ into $B \leq 2$).* □

The next theorem establishes complexity for the coverage problem over SELECTCOPY dependency graphs.

THEOREM 3.10. *Let N be the number of sources in $\mathcal{S}$, E be the number of edges in the select-copying DAG $G(\mathcal{S})$, k be the number of distinct attributes on which predicates are specified in $G(\mathcal{S})$, and b be the maximum number of constants in predicates for each attribute on each edge. Algorithm* SELECTCOPYINGCOVERAGE *solves the coverage problem in time $O((2bE)^k(N + E))$.* □

If we know that the attributes are independent of each other, then the complexity is significantly reduced, and we can show the following result.

COROLLARY 3.11. *Let N be the number of sources in $\mathcal{S}$, E be the number of edges in the select-copying DAG $G(\mathcal{S})$, and k be the number of distinct attributes on which predicates are specified in $G(\mathcal{S})$. When for each source the attributes are independent in value distribution, the coverage problem can be solved in time $O(bkE(N + E))$.* □

3.2.2 Histograms

In a histogram-copying DAG, each edge is annotated with a histogram that specifies the copy fraction for ranges of possible attribute values, and we assume uniform copying within each bucket. Note that when the attributes are correlated, we need k-dimensional histograms. We can proceed as in Algorithm SELECTCOPYINGCOVERAGE, except that each dependency graph we construct is a fraction-copying DAG and computing its coverage is #P-hard.

THEOREM 3.12. *Let N be the number of sources in $\mathcal{S}$, E be the number of edges in the histogram-copying DAG $G(\mathcal{S})$, k be the number of distinct attributes on which histograms are specified in $G(\mathcal{S})$, and b be the maximum number of buckets for each attribute on each edge.*

- *The coverage problem is #P-complete.*
- *We can get an ϵ-approximation with confidence $(1-\delta)$ (in the sense of Theorem 3.5) in the coverage in time $O((bE)^k LNE)$ in general, and in $O(bkLNE^2)$ when the attributes are independent, where $L = \frac{\log(\frac{1}{\delta})}{\epsilon^2}$.* □

4. MCP AND CMP

We now consider the closely-related maximum-coverage and cost-minimization problems. We begin by showing that in general both problems are intractable w.r.t. each of the cost models we defined previously (Section 4.1). In Section 4.2 we show that we can approximately solve both problems using a greedy algorithm. Finally, in Section 4.3 we identify copy patterns that are common in practice, under which we can exactly solve the problems with respect

Table 2: Summary of complexity of (1) cost-minimization problem, (2) maximum-coverage problem, and (3) source-ordering problem. The results apply to all cost models, unless otherwise specified. The approximation takes polynomial time.

	Cost Minimization	Maximum Coverage	Source Ordering
Non-full-copying	NP-complete, MaxSNP-hard	PP-hard	PP-hard
Full-copying	NP-complete, MaxSNP-hard	NP-complete	in NP
Single-Source Copying	PTIME	PTIME[a], NP-complete[b]	PTIME
Approximation	$\log \alpha$-approx[c]	$(1-\frac{1}{e})$-approx[c]	2-approx[c]

a	For NSCM cost model
b	For LCM or ASCM cost model
c	With PTIME coverage algorithm

to certain cost models in polynomial time. The results of this section are summarized in Table 2[3]. Note that the results apply to all copying models (fraction-copying, select-copying, histograms-copying).

4.1 Complexity

As we show in Section 3, computing the coverage of a subset of sources is #P-complete. Interestingly, although the maximum coverage problem, which requires computing coverage of a set of sources, is PP-hard[4], the cost minimization problem has a lower complexity bound and is NP-complete.

We first consider the restricted case where all copiers are full-copying copiers. Section 3 shows that for this case finding the coverage of a set of sources takes only polynomial time. However, we next show that even for this case, both the cost minimization problem and the maximum coverage problem are already NP-complete.

THEOREM 4.1. *The following hold:*

- *The maximum-coverage problem is NP-complete w.r.t. the LCM, NSCM and ASCM cost models when all copiers are full-copying copiers.*
- *The cost-minimization problem is NP-complete w.r.t. the LCM, NSCM and ASCM cost models when all copiers are full-copying copiers.* □

For the complexity of the maximum-coverage problem, the proof uses a reduction from the Knapsack problem for the LCM and ASCM cost models, and a reduction from the Set Cover problem for NSCM. For the cost-minimization problem, we use a different reduction from the Set Cover Problem (the reduction for LCM is slightly different from that for the other two cost models).

For the unrestricted versions of the problems we have the following results. Note that we have different complexity results for the two problems in the general case: cost minimization requires *all* answers to be returned, so we can ignore edges with a fraction less than 1, but the maximum-coverage problem does not have the same property and requires estimating source coverage.

THEOREM 4.2. *The following hold:*

- *The cost-minimization problem is NP-complete w.r.t. the LCM, NSCM and ASCM cost models.*
- *The maximum-coverage problem is PP-hard w.r.t. the LCM, NSCM and ASCM cost models.* □

4.2 Approximation

We now show that we can approximate the maximum-coverage and cost-minimization problems using a greedy algorithm that runs in polynomial time. In the following sections we shall see that under certain restricted conditions, our greedy algorithm can actually obtain optimal answers.

[3]Precisely, all the hardness results in this section refer to the decision versions of the optimization problems; i.e., deciding if there exists a solution achieving a given value of the objective function.

