Joint Use of Multiple Learned Statistics for Improving Online Source Selection

Thomas Hernandez
Arizona State University
Department of Computer Science and Engineering
Tempe, AZ 85287
th@asu.edu

Zaiqing Nie
Arizona State University
Department of Computer Science and Engineering
Tempe, AZ 85287
nie@asu.edu

Subbarao Kambhampati
Arizona State University
Department of Computer Science and Engineering
Tempe, AZ 85287
rao@asu.edu

ABSTRACT

The autonomous and decentralized nature of available online sources prevents most existing integration systems from supporting flexible query processing that takes into account conflicting user objectives such as coverage, cost-related, or data-quality objectives. To achieve multi-objective query processing, a data integration system must be able to determine which sources are most relevant for a particular query, given the desired objectives. To do so, it must gather and use source-specific statistics. In this paper we present an approach which automatically gathers coverage and overlap statistics as well as response time statistics, and jointly uses these statistics to select relevant sources. We describe our approach and present experimental results done in the context of BibFinder that demonstrate the efficiency and effectiveness of our approach.

1. INTRODUCTION

The availability of information sources on the web has recently lead to significant interest in query processing frameworks that can integrate online data sources. Not only must a data integration system adapt to the variety of sources available, but it also has to take into account multiple – and possibly conflicting – user objectives. Such objectives can include overall coverage objectives, cost-related objectives (e.g. overall response time, time to first tuple), and data quality objectives (e.g. density [6], provenance of results [2]). To address these conflicting user preferences, query optimization in data integration requires the ability to figure out what sources are most relevant both to the given query and to the specific user objectives. For this purpose, the query optimizer needs to access several types of source-specific statistics and perform a multi-objective optimization of the query using these statistics.

Unfortunately, the autonomous and decentralized nature of the data sources constrains the mediators to operate with very little information about the structure, scope, contents, and access costs of the information sources they are trying to integrate. While some types of statistics may well be voluntarily publicized by the individual data sources, much of these statistics need to be learned/gathered actively by the mediator. This thus raises two challenges:

- How to automatically gather various statistics from autonomous sources, and
- How to use the gathered statistics to support multi-objective query processing.

Seventh International Workshop on the Web and Databases (WebDB 2004), June 17-18, 2004, Paris, France.

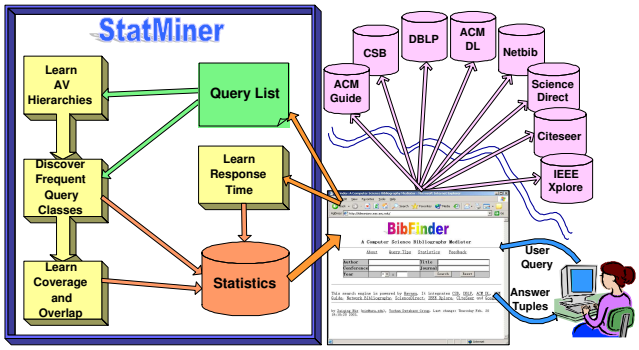

Figure 1: *StatMiner* **Architecture**

Not surprisingly, due to the lack of available source statistics, most existing integration frameworks are unable to support flexible query processing that takes conflicting user preferences into account.

In this paper we concentrate on coverage and response time statistics. We present an integrated approach (see Figure 1) for both gathering and using these statistics, which can effectively resolve conflicting user requirements on response time and coverage. Our approach is being developed and evaluated in the context of *BibFinder*[1], a publicly "fielded" computer science bibliography mediator that we have been developing for the past year. *BibFinder* integrates several online Computer Science bibliography sources, including *CSB, DBLP, Network Bibliography (NET), ACM Digital Library (ACM), ACM Guide (ACMG), IEEE Xplore (IEEE), ScienceDirect (SCI)* and *CiteSeer*[2]. The global schema exported by *BibFinder* can be modeled in terms of the following relation:

paper(title, author, year, conference/journal).

Two major issues need to be resolved by *BibFinder* in order to address conflicting cost and coverage requirements. First, each of the individual sources may export only a subset of the global relation. For example, the source *NET* only contains publications in Networks, while *DBLP* gives more emphasis to Database related publications. Second, the sources being integrated all have different response times which can vary greatly depending on the attributes being bound by the query.

