Maximizing Coverage of Mediated Web Queries

Ramana Yerneni
Stanford University
yerneni@db.stanford.edu

Felix Naumann
Humboldt-Universität zu Berlin
naumann@dbis.informatik.hu-berlin.de

Hector Garcia-Molina
Stanford University
hector@db.stanford.edu

Abstract

Over the Web, mediators are built on large collections of sources to provide integrated access to Web content (e.g., meta-search engines). In order to minimize the expense of visiting a large number of sources, mediators need to choose a subset of sources to contact when processing queries. As fewer sources participate in processing a mediated query, the coverage of the query goes down. In this paper, we study this trade-off and develop techniques for mediators to maximize the coverage for their queries while at the same time visiting a subset of their sources. We formalize the problem; study its complexity; propose algorithms to solve it; and analyze the theoretical performance guarantees of the algorithms. We also study the performance of our algorithms through simulation experiments.

1 Introduction

Web sources often provide limited information "coverage." For instance, one type of information source is search engines, such as Lycos [27], Northern Light [29] and Yahoo [30]. The search engines may index a large number of pages, but they do not even come close to covering the entire Web. For instance, a recent article [18] reports that Northern Light covers about 16% of the Web pages, while Yahoo and Lycos cover about 8% and 3% respectively. In a similar fashion, Web stores may only offer limited brands, or digital libraries may focus on particular topics (e.g., NCSTRL on computer science technical reports).

Our goal is to increase the coverage a query "sees" by pooling the information from multiple sources. For instance, to provide a search engine with broader coverage, we can construct a mediator (or meta-search engine like [28]). When a user submits a search query to the mediator, appropriate queries are sent to various search engines, say Northern Light, Yahoo and Lycos. The results of the three queries are integrated to construct the answer to the user query. The mediator may do additional processing on the integrated answer, like ranking the results according to the user's personal preference or according to other criteria.

The benefit of increased query coverage is that the query is computed over a larger body of information, thus yielding better results. For example, consider a user looking for Web pages on "basketball players." If he goes to a single search engine he will get the "top pages" (according to the metric in use) out of the 5 or 10% of Web pages that that engine indexes. The mediator query, on the other hand, can potentially find the "top pages" out of a larger set, and find more pages that may be of interest to the user. Note that the user need not get *more* data from the mediator than from a single engine. In both cases the user may get say 20 links to Web pages, but the mediator answer should be of higher quality. The same effect is expected in other scenarios: For instance, we will find more low priced CDs if we consult multiple music stores, and more compatible dates if we visit multiple singles-dating services.

There are unfortunately three factors that complicate the mediation process we have described: (1) information sources may charge for their services, so that increased coverage must be traded off for higher costs; (2) sources may provide overlapping information, so that using more sources does not necessarily increase coverage, and (3) sources may be temporarily unavailable, so figuring out how to increase coverage at a low price is hard. We briefly describe each of these issues in turn.

Although many Web sources offer free services (for now) to end users, most charge in some way when their services are used by a commercial mediator. To illustrate, consider search engines again. Many generate a substantial portion of their revenue through advertising. A popular model is to charge advertisers based on the number of page views containing their advertisements. Metrics like CPM (cost per thousand impressions) are becoming standard [25]. For instance, Yahoo quotes a \$5 CPM, meaning that it charges \$5 for every thousand page views. When users get information through a mediator, the search engine effectively loses the chance to display its advertisements and loses revenue opportunities. It is the mediator that makes the advertising revenue in this case because it gets to display the results of the search, along with its advertisements, on the user screen. In order to compensate for the loss of revenue, search engines may enter into *revenue-sharing* arrangements with mediators. A search engine may charge a mediator some amount of money for every thousand queries it receives from the mediator.

In order to reduce its query-processing costs, the mediator may decide to submit a user query to only a subset of its information providers. Of course, the coverage of the query may decrease in this case. This trade-off between coverage and cost is the crux of the problem we study in this paper. Specifically, we develop techniques to help mediators achieve the highest coverage for their queries under a given *cost limit*.

We also consider the impact of heterogeneous query-processing costs at sources. For instance, search engines may have different CPM values for their advertisers and hence may charge differently for mediator queries. Variable costs complicate our problem significantly. In particular, if all sources have the same cost, the mediator may simply visit sources in descending order of their coverage. However, if costs differ, the mediator may have to go to a smaller (less coverage) source before a larger one because the former may be cheaper.

The second complexity factor we must deal with is overlapping coverage. For instance, different search engines may index the same Web pages. In this case, the mediator is effectively paying multiple times for the same page. Depending on the profile of content overlap among sources, the mediator needs to select sources that provide "new" information at low cost. For instance, it may be a good idea to search over Northern Light and Lycos, instead of Northern Light and Yahoo, even if Lycos indexes fewer pages than Yahoo, and Yahoo charges less per query than Lycos, if

Lycos has less overlap with Northern Light than Yahoo.

The third complexity factor is the potential for intermittent source unavailability. A query to a Web source may timeout because of transient problems at the source or because of network congestion. In such a situation, the mediator must gather its information from the rest of the providers, albeit with poorer coverage. Source unavailability complicates our problem significantly as illustrated by the following example.

EXAMPLE 1.1 Consider a mediator built on top of four search engines: Northern Light, Yahoo, Lycos and Excite [26]. Suppose that our cost limit for a given query and the source costs imply that we can only query two sources. Let the content overlap among the four search engines be such that the two best options for the mediator are to visit the combination of Northern Light and Lycos or the combination of Yahoo and Excite. Let the mediator pick the first option. After having sent queries to Northern Light and Lycos, the mediator may realize that Lycos is unavailable. At this stage, the mediator cannot switch over to the option of visiting Yahoo and Excite, because it has already used up a query by visiting Northern Light. □

Note that mediators may not be able to find out which sources are unavailable by simply "pinging" the sources. Often, a Web source times out in answering a query, not because its machine is down, but because it may have too much backlog. Pinging the physical machine on which the Web source resides may not give the mediator enough information about the unavailability of the source to answer the query. The only way to find out is to send the query to the source: if the source does not answer, our request will timeout and no charge will be made. Otherwise, we will be charged for the service. Thus, we cannot find out "for free" if a source is available.

In summary, the problem of achieving high coverage for mediated queries is challenging due to three factors: revenue-sharing arrangements that introduce different source costs, content overlap, and source unavailability. In this paper, we study this problem and propose techniques to solve it effectively. In particular, we make the following contributions:

1. We formally define the problem (Section 3) and study its complexity (Section 4). We show that even simple cases of the problem are intractable, and we prove that it is impossible to solve the general case of the problem optimally.

2. We identify scenarios of the problem for which we construct efficient algorithms that generate optimal solutions (Section 5).

3. We develop approximation algorithms for the general case of the problem (Section 6) and study the quality of approximation of our algorithms analytically and through performance experiments (Section 7).

2 Related Work

Web mediation systems are an emerging trend and many system prototypes have been described in the literature [2,6,9,13,23,19]. The main focus of these systems has been the integration of data from multiple sources, and how to deal with the limited query capabilities of sources. The problem of trading off the quality of mediator answers for processing cost has received very little attention in these systems.

Conceptually, the query we consider in this paper is a UNION query of the form R_1 union R_2 union R_3 ... where R_i is what is at each source. However, unlike traditional UNION queries, we do not require a complete answer. We want the "largest" answer given some cost limit. As far as we know, these semantics have not been considered before for UNION queries in conventional query-processing contexts. Because of the different UNION semantics, our algorithms are quite different from traditional query-optimization algorithms where there is only *one* possible answer, the complete one. Also, traditional algorithms do not deal with source-content overlaps.

Recent work (e.g., [7,8]) has studied how to efficiently obtain the initial part of a query answer. In a sense, we also obtain a partial answer, although our cost and objective functions are different from traditional ones. Other recent work has focused on initially computing a low-quality answer and iteratively improving the quality. For example, [16] studies aggregate queries that are very amenable to good approximate computation based on partial data. This work, like ours, deals with trading off quality for cost. However, in our case, mediator queries may need more than just summary results. For instance, when looking for "comparison shopping" information across many Web stores, basing the answer on a small initial set of answers may not yield a good enough answer.