[4]PP-hardness [10] is the analog of #P-hardness for decision problems: for a #P problem "compute $f(x)$", the corresponding PP decision problem is "Does there exist a solution to $f(x) \geq v$, for a specified v?".

```
0: Input: Sources $\mathcal{S}$, dependency graph $G(\mathcal{S})$, cost function $c$.
   Output: Set $\bar{S} \subseteq \mathcal{S}$ as the result.
1: $\bar{S} = \emptyset, \bar{A} = \emptyset$; //$\bar{A}$ is the set of answers.
2: while ($\exists S \in \mathcal{S} - \bar{S}$ such that $T(S) \not\subseteq \bar{A}$) //$T(S)$ is the set of tuples
   in $S$.
3:     Let $S_0 \in \mathcal{S} - \bar{S}$ be the source with maximum $\frac{|S_0 - \bar{A}|}{c(S_0)}$;
4:     $\bar{S} = \bar{S} \cup \{S_0\}$; $\bar{A} = \bar{A} \cup T(S_0)$;
5: return $\bar{S}$;
```

Algorithm 4: GREEDYAPPROX: Greedy approximate algorithm for the cost-minimization problem. For the maximum-coverage problem, we only need to replace the while condition with ($\exists S \in \mathcal{S} - \bar{S}$ such that $c(\bar{S}) + c(S) \leq C_{max}$), where C_{max} is the maximum allowed cost.

We start with the cost-minimization problem. Recall from Section 3.1.2 that we can approximate the coverage in polynomial time; thus, we can efficiently estimate the number of additional tuples we obtain by querying a new source. Algorithm GREEDYAPPROX (Algorithm 4) proceeds by including sources in a greedy fashion: it iteratively picks the source that adds the maximum number of new tuples per unit cost, until no more source can add new answer tuples. The following result gives an approximation guarantee for GREEDYAPPROX.

THEOREM 4.3. *Let α be the number of tuples in the largest source in the input to the cost minimization problem.* GREEDY APPROX *obtains a $\log \alpha$-factor approximation to the optimal solution; i.e., if the optimal cost is c,* GREEDYAPPROX *obtains a cost of at most $c \cdot \log \alpha$.* □

Note that GREEDYAPPROX cannot obtain a constant-factor approximation. Indeed, we can prove that the problem is MaxSNP-hard[5]. This is because in our NP-completeness proofs for the cost-minimization problem, the reduction from the Set Cover Problem preserves the approximation ratio and thus yields L-reductions [20], so the MaxSNP-hardness of the Set Cover Problem carries over.

COROLLARY 4.4. *The cost minimization problem is MaxSNP-hard w.r.t. the LCM, NSCM, and ASCM cost models.* □

Finally, we can easily revise GREEDYAPPROX for the maximum coverage problem by iterating till reaching the cost limit, yielding the following result.

THEOREM 4.5. *We can obtain a $(1 - \frac{1}{e})$-factor approximation to the optimal solution for the maximum-coverage problem.* □

4.3 Single-Source Copying

We consider dependency graphs that satisfy the single-source copying property, i.e., each source copies from at most a single source, and copies all of its data. First, the following result establishes a PTIME complexity for cost minimization for all cost models, and then we show a result for the maximum coverage problem.

[5]MaxSNP-hardness corresponds to a class of problems that cannot be approximated within a factor of $(1+\epsilon)$ for any $\epsilon > 0$ (unless $P =$ NP) [20].

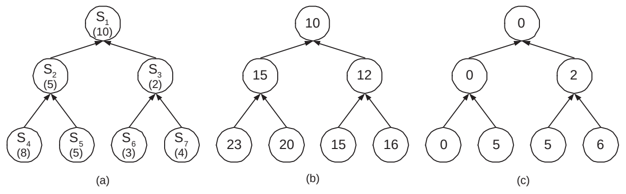

Figure 3: **Example 4.8: (a) input dependency graph; (b)-(c) new answer tuples a source can introduce after selecting each node.**

THEOREM 4.6. *The cost minimization problem can be solved optimally in PTIME w.r.t. the NSCM, LCM, and ASCM cost models when all copiers are single-source copiers.* □

THEOREM 4.7. *Let N be the number of sources in $\mathcal{S}$ and l be the maximum number of sources that are allowed to be queried. When all copiers are single-source copiers, we can find the optimal solution to the maximum coverage problem w.r.t. the number-of-sources cost model in time $O(lN)$.* □

The full proof by induction is based on a greedy algorithm (Algorithm 5) presented in the appendix. In our proof, we consider $\mathcal{T}_k$, the optimal set of k sources, and S, the best source that can be added to $\mathcal{T}_k$. We arrive at a contradiction supposing the optimal set of $k+1$ sources is obtained by adding some source S' to a set $\mathcal{T}'$ of k sources, where $\mathcal{T}' \neq \mathcal{T}_k$.

Next we illustrate the greedy algorithm using an example. The example also shows that the same algorithm is not guaranteed to obtain the optimal solution with respect to other cost models.

EXAMPLE 4.8. *Consider a set of data sources with the dependency graph in Fig.3(a) and assume we can query at most two sources. Fig.3(b) shows the number of answer tuples each source can introduce initially and so we select S_4. Fig.3(c) shows the answer tuples each source can introduce after selecting S_4; accordingly, we select S_7 and obtain the answer set $\{S_4, S_7\}$.*

Note that if we consider the linear cost model and a maximum allowed cost 35, the optimal answer is $\{S_5, S_6\}$, but the greedy algorithm incorrectly chooses $\{S_4\}$. □

In fact, the maximum coverage problem remains NP-complete for the LCM and ASCM cost models. The NP-hardness proof follows from the fact that the reduction from 0-1 Knapsack used for Theorem 4.1 only involves single-source copiers. Further, since single-source copying is a special case of full-copying, the problem remains in NP.

COROLLARY 4.9. *The maximum coverage problem is NP-complete w.r.t. the LCM and ASCM cost models when all copiers are single-source copiers.* □

5. THE SOURCE ORDERING PROBLEM

Ordering the sources optimally is the key challenge for an online query answering system over dependent sources. Our goal is to order the sources in a way that returns answers as quickly as possible. Recall from Section 2 that we are trying to maximize the area under the curve that plots the cumulative number of answers returned with time, and that given a permutation of the sources Π, we denote the area under the curve by $A(\Pi)$. The following theorem establishes some basic complexity results for the coverage problem.