[1] Available at http://rakaposhi.eas.asu.edu/bibfinder

[2] Plans are underway to add several additional scientific sources including *Computational Geometry Bibliography, Compendex, Ingenta,* and *AMS MathSciNet.*

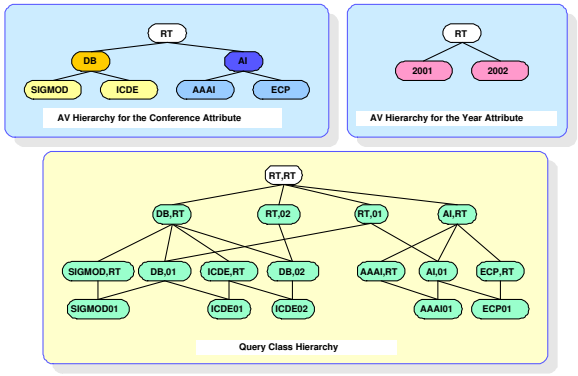

Figure 2: AV Hierarchies and the Corresponding Query Class Hierarchy

Instead of sending the query to all sources – with all the obvious drawbacks implied, our approach directs the query only to the most relevant sources, taking into account the coverage, overlap, and response time of these sources for the given query.

For example, given that a source like ACMG has a significantly broad coverage, the mediator may be tempted to send most user queries its way. This may however not be very efficient considering that ACMG tends to be significantly slower than some of the other sources that have a lower coverage. The most efficient plan may be to call a set of complementary sources that cover all of ACMG's answers but that are each much faster than ACMG. Clearly, a joint optimization of both cost and coverage is necessary to select relevant sources for a given query. Accordingly, our approach uses the statistics we learn to support multi-objective query processing. Handling the interactions between these competing objectives by using the gathered source statistics introduces several new challenges into query optimization, including the need to develop and use more sophisticated cost and reward models for query plans.

The rest of the paper is organized as follows. In the next section we discuss the problem of automatically gathering coverage and overlap statistics and response time statistics, and how that was done in the context of *BibFinder*. In section 3 we explain how we use the learned statistics to select relevant sources for a given query, either by using only coverage and overlap, or by also taking into account response time. In section 4 we describe the results of our experiments to evaluate the impact of using coverage and overlap statistics alone or in conjunction with response time statistics. Finally we conclude in section 5 by pointing out some interesting issues raised by our findings.

1.1 Related Work

There has been little work on statistics gathering in data integration scenarios. Although the utility of quantitative coverage statistics to rank the sources was explored by Florescu *et. al.* [3], the primary aim of the effort was on the "use" of coverage statistics. There also has been some previous work on learning selectivity and response-time statistics both in multi-database literature [11] and data integration literature [4]. In contrast to the previous work, in our research we look at both learning and using these source statistics to support source selection and multi-objective query processing. Finally, some cost and reward models have been suggested in [7, 10, 5, 9].

2. GATHERING SOURCE STATISTICS

In this section we discuss StatMiner, the module we use for modeling and gathering source statistics (see Figure 1). While many statistics could be useful in the source selection process, the work described here focuses on two types of statistics: coverage and overlap statistics, and response time statistics. Note that as the first step of a more general approach, this work only considers selection queries.

2.1 Coverage and Overlap Statistics

The obvious approach for gathering coverage and overlap statistics – which would consist in remembering statistics with respect to each individual query – is of course infeasible. We therefore learn these statistics with respect to *query classes*; a query class being a set of selection queries. We further define query classes such that there is a natural hierarchical subsumption among different classes. The query classes themselves are defined in terms of AV hierarchies – which can be seen as hierarchical clusters of like values of an attribute. Before describing this approach[3], we give more precise terminology.

2.1.1 Terminology

Query List: To reduce the number of statistics stored, statistics are maintained with respect to frequent query classes. We facilitate the discovery of frequent query classes by maintaining an informative log of queries, which contains for each query the query frequency, the total number of distinct answers, and the number of answers from each source set which has answers for that query.

AV Hierarchy: Since we consider selection queries, we can classify the queries in terms of the selected attributes and their values. An *AV hierarchy* (or attribute value hierarchy) over an attribute A is a hierarchical classification of the values of the attribute A, in which leaf nodes of the hierarchy correspond to specific concrete values of A, while non-leaf nodes are abstract values that correspond to the union of values below them (see Figure 2 for two simplified AV hierarchies). The classificatory attributes may either be determined by the mediator designer or using automated techniques. The AV hierarchies themselves can either be hand-coded by the designer, or can be learned automatically. In Section 2.1.2, we give details on how we learn them automatically.