Answering queries efficiently based on statistics and cached data at mediators has also been a popular topic in recent literature. The work on query processing based on statistics [1,14,22] focuses on handling queries that compute aggregated information to perform decision support tasks. We consider more general types of queries at mediators that may not be answerable by just looking at statistics. Caching data at mediators [3] allows for more efficient query processing and also helps the mediator deal with intermittent source failures. This technique is complementary to our work in the sense that one could consider the cached data as a source that is always available for the mediator. Of course, there may be additional issues raised by caching data, regarding the quality/staleness of data and data ownership, that are not relevant in our context.

Altering query plans at run time to cope with source failures has also been studied in the literature [5,15,17,24]. For example, in [24], when an unavailable source is observed, its participation is delayed by visiting other sources, but the unavailable source is retried later on. The goal is to compute the complete answer eventually and to merely alter the execution order of the query plan components. Our goal is not to compute the complete answer, but to arrive at a good enough answer under a cost limit. In particular, when our mediator encounters an unavailable source, it can just skip it. In the SIMS project [17], when sources fail during the execution of a plan, the corresponding queries are routed to alternate sites. If for an unavailable source in a plan there is no equivalent site, then the plan cannot be executed. However, in our problem context, as we are looking for as much coverage as possible, we look for alternate sources in the same domain, without restricting the search to only equivalent sources.

There have been studies (like [4] and [18]) that describe the coverage of data in Web search engines and how they can be improved through meta-search engines. We have used the notion of coverage as a measure of the quality of the answers provided by mediators, and discussed techniques to enhance the coverage of mediated queries. Coverage for database systems has been defined formally in [20]. Other metrics for measuring the quality of answers are considered in [21].

3 Framework

Each source contains a set of objects (e.g., indexed Web pages). The *size* of each source is the number of objects it contains. The *universe* contains all possible objects. We define the *coverage* of a source simply as the ratio between the size of the source and the size of the universe.

When a user poses a query to a mediator, the mediator sends corresponding queries to its sources, obtains answers from them and processes these answers to arrive at the answers to the user query. In order to be cost-effective, the mediator may contact only a subset of its sources when processing user queries. We define the *coverage of a mediated query* as the total number of distinct objects the query is computed on, divided by the size of the universe.

EXAMPLE 3.1 Consider three sources X, Y and Z, and a mediator built on top of them. Let the coverage of X be 0.5. That is, 50% of the universe of objects are represented in X. Let Y and Z have 0.3 and 0.2 coverage respectively. Assume that the sources do not overlap in their content.

If the mediator only uses source X, we have a coverage of 0.5 for the mediated query. If the mediator goes to all three sources, the coverage goes up to $0.7 + 0.3 + 0.1 = 1.0$. If only X and Z are visited, the mediated query has a coverage of $0.5 + 0.2 = 0.7$. □

Note that our definition of coverage is independent of the query at hand. We could equivalently define a query-dependent notion, where the coverage of a source is the number of objects at the source that satisfy the query divided by the total number of objects (over all possible sources) that satisfy the query. With such a definition, source X may have better coverage than Y for one query, but the reverse could hold for another query. Query-specific coverage can be estimated using traditional result-size estimation techniques [12].

Note that the more general definition does not change the nature of our problem, and the algorithms we will present still work with this more general notion. The only difference is that algorithms use query-specific coverage statistics, rather than generic ones based on the amount of data at a source. In either case, the goal is to maximize the coverage of the mediated query, since we expect large coverage to yield a better quality or a more comprehensive answer.

3.1 Cost of Source Queries

We consider a flexible cost model that has two components: money and time. Each source query has a money cost and a time cost that depend on the source. Also, different queries to the same source could cost different amounts (perhaps depending on the size of the answers).

The total money cost for a mediator query is the sum of the money costs of all the source queries involved. The total time cost is computed from the point of view of *response time*. That is, we allow mediators to initiate source queries in parallel and achieve smaller time costs.

To obtain an overall cost measure, we use a *conversion factor* F between money and time. If a query costs M money and T time, we say that its total cost is $M \times F + T \times (1 - F)$.

EXAMPLE 3.2 Consider three search engines, Northern Light, Yahoo and Lycos. Assume that queries to Northern Light have a money cost of 1 unit and a time cost of 3 units, while queries to Yahoo have a money cost of 2 units and a time cost of 2 units, and queries to Lycos have a money cost of 2 units and a time cost of 1 unit.

A mediator may use the following strategy: First go to Northern Light and Lycos in parallel, and only go to Yahoo if one of the other two sources is unavailable. The cost of this strategy depends on the availability of Northern Light and Lycos. If both are available, the money cost is 3 units and the time cost is 3 units. An alternative strategy is the following one: First go to Yahoo and Lycos in parallel, and only go to Northern Light if one of the other two sources is unavailable. If both Yahoo and Lycos are available, the money cost is 4 units and the time cost is 2 units.

To decide which strategy is best overall, we use our conversion factor. We consider the case of all sources being available. If the mediator cares only about money, F is set to 1 and the overall costs of the two strategies are 3 units and 4 units respectively (i.e., the first strategy is cheaper). If the mediator cares only about time, F is set to 0 and the overall costs are 3 units and 2 units respectively (i.e., the second strategy is cheaper). If money is three times as important as time, F is 0.75. The overall cost of the first strategy is $3 \times 0.75 + 3 \times 0.25 = 3$; the cost of the second strategy is 3.5 (i.e., the first strategy is cheaper). □

3.2 Coverage Overlap

The sets of objects at different sources can overlap. In general, quantitative as well as qualitative information about source-content overlaps may be available [4,10]. For instance, we may know that there is a 30% overlap between the sets of indexed pages of two search engines, or we may know that one set of pages is fully contained in the set of another engine. We note that quantitative overlap information is usually hard to obtain, and may be inaccurate. Thus, in this paper, we focus on qualitative overlap information. In particular, we consider the following four cases of qualitative overlap between a pair of sources:

1. Disjointness: The two sources have no objects in common. With a query-specific notion of coverage, it could be that the query yields disjoint result sets at the two sources.

2. Equivalence: The two sources have the same set of objects. Perhaps, the sources are mirrors, or they contain the same information with respect to the query at hand.

3. Subset: The collection of objects at one source is a subset of the collection of objects at the other source. Once again, the containment relationship may be absolute in nature or may be relative to the query at hand.

4. Independence: The set of objects at one source is independent of the set of objects at the other source. The probability that any given data object (or query result) is at a source is given by the source's coverage. If the universe has U objects, and source R has $cov(R)$ coverage, and a second source S has $cov(S)$ coverage, then there will be $U \times cov(R) \times cov(S)$ shared objects between the two sites. (Source R has $U \times cov(R)$ objects, and with probability $cov(S)$ each of these objects is in the second source.)

When computing the coverage of a mediated query, we need to consider the content overlap among the participating sources. For two sources R and S, the coverage of the mediated query, $cov(P)$ is computed as follows.

1. If R and S are disjoint, $cov(P) = cov(R) + cov(S)$

2. If R and S are equivalent, $cov(P) = cov(R) = cov(S)$

3. If R is a subset of S, $cov(P) = cov(S)$

4. If R and S are independent, $cov(P) = cov(R) + cov(S) - cov(R) \times cov(S)$

If there are more than two sources, we proceed incrementally. That is, we compute the combined coverage of the first two sources, and then treat them as a single source that is combined with the third source, and so on.

EXAMPLE 3.3 Consider again the search engines Northern Light, Yahoo and Lycos. Let the total number of indexable Web pages be 600 million and let the three search engines index 100 million, 50 million and 20 million pages respectively. That is, the coverage of Northern Light is 0.16, the coverage of Yahoo is 0.08 and the coverage of Lycos is 0.03.

Assume that the pages indexed by each engine are mutually independent. The combined coverage of Northern Light and Yahoo is $0.16 + 0.08 - 0.16 \times 0.08 = 0.23$. The coverage of all three engines together is $0.23 + 0.03 - 0.23 \times 0.03 = 0.25$. □

3.3 Unavailability of Sources

Sources may temporarily be unavailable during query processing, e.g., due to network congestion or source overload. Mediators have no *a priori* availability knowledge. The only way a mediator can find out if a source is available is by sending it the query at hand. (As discussed earlier, pinging is not sufficient to detect source unavailability.)