THEOREM 5.1. *The following hold: (1) The decision version of the source-ordering problem is PP-hard in the number of sources. (2) Assuming finding the coverage takes polynomial time, then the decision version of the source-ordering problem is in NP. (3) The source ordering problem can be solved optimally in PTIME when all copiers are single-source copiers.* □

The PP-hardness of the source ordering problem follows from the hardness of the coverage problem (Theorem 3.4). When coverage takes polynomial time, e.g., with full-copying, the source ordering problem is easily seen to be in NP: given a solution, we simply evaluate the total coverage in sequence and compute the area under the curve. However, the exact complexity class under full-copying remains an open problem.

The main result of this section is a *factor-2 approximation* algorithm for the source ordering problem. That is, if we denote the optimal permutation by Π_{opt} and the permutation computed by our algorithm by Π, then $A(\Pi) \geq A(\Pi_{opt})/2$. In the rest of the section, we first show that an optimal permutation must have a *monotonicity* property. We then show that although monotonicity does not guarantee an optimal solution, it ensures a 2-approximation. Finally, we give a greedy algorithm that returns a monotonic permutation; in case we can compute coverage of a set of sources in polynomial time, our greedy algorithm can generate a 2-approximation in polynomial time. Note that results in this section apply to all cost models. We next start with the formal definition of the monotonicity property, which uses notion $Incr$: for a set $\mathcal{S} = \{S_1, \dots, S_l\}$ of data sources and a permutation Π over $\{1, \dots, l\}$,

$$\begin{aligned} Incr(1) &= \langle \{S_{\Pi(1)}\} \rangle; \\ Incr(i) &= \langle \cup_{j=1}^{i} \{S_{\Pi(j)}\} \rangle - \langle \cup_{j=1}^{i-1} \{S_{\Pi(j)}\} \rangle. \end{aligned}$$

DEFINITION 5.2 (MONOTONIC PERMUTATION). *Let $\mathcal{S} = \{S_1, \dots, S_l\}$ be a set of data sources and $G(\mathcal{S})$ be its dependency graph. A permutation Π over $\{1, \dots, l\}$ is said to be* monotonic *if for each $i \in [1, l]$, we have $\frac{Incr(i)}{c(S_{\Pi(i)})} \geq \frac{Incr(i+1)}{c(S_{\Pi(i+1)})}$.* □

Intuitively, the monotonicity property says that the rate of increase of answer tuples as we query more sources decreases monotonically. Not surprisingly, we can show that the optimal permutation for source ordering must be monotonic.

LEMMA 5.3. *Given a set of sources $\mathcal{S} = \{S_1, \dots, S_l\}$ and a dependency graph $G(\mathcal{S})$. If Π_{opt} is an optimal permutation to the source-ordering problem, Π_{opt} is monotonic.* □

Whereas monotonicity is a necessary condition for optimality, the following lemma shows that it is not sufficient.

LEMMA 5.4. *There exists a set of data sources $\mathcal{S} = \{S_1, \dots, S_l\}$, a dependency graph $G(\mathcal{S})$, and a monotonic permutation Π of $\{1, \dots, l\}$, such that Π is not an optimal permutation to the source-ordering problem.* □

Next we prove the main result of this section: any monotonic permutation is at most a factor of two off from any other (and in particular, the optimal) permutation.

THEOREM 5.5. *Let $\mathcal{S} = \{S_1, \dots, S_l\}$ be a set of data sources and $G(\mathcal{S})$ be a dependency graph of $\mathcal{S}$. Let Π_{opt} be the optimal permutation to the source ordering problem and Π be a monotonic permutation of $\{1, \dots, l\}$. Then, $A(\Pi) \geq \frac{A(\Pi_{opt})}{2}$.* □

According to Theorem 5.5, we can design a 2-approximation algorithm to the source ordering problem by greedily picking the next source whose ratio of incremental return versus cost is maximal. Note that this algorithm does not necessarily generate the optimal solution (Lemma 5.4). In cases where we can solve the coverage problem in polynomial time, we can find the 2-approximation solution in polynomial time.

THEOREM 5.6. *Let N be the number of sources in $\mathcal{S}$ and E be the number of edges in the input full-copying DAG $G(\mathcal{S})$. We can find a 2-approximation solution to the source ordering problem in time $O(EN^2)$.* □

6. MORE COMPLEX QUERIES

Until now we considered answering the identity query over our data sources. We now show how our results are used for queries with selection, which are the most common in practice. We also comment on projection and join queries.

Selection queries: A typical query over a large collection of sources is specified by a selection predicate (typically by selecting values in forms). We now show how to extend our results to queries that involve equality and comparison predicates. We denote the set of predicates by $\mathcal{P}$.

We assume that for each data source S_i, we can estimate the selectivity $s_i^{\mathcal{P}}$ of $\mathcal{P}$ for S_i, i.e., the fraction of $n(S_i)$ tuples independently provided by S_i that satisfy $\mathcal{P}$. We can use traditional estimation techniques for this purpose. When we assume equal probability of a source tuple being copied, the fraction of data copied from source S_i should have the same selectivity as S_i w.r.t. $\mathcal{P}$. When we know the exact selection condition for copying, we only consider the copied data that satisfy predicates $\mathcal{P}$.

Given any input $\mathcal{I}$ including a selection query with predicate $\mathcal{P}$, we can transform the problem to an input $\mathcal{I}'$ including an identity query, such that solving any of the four problems we consider gives the same solution on $\mathcal{I}$ and $\mathcal{I}'$. In particular, given a selection query Q with predicate $\mathcal{P}$, a set of sources $\mathcal{S}$, and a dependency graph $G(\mathcal{S}) = (V, E)$, we construct $G^p(\mathcal{S})$ as follows: (1) $(V^{\mathcal{P}}, E^{\mathcal{P}}) = (V, E)$, (2) $n^{\mathcal{P}}(S_i) = s_i^{\mathcal{P}} * n(S_i)$, (3) if annotation R_{ij} is a fraction, $R_{i,j}^{\mathcal{P}} = R_{i,j}$; if R_{ij} is a selection condition, $R_{i,j}^{\mathcal{P}} = R_{i,j} \wedge \mathcal{P}$. We then have the following result.