Query Classes: Since a typical selection query will have values of some set of attributes bound, we group such queries into query classes using the AV hierarchies. A query *feature* is defined as the assignment of a classificatory attribute to a specific value from its AV hierarchy. A query class is a set of (selection) queries that all share a particular set of features. The space of query classes is just the cartesian product of the AV hierarchies of all the classificatory attributes. The AV hierarchies induce subsumption relations among the query classes. A class C_i is subsumed by class C_j if every feature in C_i is equal to, or a specialization of, the same dimension feature in C_j. A query Q is said to belong to a class C if the values of the classificatory attributes in Q are equal to, or are specializations of, the features defining C. Figure 2 shows the class hierarchy which corresponds to the two AV hierarchies in the Figure.

Coverage and Overlap: The *coverage* of a data source S with respect to a query Q, denoted by $P(S|Q)$, is the probability that a random answer tuple of query Q is present in source S. The *overlap* among a set $\widehat{S}$ of sources with respect to a query Q, denoted by $P(\widehat{S}|Q)$, is the probability that a random answer tuple of the query Q is present in each source $S \in \widehat{S}$. The overlap (or coverage when $\widehat{S}$ is a singleton) statistics w.r.t. a query Q are computed using the formula $P(\widehat{S}|Q) = \frac{N_Q(\widehat{S})}{N_Q}$, where $N_Q(\widehat{S})$ is the number of answer tuples of Q that are in all sources of $\widehat{S}$ and N_Q is the total number of answer tuples[4] for Q.

[3] Further details on coverage and overlap statistics learning can be found in [8]

[4] We assume that the union of the contents of the available sources within

Similarly, we define coverage and overlap with respect to a query class C. For example, the overlap of a source set $\widehat{S}$ (or coverage when $\widehat{S}$ is a singleton) w.r.t. a query class C can be computed using the following formula:

$$P(\widehat{S}|C) = \frac{\sum_{Q \in C} P(\widehat{S}|Q)P(Q)}{P(C)},$$

where $P(Q)$ is the probability of occurrence of Q in the query list, and $P(C)$ is the probability that a random query posed to the mediator is subsumed by the class C. The coverage and overlap statistics w.r.t. a class C will then be used to estimate the source coverage and overlap for all the queries that are mapped into C (see Section 3).

2.1.2 Learning AV Hierarchies and Discovering Frequent Query Classes

We now discuss how we can automatically learn AV Hierarchies and Query Classes based on the query list maintained by the mediator. The main idea of generating an AV hierarchy is to group similar attribute values into clusters in terms of the coverage and overlap statistics of their corresponding selection queries (i.e. the queries binding these values). The problem of finding similar attribute values thus implies finding similar selection queries. As in any clustering algorithm, we first need to define a distance function between a pair of selection queries:

$$d(Q_1, Q_2) = \sqrt{\sum_i [P(\widehat{S}_i|Q_1) - P(\widehat{S}_i|Q_2)]^2},$$

where $\widehat{S}_i$ denotes the i^{th} source set of all possible source sets in the mediator[5]. Although the number of all possible source sets is exponential in terms of the number of available sources, we only need to consider source sets with answers for at least one of the two queries to compute $d(Q_1, Q_2)$. Similarly we define a distance function to measure the distance between a pair of query classes: $d(C_1, C_2) = \sqrt{\sum_i [P(\widehat{S}_i|C_1) - P(\widehat{S}_i|C_2)]^2}$.

The coverage and overlap statistics $P(\hat{S}|Q)$ for a specific query Q are computed using the statistics from the query list maintained by the mediator. We can then use the distance function with these statistics as the basis to achieve an agglomerative hierarchical clustering for each classificatory attribute.

Once the AV hierarchies have been learned, the query class hierarchy is defined in terms of the cartesian product of the AV hierarchies. The statistics are stored with respect to these query classes. In order to keep the amount of statistics low, we would like to prune query classes which are rarely accessed. The frequently accessed classes can be discovered in a two-stage process. Starting from the query list, the frequent classes can be computed efficiently by iterating over queries with one bound feature, two bound features and so on. The computation can be made efficient by using the anti-monotone property of frequent classes [8].