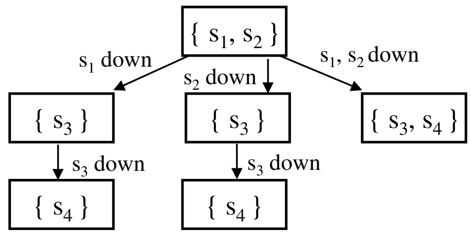

Figure 1: Example Execution Tree

To simplify our model, we assume that if a mediator observes an unavailable source, then that source will not be available for the rest of the time in which the user query is processed. That is, the mediator can eliminate a source from consideration (for the given query) once the mediator notices that the source is unavailable.

We assume there is no money cost involved in sending a query to an unavailable source. The time cost is a fixed constant, which may be based on a timeout period.

3.4 Problem Definition

Our goal is to determine how a mediator can achieve the highest coverage for its queries, while expending a cost that is under a given limit. Conceptually, we must generate an execution strategy for the mediator that achieves these objectives.

We represent a mediator execution strategy by a flexible structure called *execution tree*. The execution tree indicates the actions to take at run time, depending on the availability of the sources. The root of the tree specifies the first set of sources to be queried (in parallel). After the root is executed, the mediator follows only one branch, depending on whichever sources are available. The second node reached this way indicates the next group of sources to try. Execution continues this way, down the tree, until the cost limit is reached (or there are no more sources to try).

EXAMPLE 3.4 Consider a mediator built on top of sources $\{s_1, s_2, s_3, s_4\}$. Assume that the source costs and query limit are such that only two available sources can be visited.

A simple execution tree for this scenario is shown in Figure 1. The root of the tree, $\{s_1, s_2\}$, indicates that s_1 and s_2 should be initially queried in parallel. If s_1 is unavailable, we follow the left branch; if s_2 is unavailable, we follow the middle branch; and if both are unavailable, we follow the right branch. There is no branch for the case where s_1 and s_2 are up, since in this case we reach our cost limit. The rest of the nodes in Figure 1 tell us how to proceed under different availability scenarios. □

Each execution tree represents a set of possible *executions*, one of which will happen depending on the availability of sources. For example, in Figure 1, if all sources except s_2 are up, the execution involves sending queries to s_1, s_2 and s_3.

Given a particular availability scenario, we say that an execution tree is *locally optimal* if no other execution tree yields larger coverage (under the given cost limit). We define the notion of a *globally-optimal* execution tree as the one that is locally optimal with respect to every possible availability scenario. Thus, the problem we address in this paper is to find the globally-optimal execution tree for a given query and a cost limit.

Note incidentally that a mediator does not have to explicitly construct a full execution tree as shown in Figure 1, before starting to execute the query. Instead, the mediator can have an algorithm that makes decisions equivalent to those in the tree at run time. The net effect of this algorithm is as if the appropriate path in the execution tree is traversed in a dynamic fashion at run time. In the following sections, we present such dynamic query execution algorithms.

4 Complexity of the Problem

Our problem, in general, is complex due to a variety of factors involved – source overlaps, unpredictable source availability, and variable money and time costs.

Theorem 4.1 *In its full generality, our problem is unsolvable. That is, there are cases where there is no globally-optimal execution tree. So no algorithm can guarantee optimal execution of the mediated query in all cases.* □

Due to space limitations, we present proofs of all our theorems and lemmas in the Appendix.

Given that the problem we are dealing with is hard, we investigate various scenarios that are practically useful and that lend themselves to efficient algorithms for exact and approximate solutions. In particular, we adopt two steps of simplification. First, we assume that $F = 1$ and develop efficient algorithms that guarantee optimal solutions in some scenarios and good approximations in others. Later in the paper, in Section 7, we consider scenarios where $F < 1$, i.e., where we execute queries in parallel to improve response time. The second simplification is to consider two cases of the cost profiles. The first, simpler case, is when all sources have the same cost. The second, more complex case, is when sources can have different costs.

In Section 5, we deal with the uniform-cost case. Even when all sources have the same costs, if we allow arbitrary source overlaps, it is impossible to construct an algorithm that guarantees optimal solutions. Therefore, we consider particular overlap profiles, and present efficient algorithms that guarantee optimal solutions for those scenarios. In the case of arbitrary overlaps, these algorithms do not guarantee optimal solutions, but become efficient approximation algorithms.

In Section 6, we deal with variable source costs. Here, further simplification by considering particular overlap profiles does not make the problem tractable (see Theorem 6.1). We develop good approximation algorithms for the problem and discuss the performance guarantees yielded by these algorithms.

In Section 7, we study the performance of all our algorithms through experiments involving the most general case of variable costs, arbitrary source overlaps and unpredictable source availability. We also discuss the response-time benefits of parallel query execution and the consequent increases in the money component of query-processing costs.

5 Uniform Source Costs

In this section, we consider a simplified variation of our problem where all sources have the same cost. This situation may arise in practice, if for example, sources simply charge a fixed fee per query sent to them.

When all sources have the same cost, it may appear that the mediator can just pick the largest L/C sources and send them queries, where L is the cost limit for the query and C is the cost of each source query. However, the mediator needs to be more careful because some of these sources may be unavailable and sources overlap in their content.

In the following, we first present an algorithm that takes into account source unavailability but ignores source overlaps. Later in the section we present a second algorithm that considers source overlaps as well.

5.1 Algorithm Simple

Algorithm Simple implements a greedy strategy, choosing sources in descending coverage order. As shown in Figure 2, when the algorithm notices that a source is unavailable it does not increment the cost it has used up in accessing sources. Thus, given that each source query costs C units and the overall cost limit is L, it does not simply choose the first L/C sources. If some of these sources are unavailable, it queries more than L/C sources.

The following lemma and theorems establish the complexity and performance guarantees of algorithm Simple.

Lemma 5.1 *Algorithm Simple runs in $O(n^2)$ time, where n is the number of sources.* □

Note that, we do not take into consideration the actual time to query the sources when computing the running time of the algorithm.

Theorem 5.1 *The execution tree conceptually traversed by algorithm Simple is globally optimal, as long as the sources are mutually disjoint.* □

Theorem 5.2 *The execution tree conceptually traversed by algorithm Simple is globally optimal if the sources are independent of each other.* □

The above theorems establish that algorithm Simple produces optimal results as long as the sources are mutually disjoint or mutually independent. In a sense, we consider these two situations

Algorithm 5.1 *Algorithm Simple*

Input: Query Q, sources S; cost C, limit L

Output: Result of executing Q at S

1. $Answer \leftarrow \{\}$;
2. $R \leftarrow S$;
3. $U \leftarrow 0$;
4. While ($U < (L - C)$ and R is not empty)
    5. $Next \leftarrow maxSource(R)$;
    6. $R \leftarrow R - \{Next\}$;
    7. Execute Q at $Next$;
    8. If ($Next$ is available)
        9. Collect result into $Answer$;
        10. $U \leftarrow U + C$;
11. Return $Answer$ $\diamond$

Algorithm 5.2 *Algorithm Careful*

Input: Query Q, sources S; cost C, limit L, overlap info

Output: Result of executing Q at S

1. $Answer \leftarrow \{\}$;
2. $R \leftarrow S$;
3. $U \leftarrow 0$;
4. $CHOSEN \leftarrow \{\}$;
5. While ($U < (L - C)$ and R is not empty)
    6. $Next \leftarrow maxSource(R)$;
    7. $R \leftarrow R - \{Next\}$;
    8. If ($Next$ is not a subset of some source in $CHOSEN$)
        9. Execute Q at $Next$;
        10. If ($Next$ is available)
            11. Collect result into $Answer$;
            12. $U \leftarrow U + C$;
            13. $CHOSEN \leftarrow CHOSEN \cup \{Next\}$;
14. Return $Answer$; $\diamond$

Figure 2: Algorithms Simple and Careful.

as cases of homogeneous pairwise-overlap relationships. In the general case, we need to consider more complex overlaps. Specifically, when the pairwise-overlap relationships are heterogeneous – a mix of disjointness, independence, equivalence and containment – algorithm Simple may not follow an optimal execution tree. This behavior is illustrated by the following example.