THEOREM 6.1. *Any of the coverage problem, cost minimization problem, maximum coverage problem, and the source ordering problem gives the same solution for (a) $G(\mathcal{S})$ w.r.t. Q, and (b) $G^{\mathcal{P}}(\mathcal{S})$ w.r.t. the identity query.* $\square$

Projection queries: The main challenge introduced by projections is duplicate elimination. When we project onto a subset of attributes, the number or fraction of tuples that merge to the same tuple value may be different for different sources. In the general case, estimating the size of projection results requires accessing most of the data in each source [5]. If we assume that data provided independently by different sources have the same fraction for any projection, and assume random copying (so in expectation the fraction of copied tuples remaining after a projection is the same for all data sources), we can directly apply the results from this paper.

Join queries: It is easy to extend our dependency model for cases in which sources contain multiple tables, each with different copying sources and patterns. Under the random-copying assumption, our results extend to join queries in a rather straightforward fashion. However, when we use selection queries to model the copying pattern, we need to consider how to estimate the join selectivities. We leave that to future work.

7. CONCLUSIONS AND RELATED WORK

We considered the problem of answering queries over large collection of possibly overlapping data sources. Although we showed that many problems are intractable in general, we proposed greedy or randomized approximation algorithms that ran in polynomial time and have provable quality guarantees. In addition, we identified practical restricted classes of dependencies that yield polynomial-time optimal solutions. Together, these results provide a foundation on which to build such an integration system. One interesting direction for future work is the case in which the data itself is uncertain, and therefore seeing the data from multiple independent sources can affect our belief in the answer.

Previous work [7, 8, 18, 25] developed algorithms for detecting when a data source can be ignored in answering a query. Yet other work [9] studied the use of probabilities to model source coverage and overlap for data integration. These works are all based on coverage of sources and did not consider dependence between sources.

Several authors have discussed mechanisms that result in dependencies between sources on the Web. Leskovec et al. [17] study influences in web-data, such as how blog linkage structures evolve, and [2] provides a formalism for creating web documents by copying portions of data from other documents. Our work is a first step to integrating web data with such dependencies. Of course, a large body of recent work (see [4] for a tutorial) studies the orthogonal issue of tracking data provenance.

8. REFERENCES

[1] R. Alonso, D. Barbara, H. G-Molina, and S. Abad. Quasicopies: Efficient data sharing for information retrieval systems. In *EDBT*, 1988.

[2] P. Atzeni and G. Mecca. Cut & paste. In *Proc. of ACM PODS*, 1997.

[3] L. Berti-Equille, A. Das Sarma, X. Dong, A. Marian, and D. Srivastava. Sailing the information ocean with awareness of currents: Discovery and application of source dependence. In *Proc. of CIDR*, 2009.

[4] P. Buneman and W. Tan. Provenance in databases. In *Proc. of ACM SIGMOD*, 2007.

[5] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: how much is enough? In *Proc. of ACM SIGMOD*, 1998.

[6] X. L. Dong, L. Berti-Equille, Y. Hu, and D. Srivastava. Global detection of complex copying relationships between sources. In *PVLDB*, 2010.

[7] O. Duschka. Query optimization using local completeness. In *Proc. of AAAI*, 1997.

[8] O. Etzioni, K. Golden, and D. Weld. Tractable closed world reasoning with updates. In *Proc. of the Conference on Principles of Knowledge Representation and Reasoning*, 1994.

[9] D. Florescu, D. Koller, and A. Y. Levy. Using probabilistic information in data integration. In *Proc. of VLDB*, 1997.

[10] J. Gill. Computational complexity of probabilistic turing machines. *SIAM Journal on Computing*, 6(4), 1977.

[11] L. Haas. The theory and practice of information integration. In *Proc. of ICDT*, 2007.

[12] A. Y. Halevy, N. Ashish, D. Bitton, M. J. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka. Enterprise information integration: successes, challenges and controversies. In *Proc. of ACM SIGMOD*, 2005.

[13] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data integration: The teenage years. In *Proc. of VLDB*, 2006.

[14] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Manuscript*, 1994.

[15] W. Hoeffding. Probability inequalities for sums of bounded random variables. In *J. of the American Statistical Association*, 1963.

[16] S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1), 1999.

[17] J. Leskovec, M. McGlohon, C. Faloutsos, N. Glance, and M. Hurst. Cascading behavior in large blog graphs. In *SDM*, 2007.

[18] A. Y. Levy. Obtaining complete answers from incomplete databases. In *Proc. of VLDB*, 1996.

[19] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proc. of VLDB*, 2000.

[20] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *JCSS*, 43, 1991.

[21] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. of Computing*, 12, 1983.

[22] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[23] L. G. Valiant. The complexity of computing the permanent. *TCS*, 8(2), 1979.

[24] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[25] Z., S. Kambhampati, and U. Nambiar. Effectively mining and using coverage and overlap statistics for data integration. *TKDE*, 17, 2005.

APPENDIX

A. PROOFS

Proof of Lemma 3.2: The probability that a random tuple $t \in S_0$ appears in S_f is given by $\Pr(t \in S_f | t \in S_0) = \frac{|S_f|}{|S_0|}$. Now consider tuple t in S_0. The tuple t can appear in S_f through one of the paths from S_0 to S_f. The combined probability of these paths is given by the probability of $\mathcal{F}$. □

Proof of Lemma 3.3: We prove the theorem with a reduction from the following #P-complete problem of counting the number of satisfying assignments in a bipartite monotone 2-DNF formula [21]:

> Given sets $\mathbf{X} = \{x_1, \ldots, x_{n_1}\}$ and $\mathbf{Y} = \{y_1, \ldots, y_{n_2}\}$ of boolean variables, and conjuncts $C_1, \ldots, C_m$ where each C_i is of the form $(x_j \wedge y_k)$, count the number of satisfying assignments for $\mathcal{F} = (C_1 \vee C_2 \vee \ldots \vee C_m)$.