2.1.3 Mining Coverage and Overlap Statistics

For each frequent query class in the mediator, we learn coverage and overlap statistics. Since overlap statistics can be potentially exponential in the number of sources, we use a minimum support threshold $minoverlap$ to prune overlap statistics for uncorrelated source sets. A simple way of learning the coverage and overlap statistics is to make a single pass over the query list, map each query into its ancestor frequent classes and update the corresponding coverage and overlap statistics vectors $\overrightarrow{P(\widehat{S}|C)}$ of its ancestor classes using the query's coverage and overlap statistics vector $\overrightarrow{P(\widehat{S}|Q)}$ through the formula $\overrightarrow{P(\widehat{S}|C)} = \frac{\sum_{Q \in C} \overrightarrow{P(\widehat{S}|Q)} \times P(Q)}{P(C)}$. When the mapping and updating procedure is completed, we simply need to prune the overlap statistics which are smaller than the threshold $minoverlap$.

One potential problem of this naive approach is the possibility of running out of memory, since the system has to remember the coverage and overlap statistics for each source set and class combination. In order to handle scenarios with large number of sources, we can use a modified Apriori algorithm [1] to avoid considering any supersets of an uncorrelated source set. We first identify individual sources with coverage statistics more than *minoverlap*, and keep coverage statistics for these sources. Then we discover all 2-$sourceSet$ with overlap more than *minoverlap*, and keep only overlap statistics for these source sets. This process continues until we have the overlap statistics for all the correlated source sets.

2.1.4 Learned Statistics from BibFinder

The statistics gathering approach described thus far has been evaluated in the context of *BibFinder*. The query list consisted of 25000 real queries asked by *BibFinder* users. Among them, we randomly chose 4500 queries as test queries and the others were used as training data. The AV Hierarchies for all of the four attributes (i.e. author, title, year, and conference) were learned automatically as described previously. The learned Author hierarchy has more than 8000 distinct values, the Title hierarchy keeps only 1200 frequently asked keyword itemsets, the Conference hierarchy has more than 600 distinct values, and the Year hierarchy has 95 distinct values[6]. Next, the query class hierarchy obtained from these four learned AV hierarchies contained from 500 to 10000 classes, depending on the pruning threshold.

2.2 Response Time Statistics

Unlike coverage and overlap statistics, which need to be learned with respect to specific query features (and hence classes), the advantage of response time statistics is that they are mostly independent of the actual values being bound to attributes in the query. On the flip side though, not only are the response time statistics dependent both on the sources themselves as well as the status of the network, but they may also depend on the set of attributes – as opposed to values – that are bound in the query. We thus need to estimate several types of statistics for the integrated sources (including the time required to open a connection, the time required to parse the results, and the total response time) each with respect to the binding attributes of the queries.

2.2.1 Different Types of Response Time

Average Connection Time: Connection time essentially tells us how long it takes to receive one result page from the source. Note that depending on the source, it may be necessary to request and receive several result pages from a single source[7].

Average Parsing Time: Parsing time refers to the time needed by the mediator itself to parse the results received from the source. Although this is not exactly a response time component, it is important to take it into account as well since the format of the results sent back by the source will greatly influence the parsing time, and

the system covers 100% of the answers of the query.

[5]Note that we are not measuring the similarity of the answers of Q_1 and Q_2, but rather the similarity of the way their answer tuples are distributed over the sources. In this sense, we may find a selection query ($conference$ = "$AAAI$") and another query ($conference$ = "$SIGMOD$") to be similar in as much as the sources having tuples for the former also have tuples for the latter.

[6]Note that we consider a range query (for example: ">1990") as a single distinct value.

[7]For example, ACM only returns 20 answers per page, and therefore may require to navigate through several result pages to retrieve all results.

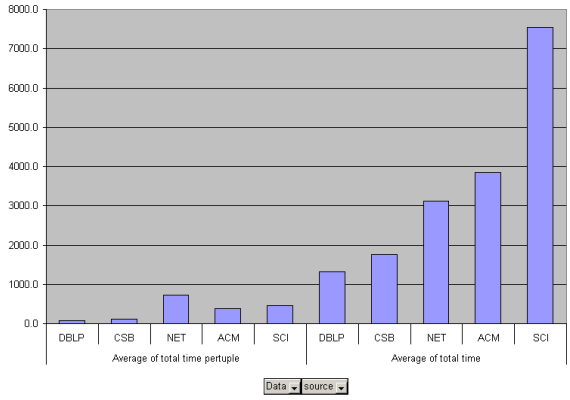

Figure 3: Average total time per tuple and average total time for each source (in ms).

hence the overall response time.