EXAMPLE 5.1 Consider three sources s_1, s_2 and s_3, with coverages 0.3, 0.2 and 0.1 respectively. Assume that s_2 is a subset of s_1, while s_3 is disjoint from s_1 (and s_2). Let each source-query cost 5 units and let the overall cost limit be 10 units. Assume that all the sources are available. That is, the mediator can query can query at most two 2 sources.

Algorithm Simple will yield an execution that includes s_1 and s_2, the two largest sources. The overall coverage of this execution is 0.3 (s_2 does not contribute anything). Another set of sources, namely s_1 and s_3, has a coverage of 0.4. Thus, algorithm Simple can miss the optimal solution when some sources can be subsets of others. $\square$

5.2 Algorithm Careful

We now consider the case of uniform source costs where the sources overlap in complex ways. We present a new algorithm, called Careful, that takes into consideration information about source overlaps.

The idea behind algorithm Careful is a greedy strategy that chooses sources in descending order of their coverage, while weeding out sources that are not going to contribute to the overall coverage because their contents are contained in sources that have already been queried. The algorithm is formally presented in Figure 2.

The complexity and performance guarantees of algorithm Careful are stated in the following lemma and theorems.

Lemma 5.2 *Algorithm Careful runs in $O(n^2)$ time, where n is the number of sources.* □

Theorem 5.3 *The execution tree conceptually traversed by algorithm Careful is globally optimal, if the source overlaps are limited to equivalence, containment and disjointness relationships.* □

Theorem 5.4 *The execution tree conceptually traversed by algorithm Careful is globally optimal, if the source overlaps are limited to equivalence, containment and independence relationships.* □

To illustrate the difference between algorithms Simple and Careful, recall that in Example 5.1 there were three sources s_1, s_2 and s_3 with s_2 being a subset of s_1 and s_3 being disjoint from s_1 and s_2. In this scenario, algorithm Simple made a suboptimal choice. However, algorithm Careful correctly chooses the optimal set of sources to query (i.e., s_1 and s_3). However, as illustrated by the following example, when the source overlaps include a mix of independence and disjointness relationships, algorithm Careful may not generate an optimal execution.

EXAMPLE 5.2 Consider three sources s_1, s_2 and s_3, with coverages 0.5, 0.5 and 0.4 respectively. Assume that s_1 and s_2 are independent, while s_3 is disjoint from s_1 and s_2. Let each source-query cost 5 units and let the overall cost limit be 10 units. Assume that all the sources are available. That is, the mediator can issue the query to two sources.

Algorithm Careful will yield an execution that includes s_1 and s_2, the two largest sources, with an overall coverage of 0.75. However, there are executions involving s_1 (or s_2) and s_3 that yield a coverage of 0.9. Thus, algorithm Careful can miss the optimal execution when there is a mix of independent and disjoint overlaps. □

6 Variable Source Costs

In this section we relax the condition that source costs must be uniform. That is, we allow the costs of different sources to be different.

6.1 Algorithm Ratio

We start with the simple case when sources have variable costs, but the sources do not overlap. To account for the variable costs, we need to be sensitive to source costs as well as their coverages. That is, a small inexpensive source may be better than a large expensive source. Thus, we consider a simple extension of algorithm Simple that chooses sources with the highest coverage/cost ratios. The new algorithm, called Ratio, simply chooses the next available source in the descending order of coverage/cost ratios.

Algorithm Ratio is presented formally in Figure 3. Note that in Line 5, it calls the function *maxCoverageCostRatio(R)* to obtain the next source to send the query to. We assume that this function eliminates sources from R whose cost is larger than $(L - U)$, while it searches for the source whose cost is under the limit and whose coverage/cost ratio is the largest.

Even though the new greedy strategy based on the coverage/cost ratio seems to be more careful than the one used in algorithm Simple, it is not guaranteed to produce optimal executions. In fact, as demonstrated by the following example, algorithm Ratio can produce executions that are arbitrarily bad.

EXAMPLE 6.1 Consider two sources s_1 and s_2. Let s_1 have a coverage of 0.99 and a cost of 100 units. Let s_2 have a coverage of 0.1 and a cost of 10 units. Let the sources be mutually disjoint and let the overall cost limit be 100 units. Since the coverage/cost ratio of the smaller source s_2 is slightly higher (.01) than that of the larger source s_1 (.0099), algorithm Ratio will query s_2 first. Having used up 10 units of cost, the larger source cannot be queried anymore without violating the cost limit. That is, algorithm Ratio is *stuck* with an execution that is almost 10 times worse than the optimal execution that queries only the larger source s_1. □

The above example illustrates that we can trick algorithm Ratio to not only miss the optimal solutions but also miss by arbitrarily large margins. Example 6.1 also shows that, surprisingly, algorithm Ratio might even do worse than algorithm Simple. In the scenario of Example 6.1, algorithm Simple produces the optimal execution (query only s_1). This execution is almost 10 times better than the execution produced by algorithm Ratio.

The good news about algorithm Ratio is that it is a simple algorithm that runs very efficiently and it performs well in practice. The bad news is that it does not guarantee optimal solutions. In fact, unlike in Section 5, we will not be able to come up with efficient algorithms that guarantee optimal solutions when source costs vary. Even with the simplification of having no source overlaps, our problem remains hard. In fact, even if we add a further simplification by assuming that all sources are available, we still have an intractable problem on our hands, as stated by the following theorem.

Theorem 6.1 *The simplified problem, with no source overlaps, all sources being available and* $F = 1$*, is NP-hard.* □

6.2 Algorithm Dominating

Given that the problem we are dealing with is intractable, algorithm Ratio can be viewed as a simple approximation algorithm. Unfortunately, it does not have any performance guarantees in

Algorithm 6.1 *Algorithm Ratio*

Input: Query Q, sources $S = \{s_1, s_2, \ldots s_n\}$;
costs $\{c_1, c_2, \ldots c_n\}$; limit L

Output: Result of executing Q at S

1. $Answer \leftarrow \{\}$;
2. $R \leftarrow S$;
3. $U \leftarrow 0$;
4. While (R is not empty)
   5. $Next \leftarrow maxCoverageCostRatio(R)$;
   6. $R \leftarrow R - \{Next\}$;
   7. Execute Q at $Next$;
   8. If ($Next$ is available)
      9. Collect result into $Answer$;
      10. $U \leftarrow U + c_{Next}$;
11. Return $Answer$ $\diamond$

Algorithm 6.2 *Algorithm Dominating*

Input: Query Q, sources $S = \{s_1, s_2, \ldots s_n\}$;
costs $\{c_1, c_2, \ldots c_n\}$; limit L

Output: Result of executing Q at some of the sources in S

1. $Answer \leftarrow \{\}$;
2. $R \leftarrow S$;
3. $U \leftarrow 0$;
4. While (R is not empty)
   5. $greedy \leftarrow greedySequenceCoverage(R)$;
   6. $single \leftarrow singleLargestCoverage(R)$;
   7. If ($greedy > single$)
      8. $Next \leftarrow maxGreedySequence(R)$;
   9. Else
      10. $Next \leftarrow maxSource(R)$;
   11. $R \leftarrow R - \{Next\}$;
   12. Execute Q at $Next$;
   13. If ($Next$ is available)
      14. Collect result into $Answer$;
      15. $U \leftarrow U + c_{Next}$;
16. Return $Answer$ $\diamond$

Figure 3: Algorithms Ratio and Dominating.

terms of how bad a solution it can produce. In this section we present a more powerful algorithm, called Dominating, that achieves bounded optimality.

Algorithm Dominating first considers a sequence of sources greedily based on the coverage/cost ratio as does algorithm Ratio. However, before sending queries to the selected sources, the overall coverage of the greedy sequence is compared with the coverage of the largest source. If the largest source has a higher coverage than the total greedy sequence, the greedy sequence is discarded, and only the largest source is queried.

This technique of choosing a *dominating* single source in preference to a greedy sequence of sources protects algorithm Dominating against notoriously bad cases like the one in Example 6.1. In that case, algorithm Dominating finds the optimal solution by querying the larger source, even though based on the coverage/cost ratio the smaller source should be queried.