Given the input to the above problem, we construct an input to the coverage problem whose solution gives an answer to the above problem. Let the set of sources be $\mathcal{S} = \{S^0, S^1_1, \ldots, S^1_{n_1}, S^2_1, \ldots, S^2_{n_2}, S^3\}$. Let the only source that adds new tuples be S^0, and the dependencies between the sources be as follows. Each S^1_j copies from S^0, and S^3 copies from each S^2_k, and all these edges have fraction $f = 0.5$. Additionally, for each clause $C_i = (x_j \wedge y_k)$, add an edge from S^2_k to S^1_j, i.e., S^2_k copies from S^1_j, with corresponding fraction $f = 1.0$. We are now interested in computing coverage of the set containing the single source $\{S^3\}$.

Intuitively, each S^1_j corresponds to variable x_j, S^2_k corresponds to variable y_k, and the source S^3 may obtain tuples of S^0 only through one of the "paths" corresponding to the clauses. Therefore, using Lemma 3.2, the coverage of the set $\{S^3\}$ is given by the probability Pr of $\mathcal{F}$ in the input problem, where all variables in $\mathbf{X}$ and $\mathbf{Y}$ are independent of one another and have a probability of 0.5 each. Since each variable has a probability of 0.5, the number of satisfying assignments N and the probability Pr of $\mathcal{F}$ are related by $N = Pr * 2^{n_1+n_2}$. □

Proof of Theorem 3.4: Since the limited-coverage problem is #P-hard, the general version of the coverage problem is also #P-hard. The general version of the coverage problem can be solved using a polynomial number of limited coverage problems (see Algorithm 2). If any of the limited-coverage problems gives a non-zero coverage, the general coverage problem has non-zero coverage. Therefore, we establish the #P-completeness from the fact that the limited-coverage problem is in #P: the computation of the probability of $\mathcal{F}$ is in #P. □

Proof of Theorem 3.5: Let $X_i \in \{0, 1\}$ be the random variable that is 0 if the i^{th} iteration falsifies $\mathcal{F}$ and 1 if it satisfies $\mathcal{F}$. We have that expected value $\mathcal{E}(X_i) = \Pr(\mathcal{F})$. Hence, using Hoeffding's inequality [15], which is a special case of Chernoff's bound, we have

$$\Pr(|\Pr(\mathcal{F}) - \frac{C_L}{L}| \geq \epsilon) \leq e^{-2L\epsilon^2}$$

Our result follows by bounding the right-side by δ. □

Proof of Theorem 3.6: Clearly, $|\cup_{S_i \in \mathcal{S}} S_i| = \sum_{S_i \in \mathcal{S}} n(S_i) = D$. Furthermore, the number of tuples from any source S_i present in $\cup_{S_i \in \mathcal{T}} S_i$ is given by N_i, where $N_i = 0$ if $n(S_i) = 0$ or if S_i does not have any descendant in $\mathcal{T}$. Therefore, we have $\langle \mathcal{T} \rangle = |\cup_{S_i \in \mathcal{T}} S_i| = \sum_i N_i$, and hence coverage given by $\frac{\sum_i N_i}{D}$.

Algorithm COVERAGE invokes LIMITEDCOVERAGE at most once for each data source, and thus also takes polynomial time in the number of sources. □

Proof of Corollary 3.10: We create at most $(2bE + 1)^k$ full-copying DAGs: each attribute has at most bE end points and so at most $2bE + 1$ value ranges, and we consider all combinations of ranges for the k attributes. For each full-copying DAG the coverage problem can be solved in time $O(N + E)$ (Theorem 3.7). □

Proof of Corollary 3.11: If we know that distinct attributes are independent of each other, then the complexity is significantly reduced, and we can show the following result. In particular, if all attributes are independent of one another, the dependence on k also becomes polynomial: Instead of considering $P = \prod_{i=1}^{k}(2l_i + 1)$ dependency graphs, we now only need to consider $S = \sum_{i=1}^{k}(2l_i + 1)$ dependency graphs, corresponding to the ranges of values for each attribute independently, as described below.

1. Suppose we have predicates on k attributes $A_1, \ldots, A_k$.
2. For every attribute A_i appearing in any selection query, collect and order all the constants $\{a_{i_1}, \ldots, a_{i_{l_i}}\}$ that appear in the predicates A *op* a, at any edge. Suppose (without loss of generality) $\forall q, a_{i_q} < a_{i_{q+1}}$
3. Define s^q_i to be the selectivity of the q^{th} range for A on a particular source S_0.
4. Create $S = \sum_{i=1}^{k}(2l_i + 1)$ dependency graphs, $G_1, \ldots, G_S$, with all edge fractions being 0 or 1 as follows. Consider a graph for one range, say $a_{i_{q-1}} \leq A < a_{i_q}$. For every edge disregard all predicates on attributes other than A. Associate a fraction of 0 if a predicate on A falsifies the condition above, otherwise associate a fraction of 1.
5. Determine the coverage c^q_i of the graph based on the selectivities s^q_i's for each source in the graph: Source is assumed to have s^q_i tuples, and similarly all other sources are assumed to have number of tuples given by their selectivities. Determine c^q_i for the full-copying graph based on Section 3.1.3.
6. The combined coverage is given by the following expression

$$C = \sum_{q_1=1..(l_1+1),\ldots,q_k=1..(l_k+1)} \prod_{i=1..k} c^{q_i}_i$$

Intuitively, we separately compute the coverage for each possible range of values for each attribute. The coverage for each combination of values is then given by their product because attributes are independent of each other. □