Average Total Time: The average total time gives us an idea of the average time necessary to call, retrieve and parse all the results from each source, regardless of the actual number of results retrieved. Unlike the previous response time statistics, the total time takes into account the case when several pages must be opened from a single source for a particular query. In other words the total time adds up as many connection times as were necessary to retrieve all results.

These response time statistics must be gathered for each source and every type of query binding. Information about the number of results returned by each source must be stored along with the types of response time statistics presented above. Storing the number of results allows to estimate a *per tuple* average of the different response times. As some have shown (c.f. [4]), it is possible that response time is affected by time of day and/or day of week. Therefore it is important, when gathering these response time statistics, to take into account the precise time at which the query is asked. Furthermore, since different query types may lead to very different response times on the same source, the exact query bindings must be stored as well. In order to determine which are the factors that influence the response time of each source, each query attribute and its impact on response time should be analyzed separately.

2.2.2 Learned Response Time Statistics from BibFinder

In order to obtain response time statistics in BibFinder, the sources integrated at the time of the experiments (i.e. *DBLP*, *CSB*, *NET*, *ACM*, and *SCI*) were methodically probed. The experiments were run at regular intervals during day and night, and all information was stored with respect to the time of day it was gathered, as well as the specific source and query. Two different query lists were used for each run. A list of 37 queries was used to probe ACM, DBLP, and CSB. A list of 13 queries was used for NET and SCI. The attributes that were bound in the queries varied as well as the values being bound. Range queries (for the year attribute) were also included.

DBLP has the fastest average connection time with close to 485ms, compared to SCI which had the worst average with 2420ms. The average parsing time per answer tuple was below 100ms for every source except NET which had a parsing time of 501ms per tuple[8].

Figure 3 summarizes the total time statistics. When averaging the total response time on a tuple per tuple basis, the sources can be divided into two sets. DBLP and CSB are the fastest with an average time close to 100ms (i.e. 73ms for DBLP and 107ms for CSB). The second set of sources comprises ACM, SCI, and NET, which have total response times per tuple approximately between 400ms and 700ms.

When looking at the average total time, the sources are ranked in the same order of connection response time. Interestingly, even though either DBLP or CSB often resulted in the highest coverage for most queries, they both have the lowest overall response time at 1323ms and 1767ms respectively. This means that their connection speed and simple result page design (hence faster parsing) compensated for the high number of results they returned. Following DBLP and CSB were NET and ACM between 3000ms and 4000ms, and far behind SCI at 7536ms[9].

Figure 4 shows how some attributes can influence the response time of the sources. The individual tables contain the average total time for different types of queries asked to *BibFinder*, depending on the attributes being bound[10]. Figure 4b. summarizes the effect of having a year range in the query. A range query deteriorates greatly the response time of SCI, as well as that of DBLP, CSB, and NET. However it greatly improves the response time of ACM. Figure 4c. explores non-range queries based on the bindings of the author attribute. It can be seen that binding the author attribute will greatly worsen DBLP's response time. On the other hand, it will improve the response times of NET and ACM.

3. USING THE SOURCE STATISTICS

This section discusses how to use the learned statistics in the source selection process. We will start by discussing how to use the learned statistics to estimate the coverage and overlap statistics for a new query, and how these statistics are used to generate query plans. We will then discuss how to use coverage and overlap statistics in conjunction with response time statistics to further improve source selection.