The technique incorporated by algorithm Dominating is well known in the literature related to problems like the *knapsack* problem [11]. In fact, this technique guarantees 50% optimality for the knapsack problem. However, we cannot simply carry over solutions of the knapsack problem in order to solve our problem. In particular, when sources can become unavailable, the use of this technique no longer guarantees 50% optimality, as illustrated by the following example.

EXAMPLE 6.2 Consider three sources s_1, s_2 and s_3. Suppose that s_1 has a coverage of 0.9 and a cost of 100, s_2 has a coverage of 0.1 and a cost of 5 units, and s_3 has a coverage of 0.9 and a cost of 90 units. Let the overall cost limit be 100 units and let the three sources be mutually disjoint. Finally, let s_1 and s_2 be available while s_3 is unavailable.

Algorithm Dominating first considers the greedy sequence $< s_2, s_3 >$. Then, it compares the combined coverage of this sequence with the coverage of the largest source s_1. Consequently, it decides to go with the greedy sequence. After executing the s_2 query, the algorithm attempts the s_3 query but discovers that s_3 is unavailable. Unfortunately, algorithm Dominating is *stuck* with the small source s_2, as the larger source s_1 can no longer be queried (its cost exceeds the current budget). Algorithm Dominating ends up with a coverage of 0.1, while the optimal solution achieves a coverage of 0.9. □

To limit the potential impact of unavailable sources, algorithm Dominating incorporates the following two techniques:

1. Large sources first, within a sequence of high coverage/cost ratios: Whenever algorithm Dominating chooses a greedy sequence of sources over a single large source, it executes the queries at the sources in descending order of their coverages. The idea is that if all the sources in the greedy sequence are available, it does not matter what relative order these sources are queried. If some of these sources are not available, it may be better if sources queried before the greedy sequence is reconsidered are as large as possible.

2. Dynamic adaptation by recalculation after each source failure: Whenever algorithm Dominating encounters an unavailable source while executing the query from a greedy sequence of sources, it reassesses the continuation of the previously chosen greedy sequence. Specifically, it allows the possibility of abandoning the suffix of the greedy sequence and instead querying a single large source that is larger than all the sources in the suffix put together. To exploit this technique to the fullest extent, algorithm Dominating reassesses the suffix of a greedy sequence in each iteration, whether or not the earlier source in the greedy sequence is unavailable.

Algorithm Dominating is formally presented in Figure 3. It computes the coverage of the greedy sequence in Line 5 and the coverage of the largest source in Line 6. Note that, as in the case of algorithm Ratio, we assume that the call to $greedySequenceCoverage(R)$ eliminates from R all sources whose cost is larger than $(L - U)$.

In the following lemmas and theorem, we state the complexity and performance guarantees for algorithm Dominating.

Lemma 6.1 *Algorithm Dominating runs in $O(n^2)$ time, where n is the number of sources.* □

Lemma 6.2 *If algorithm Dominating selects the largest source in its first iteration and this source is available, then it is guaranteed to achieve 50% optimality.* □

Theorem 6.2 *In general, algorithm Dominating guarantees solutions that are within a factor of* $\frac{1}{n-1}$ *of the optimal solutions, for* $n > 1$*, where* n *is the number of sources.* □

The bound of Theorem 6.2 is tight, as we can construct examples where algorithm Dominating only reaches $1/(n-1)$ optimality. However, in many practical situations algorithm Dominating yields solutions that are often optimal or near-optimal. This fact is demonstrated by the excellent performance of algorithm Dominating in our experimental study (see Section 7).

6.3 Algorithm Super

We now drop the restriction that sources cannot overlap. As in Section 5.2, we allow different dependencies between sources. Again, two sources may be disjoint, identical, may have a subset/superset relationship or be independent. To deal with these content-overlap situations we modify algorithm Dominating to arrive at algorithm Super.

The only difference between algorithm Super and algorithm Dominating is that when looking for a greedy sequence (see Line 5 of algorithm Dominating in Figure 3), algorithm Super eliminates sources from contention if they are subsets of other sources already visited. Due to space limitations, algorithm Super is presented in the Appendix.

We do not have any theoretical guarantees of bounded optimality for algorithm Super, in the case of arbitrary source overlaps. However, our performance experiments (see Section 7) demonstrate that it performs very well in practice.

7 Performance Evaluation

In this section we study the performance of the algorithms we have presented. Our goal is to address the following questions:

- How far from optimal are the algorithms? Two of our algorithms, Simple and Careful, guarantee optimal solutions in some cases, but how poorly do they perform in more general scenarios? The remaining algorithms solve our problem approximately, so how good are their approximations in practice?

- All our algorithms optimize the money-cost component and ignore the time component. Can we add some parallelism to the solutions generated by our algorithms so that answers are obtained faster? If we add parallelism, does the money cost increase, as the time cost decreases?

We answer these questions by conducting performance experiments and analyzing their results. Our objective is not to answer the questions in absolute terms, since there are many parameters (number of sources, coverage of sources, availability profiles, etc.) that can be varied. Rather, we consider a "representative scenario" based on Web meta-search engines, in order to gain some insights and understand the trade-offs.

For our experiments, we implemented all the algorithms; created a simulation testbed with representative synthetic data; ran the algorithms on the testbed; exhaustively computed the optimal solutions for the testbed; and compared the performance of the algorithms with the optimal solutions. Due to space limitations, we only present selected results.

7.1 Parameters of Simulation

The main parameters of our simulation are the coverage of sources, the cost variation of sources and source availability. We consider a testbed of 10 sources whose sizes are chosen to reflect the results of a study on the Web coverage of search engines [18]. In particular, the sizes of the sources are randomly chosen between 5% and 25% of the universe. For availability, coverage overlaps and cost ranges, we chose the following profile:

1. As a base setting, we chose source unavailability to be 0.1. This means that each source is unavailable with probability 0.1, and on the average one source out of 10 will be unavailable for a given query. For the experiments studying the effects of source unavailability, we varied unavailability from 0 to 0.5.

2. In our base setting, we assume that 95% of the pairwise relationships between sources are independent. The remaining 5% of the relationships are evenly distributed between subset and equivalent relationships. We assume that no pair of sources is disjoint, since we believe this is never the case for search engines.

3. The default source cost range is 1 to 5 units. That is, the cost of a query is uniformly distributed between 1 and 5. When studying the effects of the cost range, the lower bound of the cost range is fixed at 1 while the upper bound is varied from 1 to 10.

7.2 Comparing the Algorithms

In each experiment we ran 100 trials, with a cost limit of 7 units, and computed the average behavior of the various algorithms. In particular, for each trial, we randomly generated a source profile, based on the parameter setting described above; picked a cost limit; ran the algorithms; computed the coverage of the answer generated by the algorithms; computed the maximum coverage possible by considering the best possible subset of sources to visit; and obtained the error margin for the various algorithms with respect to the optimal solution. We then computed the average error margins over the 100 trials and reported them in the graphs presented below.

In Figure 4 we show the behavior of the algorithms as source unavailability is varied. On the horizontal axis, the unavailability is plotted from 0 to 0.5, while the vertical axis represents the average error with respect to the optimal solutions. The figure shows that, surprisingly, the algorithms do better as the unavailability *increases*. This "unexpected" behavior is due to the reduced chances for the algorithms to make bad choices when only a few sources are available.

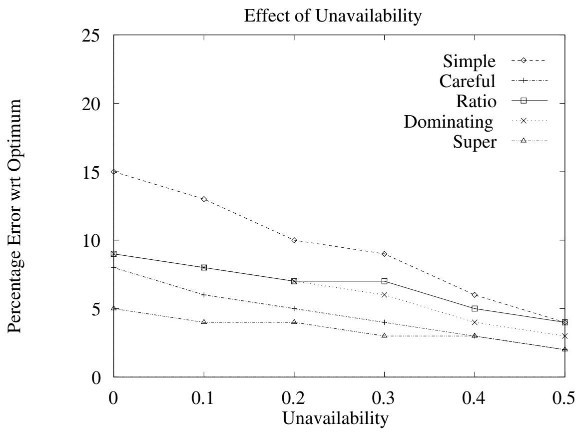

Figure 4: Varying availability of sources.