EXAMPLE A.1. *In Figure 2(d), assume $A \leq 2$ had selectivity of 0.3, and $B \leq 2$ had selectivity of 0.4, then selectivity of $A \leq 2 \wedge B \leq 2$ is $0.3 \cdot 0.4 = 0.12$. Hence if the coverage of the dependency graph corresponding to $A \leq 2$ and $B \leq 2$ is 1, then the total fraction of tuples with $A \leq 2 \wedge B \leq 2$ in the final source is also be 1, and its contribution to the coverage would be 0.12.* □

Proof of Theorem 3.12: The #P-completeness directly follows from Theorem 3.4: the #P-hardness reduction carries over, and once again coverage computation is in #P since we only need to solve multiple fraction-copying coverage problems as in Algorithm SELECTCOPYINGCOVERAGE: If at least one of the subproblems has coverage > 0, the overall problem has coverage > 0. For the (ϵ, δ) approximation, the time complexity follows from Theorem 3.5, Theorem 3.10, and Corollary 3.11: The algorithm corresponding to Theorem 3.5 is either applied on $O((bE)^k)$ graphs (similar to the proof for Theorem 3.10) in general and $O(bkE)$

graphs (similar to Corollary 3.11) when attributes are independent of each other. □

Proof of Theorem 4.1:

First bullet: We first prove hardness of the problem. NP-hardness w.r.t. the LCM model can be proved by a reduction from the NP-hard 0-1 Knapsack problem. Given a maximum weight W, a set of n items each with value $\{v_1, \ldots, v_n\}$ and weight $\{w_1, \ldots, w_n\}$, the 0-1 knapsack problem looks for a subset of items whose total weight does not exceed W and maximizes the total value. Given a 0-1 Knapsack problem where $w_i = v_i$ for each $i \in [0, n]$, we can construct an equivalent MCP as follows. For each item i, create a source S_i with v_i independent tuples (and hence cost v_i). There is no dependency between the sources. Set C_{max} in the MCP to W. We can easily prove the correspondence of the optimal solution of the 0-1 Knapsack problem and the solution of the maximum coverage problem.

NP-hardness w.r.t. the ASCM model can be proved by a similar reduction.

We next prove NP-hardness w.r.t. the NSCM model by a reduction from the NP-hard Set Cover problem. Given a universe $U = \{1, \ldots, m\}$, subsets of U, $\{\bar{s}_1, \ldots, \bar{s}_n\}$, such that $\cup_{i=1}^{n} \bar{s}_i = U$, the Set Cover problem decides if there is a set cover of size k. We construct an input to the MCP as follows. Construct m independent sources $S_1, \ldots, S_m$ where each $S_i, i \in [1, m]$, has a single tuple i. Then, for each subset $\bar{s}_j, j \in [1, n]$ in the set cover problem, construct a source S'_j in the MCP where S'_j copies all data from the sources corresponding to all elements of $\bar{s}_j$ and does not add any new tuples. We can prove that there is a set cover of size k if and only if with cost k the MCP has a solution that returns all independently added source tuples (this can be decided in polynomial time when all copyings are full-copying).

In NP: We next show that the decision version of the MCP is NP-complete. Given a target maximum coverage $\mathcal{C}$, suppose a subset $\mathcal{T}$ of sources gives coverage $\geq \mathcal{C}$. Because all sources are full-copying, we can get the coverage of $\mathcal{T}$ in polynomial time. Additionally, the size of each source (i.e., coverage of that single source) can also be obtained in polynomial time. Hence, for each of the cost models, we can polynomially give a solution exceeding some maximum coverage $\mathcal{C}$. □

Second bullet: NP-hardness of the problem w.r.t. the NSCM and ASCM models can be proved by a similar reduction from the Set Cover Problem as in the proof of Theorem 4.1.

The hardness w.r.t. the LCM model is also through a reduction from the Set Cover Problem, but constructed differently. We assume an input to the Set Cover Problem, where for each element i of U, there is a singleton set containing only i. (The problem still remains NP-hard.)

We next construct an input to CMP as follows. Construct $\mathcal{S}$ containing $(m + n + 1)$ sources $\{S_M, S_1, \ldots, S_m, S'_1, \ldots, S'_n\}$, and the following dependency DAG $G(\mathcal{S})$. S_M is the root of $G(\mathcal{S})$ and containing $M > m^2$, tuples. For each $i \in [1, m]$, there is an edge $S_i \rightarrow S_M$ with fraction 1 and $n(S_i) = 1$ (i.e., S_i adds a single new tuple i that is not present in S_M). For each $j \in [1, n]$, S'_j copies all data from the sources corresponding to all the elements of set $\bar{s}_j$ (so the edge has fraction 1) and has $n(S'_j) = 0$.

The solution of the above CMP has cost k if and only if there is a set cover of size k:

1. **"if":** If there is a set cover of size k, pick the corresponding set $\mathcal{T}$ of sources from $\{S'_1 \ldots, S'_j\}$. Clearly $\mathcal{T}$ returns all answer tuples. Further, the total cost of query answering $c(\mathcal{T}) \leq k * (M + m)$, since each S'_j has the M tuples from S_M and at most m other tuples. Now consider any other solution $\mathcal{T}'$ containing at least $k+1$ sources; $c(\mathcal{T}') \geq (k+1) * M$. Since $M > m^2$ and $k \leq m$, we have $c(\mathcal{T}') > c(\mathcal{T})$.
2. **"only if":** Consider an optimal solution $\mathcal{T}$ for the CMP containing k sources. If there is any $S_i \in \mathcal{T}$, we can replace it with S'_j where $\bar{s}_j$ is the singleton set containing i and the resulting set $\mathcal{T}'$ is still optimal. The set of the k sets corresponding to the sources in $\mathcal{T}'$ is a set cover for U.

The proof that the decision version of CMP is NP-complete follows as in the case of MCP above: For any solution we can evaluate the total cost and compute the coverage in polynomial time, since the sources are full-copying. □

Proof of Theorem 4.2: Cost minimization requires *all* tuples to be covered and no edge with fraction less than 1 (or select condition other than *true*) can guarantee all tuples in the derived relation; thus, the problem can be solved by first removing all such dependency edges and then solving the problem on the resulting full-copying dependency graph. On the other hand, the maximum-coverage problem does not have the same property and thus requires estimating coverage of a set of nodes, resulting in PP-hardness: PP-hardness follows from the #P-hardness of the coverage problem (Theorem 3.4). □

Proofs of Theorem 4.3 and Theorem 4.5: We have the following direct L-reduction from the cost minimization problem to the weighted set cover problem: The set of all tuples corresponds to the universal set U, and each source S_i corresponds to a subset $s_i \subseteq U$, where s_i contains the elements corresponding to the tuples in S_i. The weight of s_i is the cost of S_i. GREEDYAPPROX mimics the greedy algorithm for weighted set cover that yields an approximation ratio of $\log \alpha$ [24].

We can easily revise the GREEDYAPPROX algorithm for the maximum coverage problem (MCP): the only difference is that we iterate until reaching the cost limit. We call this the GREEDYAPPROXMCP, which obtains a $(1-\frac{1}{e})$-approximation for the number-of-sources cost model. The $(1 - \frac{1}{e})$-approximation is based on reducing MCP to the k-Coverage problem [14]. Further, we can adapt the approximation ratio for all cost models using a an approximation algorithm for the Budgeted Maximum Coverage Problem proposed in [16]; BMCP is a generalization of the set cover problem with weights on elements and sets. □

Proof of Corollary 4.4: Note that our reductions from set cover in the proof of Theorem 4.1 give L-reductions. A ρ-approximation to the reduced CMP problem gives a ρ-approximation to the original set cover problem. □

Proof of Theorem 4.6: A simple algorithm yields an optimal solution for all cost models. Note that under single-source copying, the dependency graph is a tree. (1) We first find all "special nodes" in the tree: A node is special if it adds at least one tuple independently, and no descendent node adds any tuple independently. Note, clearly, that no two special nodes are ancestor/descendants of each other. Further, any solution to CMP must include at least one node from the subtree rooted at each special node. (2) For the NSCM and LCM cost models, we simply return all special nodes as the solution to CMP. For the ASCM cost model, from each subtree rooted at each special node, we pick the source that has least cost. The set of selected nodes gives an optimal solution to CMP.□

The following proposition establishes several properties of the restricted case on which we base Algorithm 5, used in the proof of Theorem 4.7.

PROPOSITION A.2. *Let S be a set of sources with dependency graph $G(\mathcal{S})$, where all copiers are single-source copiers. The graph $G(\mathcal{S})$ has the following properties.*

```
0: Input: Sources $\mathcal{S}$, dependency graph $G(\mathcal{S})$, maximum number of allowed sources $k$.
   Output: Set $\mathcal{T} \subseteq \mathcal{S}$ as the result of MCP.
1: $\mathcal{T} = \emptyset$;
2: Traverse $G(\mathcal{S})$ in depth-first order; for the root node $R$, $A[R] = n(R)$, and for any other node $S$, $A[S] = n(S) + A[P(S)]$, where $P(S)$ is the parent of $S$.
3: for (i = 1 : k)
4:     Find the leaf node $L$ with the highest $A[L]$;
5:     Add $L$ to $\mathcal{T}$ and mark $A[L] = 0$;
6:     while $(A[P(L)] \neq 0)$
7:         for each (descendant $D$ of $P(L)$ but not of $L$)
8:             $A[D] = A[D] - A[P(L)]$;
9:         $A[P(L)] = 0$; $L = P(L)$;
10: return $\mathcal{T}$;
```

Algorithm 5: SSCMCP: Greedy algorithm for the maximum coverage problem with respect to the number-of-sources cost model when all copiers are single-source copiers.

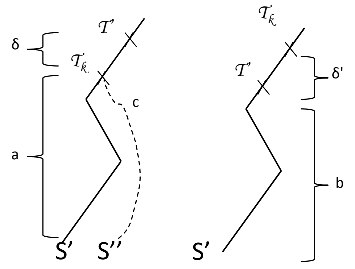

Figure 4: Leaf Root Paths under single-source copying.

- *The graph $G(\mathcal{S})$ is a set of trees.*
- *Given a number k, there exists a set $\bar{L}$ of k leaf nodes in $G(\mathcal{S})$, such that there does not exist any set of k sources whose coverage is higher than $\bar{L}$.*
- *Let S be a leaf node in $G(\mathcal{S})$. Let $\bar{S} \subseteq \mathcal{S}$ be a set of nodes in $G(\mathcal{S})$. Let S' be the node in $\bar{S}$ that has the lowest common ancestor with S and let S_{LCA} be this ancestor. Let $\bar{A}$ be the set of nodes on the path from S_{LCA} (excluding S_{LCA}) to S. Then*

$$\langle \bar{S} \cup \{S\} \rangle - \langle \bar{S} \rangle = \sum_{A \in \bar{A}} n(A). \quad \Box$$

Proof of Theorem 4.7: We prove the optimality of SSCMCP by showing that the optimal $k+1$ set of sources can be obtained by adding one source to the optimal solution for k sources. This, in conjunction with the fact that the greedy algorithm obviously returns the optimal solution for $k = 1$, completes the proof.