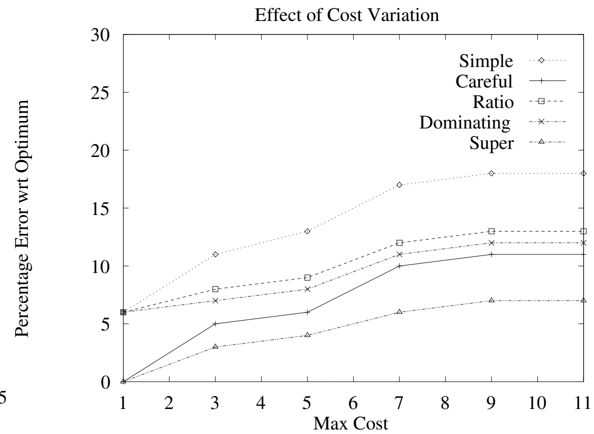

Figure 5: Varying cost range of sources

Figure 5 shows the effect of varying cost ranges for source queries. Source costs vary between 1 and the value shown on the horizontal axis. The vertical axis again shows the average margins of error. The results show that as the cost range increases the algorithms tend to perform worse. The rate of performance loss is much worse for algorithms Simple and Careful than it is for algorithms Ratio, Dominating and Super. This behavior is understandable because the first two make the simplifying assumption that all source queries cost the same. The degree to which this assumption is violated is directly related to the cost range.

We also experimented with varying coverage-overlap profiles, but do not show the results here. However, we note that once again the relative performance of the algorithms – Simple being the worst and Super being the best – held in this experiment. Also, not surprisingly, Careful and Super performed very well even when the number of subset and equivalent relationships were increased significantly, while Simple, Ratio and Dominating performed poorly with increasing percentage of non-independent overlaps.

On the whole, we note that all the algorithms performed very well over a wide range of scenarios. In particular, Careful and Super did consistently well with Super being the best performer among all algorithms and in all the experiments involving variations in source unavailability, cost range and source overlaps. Our experiments showed the surprising result that Dominating did not perform much better than Ratio. Actually, what we observed is that Ratio performed very well and there was not much room for improvement on it for Dominating.

7.3 Parallel Query Execution

The algorithms of Sections 5 and 6 assume that $F = 1$ and try to maximize coverage while only taking into account money costs. If $F < 1$ the problem is substantially more complex, since an algorithm must not only decide the order for source queries, but must also decide what queries to submit in parallel (to reduce response time). A full treatment of this general problem is beyond the scope of this paper.

Instead, we study the impact of parallel execution on money costs, to get a sense of how hard it is to parallelize execution. We start by modifying our algorithms so that they execute k queries in parallel at each step. The first set of k sources is computed as follows. The first source is the one the original algorithm would select. The second source is what the algorithm would have chosen if the first source was available. The third source is what would be chosen if the first two sources were available, and so on, until k sources are selected. Of course, at this point, it is not known if all the sources are available, so the larger k is, the higher the probability of making a mistake. After the first k sources are queried, the algorithm discovers which were available, and selects the next set of k in a similar fashion.

As we increase k we have more parallelism and our time cost is reduced. However, with higher k, our money cost may increase because we decide to run queries with incomplete availability information. Our experiments look at the increase in money cost as k increases. If the increase is "moderate," then parallelization is desirable since we reduce our time cost with little money penalty.

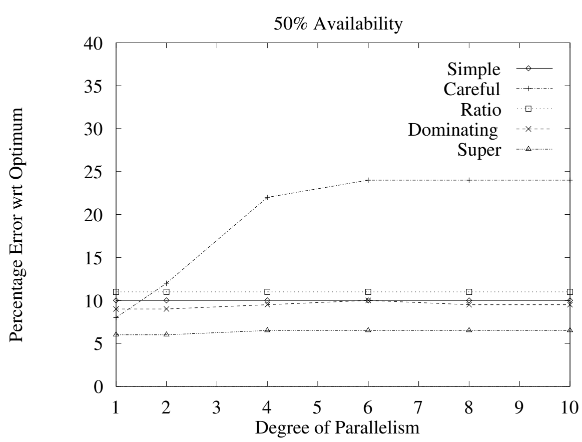

Figure 6: Parallel query execution.

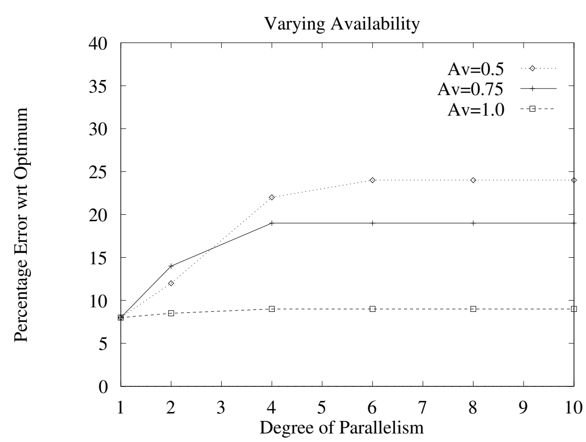

Figure 7: Parallelism vs. availability.

In Figure 6 we show the performance of the algorithms as the degree of parallelism, k, is increased from 1 to 10 (i.e., all sources at once). For this experiment we chose source unavailability to be 0.5 and 10% of the pairwise relationships to be non-independent. As seen from the graph, the higher degrees of parallelism have a marked adverse effect on algorithm Careful, while the other algorithms did not show much adverse affects. Simple and Ratio are not expected to have any loss of performance with respect to the money cost, as they adopt increased parallelism, because the set of sources they visit remains the same irrespective of the degree of parallelism. However, the remarkably stable performance of Dominating and Super is a pleasant surprise. This observation is particularly useful because Super is not only the best performer when only money costs are taken into account but also is insensitive to the degree of parallelism. That is, the best solution to our problem is to employ Super with the largest practical degree of parallelism possible.

When we varied the source availability, we observed that the performance degradation of algorithm Careful decreased with increased availability. This behavior is reported by the results of our experiment in Figure 7. There is an easy explanation for this trend: as fewer sources are unavailable, the chances of mistakes for Careful due to increased degree of parallelism is lower; consequently, the reduced error margins. We also varied the source-overlap profile and studied its effect on how parallel execution alters the performance of the algorithms. Once again, we observed that as the source overlaps increase, the penalty for money factor goes up with increased degree of parallelism. Super still turned out to be the best algorithm and it again remained more or less insensitive to the degree of parallelism, even when the overlaps are many.

8 Conclusions

We described the problem of maximizing the coverage of mediated queries while keeping the mediator costs under check. We discussed the complications arising from source coverage profiles, cost profiles and intermittent unavailability. The complexity of the problem is studied and dynamic query-execution algorithms are developed for the problem. We presented complexity and analytical performance results for the algorithms. Finally, we conducted extensive performance experiments based on the scenario of meta-search engines. Our experiments helped us study the practical characteristics of the various algorithms and draw useful conclusions regarding the trade-off between time and money costs involved in processing mediator queries.

References

[1] S. Acharya, P. Gibbons, V. Poosala. Aqua: A Fast Decision Support System using Approximate Query Answers. *Proc. VLDB Conference*, 1999.

[2] J. Ambite, et al. ARIADNE: A System for Constructing Mediators for Internet Sources. *Proc. SIGMOD Conference*, 1998.

[3] N. Ashish, C. Knoblock, C. Shahabi. Intelligent Caching for Information Mediators: A KR Based Approach. *Proc. KRDB Conference*, 1998.

[4] K. Bharat, A. Broder. A Technique for Measuring the Relative Size and Overlap of Public Web Search Engines. *Proc. WWW Conference*, 1998.

[5] P. Bonnet, A. Tomasic. Partial Answers for Unavailable Data Sources. *Proc. Flexible Query Answering Systems Conference*, 1998.

[6] M. Carey, et al. Towards heterogeneous multimedia information systems: the Garlic approach. *RIDE Workshop*, 1995.

[7] M. Carey, D. Kossman. On Saying "Enough Already!" in SQL. *Proc. SIGMOD Conference*, 1997.

[8] S. Chaudhuri, L. Gravano. Evaluating Top-k Selection Queries. *Proc. VLDB Conference*, 1999.

[9] S. Chawathe, et al. The TSIMMIS Project: Integration of Heterogeneous Information Sources. *IPSJ*, 1994.