We prove the result by contradiction. Let $\mathcal{T}_k$ be the optimal set of k sources and let S be the best source that can be added to $\mathcal{T}_k$. Suppose the optimal set of $k+1$ sources is $\mathcal{T}' \cup \{S'\}$, where $\mathcal{T}'$ is a set of sources and S' is a source such that $S' \notin \mathcal{T}_k \cup \{S\}$. (We must have such a source $S' \notin \mathcal{T}_k \cup \{S\}$ as otherwise $\mathcal{T}_k \cup \{S\} = \mathcal{T}' \cup \{S'\}$, which is optimal for $k+1$ sources.) We have $\langle \mathcal{T}_k \rangle \geq \langle \mathcal{T}' \rangle$ and $\langle \mathcal{T}_k \cup \{S\} \rangle < \langle \mathcal{T}' \cup \{S'\} \rangle$. We show that we can either find a solution with k sources better than $\mathcal{T}_k$, or a solution with $k+1$ sources better than $\mathcal{T}' \cup \{S'\}$.

The main idea used in our argument is the fact that the coverage of any set of sources can be represented by the nodes covered by leaf$\rightarrow$root paths from all the nodes in the set. Whenever a source is added, the increase in coverage is given by the total number of uncovered nodes from it to the covered set of nodes (Proposition A.2). Consider sources S, S', and the sets $\mathcal{T}_k$ and $\mathcal{T}'$ as shown in Figure 4. The figure has marked places where the leaf$\rightarrow$root paths of S, S' meet the already covered nodes of $\mathcal{T}_k$ and $\mathcal{T}'$. Let P_k and P' be the points where S' and S meet $\mathcal{T}_k$ and $\mathcal{T}'$ respectively. The increase in $\mathcal{T}_k$ due to S' is a and the increase in $\mathcal{T}'$ is $a + \delta$: Since there is just one leaf$\rightarrow$root path because each source copies from at most one other source, the a tuples added must completely overlap. Further, S' must add more tuples to $\mathcal{T}'$, otherwise $\mathcal{T}_k \cup \{S'\}$ would result in a higher coverage. Similarly, the figure shows the increments on adding S to $\mathcal{T}'$ and $\mathcal{T}_k$.

Recall $S' \notin \mathcal{T}_k$. Let S'' be the source in $\mathcal{T}_k$ that meets S''s leaf$\rightarrow$path at P_k, with number of tuples added till then being c. We must have $c \geq a$ as otherwise $\mathcal{T}_k - S'' + S'$ is a better solution for k sources. As S'' meets S' at P_k, it cannot belong to $\mathcal{T}'$. Now consider adding S'' instead of S' to $\mathcal{T}'$. If $c > a$, we have an even better solution for $k+1$ sources. Otherwise, if $c = a$, S' and S'' are equivalent in terms of coverage addition, and $\mathcal{T}' \cup \{S''\}$ is also optimal for $k+1$ sources. Hence, we find some other $S'_2 \notin \mathcal{T}_k \cup \{S\}$, $S'_2 \neq S'$, instead of S' and apply the same argument above. $\Box$

Proof of Theorem 5.1: Since the coverage problem was shown to be #P-hard, and the decision version of the source ordering problem is at least as hard, we obtain PP-hardness. When the computation of coverage can be performed in polynomial time, for any sequence, we progressively compute the coverage for every set of sources, and thus can compute the exact area under the curve. Under single-source copying, the source ordering problem becomes PTIME, based on the observation from the proof of Theorem 4.6, as a greedy ordering of the special nodes yields an optimal solution. $\Box$

Proof of Lemma 5.3: Suppose, in contrast, that Π_{opt} is not monotonic. If we denote $c_i = c(S_{\Pi_{opt}(i)})$, then there exists $i \in [1, l-1]$, such that $\frac{Incr(i)}{c_i} < \frac{Incr(i+1)}{c_{i+1}}$. Construct Π as the same as Π_{opt} except that $\Pi(i+1) = \Pi_{opt}(i)$ and $\Pi(i) = \Pi_{opt}(i+1)$. We next show that Π is strictly better than Π_{opt}, leading to a contradiction.

After switching $\Pi_{opt}(i)$ and $\Pi_{opt}(i+1)$ to get Π, we have

$$\begin{aligned} & A(\Pi_{opt}) - A(\Pi) \\ = \; & c_i \cdot Incr(i) + c_{i+1} \cdot (Incr(i) + Incr(i+1)) \\ & -c_{i+1} \cdot Incr(i+1) - c_i \cdot (Incr(i) + Incr(i+1)) \\ = \; & c_{i+1} \cdot Incr(i) - c_i \cdot Incr(i+1) < 0 \end{aligned}$$

This proves the claim. $\Box$

Proof of Lemma 5.4: Consider $\mathcal{S}$ with 103 data sources. For each $i \in [1, 100]$, S_i independently provides a single tuple. Source S_{101} copies all data from $\{S_1, \ldots, S_{50}\}$; source S_{102} copies all data from $\{S_{51}, \ldots, S_{100}\}$; source S_{103} copies all data from $\{S_{25}, \ldots, S_{75}\}$; and none of these three sources adds new tuples. Consider the source ordering $S_{103} \rightarrow S_{102} \rightarrow S_{101}$ and then an arbitrary ordering of the rest of the sources. Under the number-of-sources cost model, the rate of increase of coverage is monotonically decreasing; that is, the permutation is monotonic. However, it is not optimal: an optimal permutation is $S_{101} \rightarrow S_{102} \rightarrow S_{103}$ (and then the rest of the sources). $\Box$

Proof of Lemma 5.5: Let $C = \sum_{i=1}^{l} c(S_i)$. Also, $\langle \mathcal{S} \rangle = \sum_{i=1}^{l} Incr(i)$. Then, we have

$$A(\Pi_{opt}) \leq C \cdot \langle \mathcal{S} \rangle.$$

Since Π is monotonic, for each unit of cost, the incremental return decreases monotonically. Thus, we have

$$A(\Pi) = \sum_{j=1}^{l} \left(c_j \cdot \sum_{i=1}^{j} Incr(i) \right) \geq \sum_{j=1}^{C} j \cdot \frac{\langle \mathcal{S} \rangle}{C} > \frac{C}{2} \cdot \langle \mathcal{S} \rangle \geq \frac{A(\Pi_{opt})}{2}$$

$\Box$