[10] J. Cho, S. Narayanan, H. Garcia-Molina. Finding Replicated Web Collections. *Proc. SIGMOD Conference*, 2000.

[11] M. Garey, D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. *W.H.Freeman and Company*, 1979.

[12] H. Garcia-Molina, J. Ullman, J. Widom. Database System Implementation. *Prentice Hall*, 1999.

[13] M. Genesereth, A. Keller, O. Duschka. Infomaster: An Information Integration System. *Proc. SIGMOD Conference*, 1997.

[14] P. Gibbons, Y. Matias. New Sampling-Based Summary Statistics for Improving Approximate Query Answers. *Proc. SIGMOD Conference*, 1998.

[15] S. Hanks, D. Weld. The Systematic Plan Adaptor: A Formal Foundation for Case-based Planning. *University of Washington Computer Science Technical Report 92-09-04*, 1992.

[16] J. Hellerstein, P. Haas, H. Wang. Online Aggregation. *Proc. SIGMOD Conference*, 1997.

[17] C. Knoblock. Planning, Executing, Sensing, and Replanning for Information Gathering. *Proc. IJCAI Conference*, 1995.

[18] S. Lawrence, C. Giles. Accessibility of Information on the Web. *Nature*, vol. 400, July 1996.

[19] A. Levy, A. Rajaraman, J. Ordille. Query Answering Algorithms for Information Agents. *Proc.*

AAAI Conference, 1996.

[20] A. Motro, I. Rakov. Estimating the Quality of Databases. *Proc. Flexible Query Answering Systems Conference*, 1998.

[21] F. Naumann, U. Leser, J. Freytag. Quality-driven Integration of Heterogeneous Information Systems. *Proc. VLDB Conference*, 1999.

[22] V. Poosala, V. Ganti, Y. Ioannidis. Approximate Query Answering using Histograms. *IEEE Data Engineering Bulletin*, 22(4), 1999.

[23] A. Tomasic, L. Raschid, P. Valduriez. Scaling Access to Heterogeneous Data Sources with Disco. *TKDE* 10(5), 1997.

[24] T. Urhan, M. Franklin, A. Tomasic. Cost-Based Query Scrambling for Initial Delays. *Proc. SIGMOD Conference*, 1998.

[25] www.doubleclick.com. DoubleClick – Internet Advertising Solutions Company.

[26] www.excite.com. Excite search engine.

[27] www.lycos.com. Lycos search engine.

[28] www.metacrawler.com. MetaCrawler meta-search engine.

[29] www.northernlight.com. Northern Light search engine.

[30] www.yahoo.com. Yahoo! search engine.

A Supplemental Material on Algorithms and Proofs (Appendix)

Theorem A.1 *(Theorem 4.1 of the main paper) In its full generality, our problem is unsolvable. That is, there are cases where there is no globally-optimal execution tree. So no algorithm can guarantee optimal execution of the mediated query in all cases.* □

Proof: We prove the theorem by constructing a scenario with no globally-optimal execution tree. Consider four sources s_1, s_2, s_3 and s_4 with object sets $\{a, b\}$, $\{c, d\}$, $\{a, c\}$ and $\{b, d\}$ respectively. Let each source have a money cost of 1 unit and let the overall cost limit be 2 units with $F = 1$ (i.e., we don't care about the time component). Since response time is not important, we can focus on execution trees whose nodes refer to one source at a time.

We proceed by contradiction, assuming a globally-optimal execution tree. Without loss of generality, let the root node of this tree be s_1. This node has two children, one indicating the source to visit if s_1 is down and the other indicating the source to visit if s_1 responded. Obviously, the latter must be s_2. If s_1 was available and s_2 was not, it is clear that there are better execution trees with s_3 or s_4 as their root nodes. Thus, the execution tree we started with cannot be globally optimal. Since no globally-optimal execution tree exists, there can be no algorithm that guarantees optimal coverage for mediated queries in all cases. ■

Lemma A.1 *(Lemma 5.1 of the main paper) Algorithm Simple runs in $O(n^2)$ time, where n is the number of sources.* □

Proof: Each iteration of the *while* loop in algorithm Simple (see Figure 2) takes $O(n)$ time as it involves finding the next best source. It is also clear that the number of iterations of the *while* loop is $O(n)$ as in each iteration a source is removed from the remaining list of sources to be considered. ■

Theorem A.2 *(Theorem 5.1 of the main paper) The execution tree conceptually traversed by algorithm Simple is globally optimal, as long as the sources are mutually disjoint.* □

Proof: Let the set of sources that are available (those that answered the queries posed to them) in the execution produced by algorithm Simple be CHOSEN. We will show that there is an optimal execution whose set of available sources is the same as CHOSEN. That is, the execution produced by algorithm Simple is optimal.

Consider an optimal execution that visit the set of sources OPT. First, we note that CHOSEN has L/C sources and we assume that OPT also has L/C sources. Obviously, OPT cannot have more than L/C sources and if it has fewer than L/C sources we can extend OPT by querying additional sources, without compromising the optimality of the execution.

Now, we show that if OPT and CHOSEN differ in the specific sources they contain, we will construct another optimal set that is closer to CHOSEN.

Let s be a source in CHOSEN that is not in OPT. There must exist a source s' in OPT that is not in CHOSEN (because they have the same number of sources). Moreover, s' cannot be larger than s (because otherwise s' would have been chosen by algorithm Simple ahead of s). Consequently, if we replace s' by s in OPT, we will end up with a set that is at least as good. The reason why the replacement is guaranteed to produce a set that cannot be worse is that all the sources are mutually disjoint and we are bringing in a source that is at least as large as the replaced source. Thus, we have produced a new optimal set that is closer to CHOSEN. Continuing in this manner, we can construct an optimal set set of available sources that is identical to CHOSEN. ∎

Theorem A.3 *(Theorem 5.2 of the main paper) The execution tree conceptually traversed by algorithm Simple is globally optimal if the sources are independent of each other.* □

Proof: The proof is similar to that of Theorem A.2. ∎

Lemma A.2 *(Lemma 5.2 of the main paper) Algorithm Careful runs in $O(n^2)$ time, where n is the number of sources.* □

Proof: Each iteration of the *while* loop in algorithm Careful (see Figure 2) takes $O(n)$ time as it involves finding the next best source and checking if it is superfluous. It is also clear that the number of iterations of the *while* loop is $O(n)$ as in each iteration a source is removed from the remaining list of sources to be considered. ∎

Theorem A.4 *(Theorem 5.3 of the main paper) The execution tree conceptually traversed by algorithm Careful is globally optimal, if the source-content overlaps are limited to equivalence, containment and disjointness relationships.* □

Proof: We will show that there exists an optimal execution that has the same set of sources as the set of available sources chosen by algorithm Careful.

Let the sequence of available sources visited by algorithm Careful be CHOSEN. Collect the set of available sources in an optimal execution and order them according to decreasing sizes. Let this sequence of sources be OPT.

First, we note that OPT and CHOSEN have the same number of sources. In particular, they both have L/C sources. If not, we can extend them with "useless" sources that do not contribute any more coverage to the overall result.

Let i be the first position where CHOSEN and OPT differ. That is, CHOSEN and OPT agree in the first $(i-1)$ positions. Let s be the source in CHOSEN at the ith position and let s' be the source in OPT.

There are two cases to consider – s is smaller than s' or s is at least as large as s'.

In the first case (s is smaller than s'), algorithm Careful must have considered s' before s and rejected it because s' is a subset of some source in the first $(i-1)$ positions of CHOSEN (and OPT). Thus, dropping s' from OPT does not lead to a suboptimal solution. In particular, we can construct another execution by replacing s' by s. The newly constructed execution is also optimal and its sequence of sources will agree with CHOSEN in the first i positions.

In the second case (s is at least as large as s'), consider the sequence obtained by replacing s' by s in OPT. Since s is in CHOSEN, it is disjoint from all the sources in CHOSEN. In particular, it is disjoint from each of the first $(i-1)$ sources in CHOSEN, OPT and the newly constructed sequence. If s is disjoint from all the sources that appear after s' in OPT, then the newly constructed sequence is guaranteed to correspond to an optimal execution, as we have just replaced a smaller s' with a larger s. Since OPT is sorted in decreasing order of sources sizes, s cannot not be a subset of any of the sources in OPT that appear after s'. Therefore, the only way the new sequence can be for a suboptimal execution is when s is a superset of some sources in OPT that appear after s'. In that case, we replace all occurrences of such sources by s to obtain yet another sequence. Comparing this sequence to OPT, we see that the only difference in the sets of available sources queried is the substitution of s for sources that are subsets of s. Thus, we have constructed a new sequence that will correspond to an optimal execution and that agrees with CHOSEN up to the first i positions.

In summary, we have shown that we can make CHOSEN agree with an optimal sequence up to the i positions if it agrees with an optimal sequence up to the $(i-1)$ positions. Thus, by applying the argument for increasing values of i (starting from CHOSEN and OPT differing in the first position), we see that we can construct an optimal sequence of sources that agrees completely with CHOSEN. Hence, algorithm Careful is guaranteed to produce optimal solutions. ■

Theorem A.5 *(Theorem 5.4 of the main paper) The execution tree conceptually traversed by algorithm Careful is globally optimal, if the source-content overlaps are limited to equivalence, containment and independence relationships.* □

Proof: The proof is similar to that of Theorem A.4. ■

Theorem A.6 *(Theorem 6.1 of the main paper) The simplified problem, with no source overlaps, all sources being available and $F = 1$, is NP-hard.* □

Proof: The proof is by reducing the knapsack problem [11] to the simplified version of our problem.

In the knapsack problem, we are given a set of items with benefits and costs. We are asked to find the subset that has the largest benefit with a total cost that is under a given cost limit. We translate this problem into our problem as follows.

For each item of the knapsack problem, we create a source and populate it with as many unique objects as the benefit of the item. That is, we create a set of mutually disjoint sources whose coverages are in the same proportion as the benefits of the corresponding knapsack items. The costs of the source queries are the same as the costs of the corresponding items in the knapsack problem. Finally, the cost limit for the query in our problem is equal to the cost limit specified in the knapsack problem.

Given the above translation, if we can find an efficient algorithm that guarantees optimal solutions to our problem, we can use this algorithm to solve the knapsack problem as follows. We can run this algorithm on the instance of our problem that we construct from a given knapsack problem. Then, observing the set of sources chosen by the algorithm to process the query, we can list out the corresponding items for a solution to the knapsack problem.

Since the knapsack problem is known to be NP-hard, we deduce that our problem is also NP-hard. ■

Lemma A.3 *(Lemma 6.1 of the main paper) Algorithm Dominating runs in $O(n^2)$ time, where n is the number of sources.* □

Proof: We first note that, in algorithm Dominating (see Figure 3), the number of iterations of the *while* loop is $O(n)$ as in each iteration a source is removed from the remaining list of sources to be considered. Even though it appears that in each iteration the algorithm needs to find the best greedy sequence, it can be implemented with a preprocessing step of constructing a global greedy sequence at the beginning and then efficiently deriving the best possible greedy sequence in each iteration. The construction of the global greedy sequence can be accomplished in $O(n^2)$ time and the derivation of the best possible greedy sequence in each iteration takes $O(n)$ time. In summary, each iteration of the while loop takes $O(n)$ time, there are $O(n)$ iterations and there is an $O(n^2)$ preprocessing step involved. Therefore, algorithm Dominating can run in $O(n^2)$ time. ∎

Lemma A.4 *(Lemma 6.2 of the main paper) If algorithm Dominating selects the single-largest source in its first iteration and this source is available, then it is guaranteed to achieve 50% optimality.* □

Proof: Let the coverage of the single-largest source be M. Since algorithm Dominating picked the single-largest source in its first iteration we know that M is at least as large as the combined coverage of the greedy sequence, say G. The greedy sequence chooses sources that have the highest coverage per unit cost. Consider adding one extra source to extend the greedy sequence without regard to cost limit (i.e., the addition of this source makes the total cost of the sequence to be larger than the limit). Let this source be the one that has the highest ratio of coverage and cost, among all the remaining sources. Let the coverage of the source be X. Then, we know that $G+X$ is at least as large as the optimum coverage. Also, M is at least as large as F. So, $(G+M)$ is at least as large as the optimum. Finally, since algorithm Dominating picked the single-largest source in the first iteration, $2M$ is at least as large as the optimum. Since the coverage achieved by algorithm Dominating is at least as large as M, we can say that it is guaranteed to achieve at least 50% optimality. ∎

Theorem A.7 *(Theorem 6.2 of the main paper) In general, algorithm Dominating guarantees solutions that are within a factor of $\frac{1}{n-1}$ of the optimal solutions, for $n > 1$, where n is the number of sources.* □

Proof: We prove the theorem inductively.

Base case: $n = 2$. When $n = 2$, there are three cases to consider. In the first case, both the sources are unavailable. In this case, algorithm Dominating produces the optimal solution, albeit not returning any results to the user query. In the second case, exactly one source is available. If the source has a cost that is under the limit for the query, algorithm Dominating correctly chooses the source and produces the optimal result. Otherwise, the optimal result is empty and algorithm Dominating does come up with the empty result. In the third case, both the sources are available. If both sources can be queried (their combined cost is lower than the limit), algorithm Dominating queries both sources and obtains the optimal result. If neither source can be queried, algorithm Dominating yields the optimal result containing no answer objects. If either source but not both can be queried, algorithm Dominating, picks the larger source and so obtains the optimal result. If one of the two sources has a cost under the limit while the other has a cost over the limit, algorithm Dominating once again obtains the optimal result by executing the query at that source. Thus, in all cases, for $n = 2$, algorithm Dominating is guaranteed to produce optimal solutions. In other words, its solutions are guaranteed to be within a factor of $\frac{1}{n-1}$ of the optimal solutions,

Induction case: $n > 2$. Based on Lemma 6.2, whenever algorithm Dominating decides on the single largest source and that source is available, it achieves a result that is within a factor of $1/2$. If the single-largest source is unavailable or if the greedy sequence is chosen in the first place, we examine the largest source of this greedy sequence. By executing the query at the largest source of the greedy sequence first, algorithm Dominating is guaranteed a solution that achieves at least $1/(n-1)$ of the optimal coverage. If this source is available, our goal is reached, if it is not available, the problem is reduced to one with $n-1$ sources, for which we can inductively guarantee $1/(n-2)$ optimality. ∎

A.1 Algorithm Super

In Figure 8, we present formally algorithm Super. In Line 6, when a greedy sequence is being considered, we assume that the call to $SEgreedySequenceCoverage$ eliminates all sources that are subsets of sources already visited, in addition to eliminating sources that cost too much. Moreover, we assume that the coverage of the best greedy sequence is computed for sequences that do not contain sources that are subsets of other sources in the sequence.

Lemma A.5 *Algorithm Super runs in $O(n^3)$ time, where n is the number of sources.* □

Proof: We first note that, in algorithm Super (see Figure 8), the number of iterations of the *while* loop is $O(n)$ as in each iteration a source is removed from the remaining list of sources to be considered. In each iteration, we find the best greedy sequence in $O(n^2)$ time, at the same time eliminating the appropriate subset sources. Thus, the total running time of the algorithm is $O(n^3)$. ∎

Algorithm A.1 *Algorithm Super*

Input: Query Q, sources $S = \{s_1, s_2, \ldots s_n\}$;
costs $\{c_1, c_2, \ldots c_n\}$; limit L; overlap information

Output: Result of executing Q at some of the sources in S

1. $Answer \leftarrow \{\}$;
2. $R \leftarrow S$;
3. $U \leftarrow 0$;
4. $CHOSEN \leftarrow \{\}$;
5. While (R is not empty)
    6. $greedy \leftarrow SEgreedySequenceCoverage(R)$;
    7. $single \leftarrow singleLargestCoverage(R)$;
    8. If ($greedy > single$)
        9. $Next \leftarrow maxGreedySequence(R)$;
    10. Else
        11. $Next \leftarrow maxSource(R)$;
    12. $R \leftarrow R - \{Next\}$;
    13. Execute Q at $Next$;
    14. If ($Next$ is available)
        15. Collect result into $Answer$;
        16. $U \leftarrow U + c_{Next}$;
17. Return $Answer$; $\diamond$

Figure 8: Algorithm Super.