3.1 Using Coverage and Overlap Statistics

Query Mapping: Given a query Q, we need to gets its coverage and overlap statistics. The statistics will naturally depend on what query class(es) the query should be seen as belonging to. To determine which class(es) the query belongs to, we first get all the abstract values from the AV hierarchies corresponding to the binding values in Q. Both the binding values and the abstract values are used to map the query into query classes with statistics. For each attribute A_i with bindings, we generate a feature set $ftSet_{A_i}$ which includes the corresponding binding value and abstract values for the attribute. The mapped classes will be a subset of the candidate class set $cSet = ftSet_{A_1} \times ftSet_{A_2} \times ... \times ftSet_{A_n}$ where n is the number of attributes with bindings in the query. Let $sSet$ denote all the frequent classes which have learned statistics and $mSet$ denote all the mapped classes of query Q. Then the set of mapped classes is: $mSet = cSet - \{C|(C \in cSet) \cap (C \notin sSet)\} - \{C|(\exists C' \in (sSet \cap cSet))(C' \subset C)\}$. In other words, to obtain the mapped class set we remove all the classes which do not have any learned statistics as well as the classes which subsume any class with statistics from the candidate class set. The reason for the latter is because the statistics of the subsumed class are more specific to the query. Once we have the relevant class set, we compute the estimated coverage and overlap statistics vector $\overrightarrow{P(\widehat{S}|Q)}$ for the new query Q using the coverage and overlap statistics vectors of the mapped classes $\overrightarrow{P(\widehat{S}|C_i)}$ and their corresponding tightness information $t(C_i)$: $\overrightarrow{P(\widehat{S}|Q)} = \sum_{C_i} \frac{t(C_i)}{\sum t(C_i)} \overrightarrow{P(\widehat{S}|C_i)}$. Since the

[8]The fact that NET returns a single page with the complete set of results and their full abstract may explain the relatively high parsing time compared to other sources.

[9]One particular year range query increased the average total time for SCI by about 2000ms.

[10]Due to a lack of space we only show statistics for binding attributes *author* and *year*.

author	(All)
title	(All)
year	(All)
range	(All)
conference	(All)

Average of total time	
source	Total
DBLP	1323
CSB	1767
NET	3117
ACM	3849
SCI	7536

a.

title	(All)
year	(All)
author	(All)
conference	(All)

Average of total time	range	
source	no	yes
DBLP	1143	4462
CSB	1702	2919
NET	2906	4277
ACM	3986	1443
SCI	3411	30221

b.

title	(All)
year	(All)
range	no
conference	(All)

Average of total time	author	
source	no	yes
DBLP	516	2964
CSB	1741	1586
NET	4215	1336
ACM	4395	2807
SCI	3440	3377

c.

Figure 4: Average total time for queries with specific bindings (all bindings in a, *year* range in b, *author* with no *year* range in c.

classes with large tightness values are more likely to provide more accurate statistics, we give more weight to query classes with large tightness values.

Using Coverage and Overlap Statistics to rank sources: Once we have the coverage and overlap statistics, we can use the *Simple Greedy* and *Greedy Select* algorithms described in [3] to select sources. Specifically, *Simple Greedy* generates plans by greedily selecting the top k sources ranked only according to their coverages, while *Greedy Select* selects sources with high residual coverages calculated using both the coverage and overlap statistics (residual coverage of a source S_2 for a query Q w.r.t. source S is $P(S_2|Q) - P(S_2 \wedge S|Q)$). An algorithm for efficiently computing residual coverage using the estimated coverage and overlap statistics is discussed in [8]. Specifically, we only keep overlap statistics for correlated source sets with sufficient number of overlap tuples, and assume that source sets without overlap statistics are disjoint (thus their probability of overlap is zero). If the overlap is zero for a source set $\widehat{S}$, we can ignore looking up the overlap statistics for supersets of $\widehat{S}$, since they will all be zero by the anti-monotone property.

3.2 Multi-objective query processing

By using only coverage and overlap statistics, a data integration system would always call the sources with highest coverage regardless of their response time, even though there may be a better, faster source set to call to achieve the same coverage. In our current work, we experiment with two different source utility models which take into account both the coverage statistics and the response time statistics of each source. In both formulas, $cost(S_i|Q)$ denotes the average total response time of source S_i for queries with the same binding attributes as Q.

Our *Discount* utility function essentially uses a delay penalty:

$$utility_D(S_i) = P(S_i|Q) \times \gamma^{cost(S_i|Q)}$$

The penalty ensures that if the delay to achieve a high coverage is large, then the utility of that source is reduced considerably, and faster sources with lower coverage would be preferred. Our *Weighted Sum* utility function uses coverage and response time in a linear combination:

$$utility_W(S_i) = \delta \times log(P(S_i|Q)) - (1-\delta) \times cost(S_i|Q)$$

Naturally, the discount factor γ and the weight δ can vary, thereby leading to source call plans which can put more or less weight on the source response time. In both functions, increasing γ or δ will logically give more importance to coverage, while decreasing them favors response time.

Both $utility_D$ and $utility_W$ are used in a greedy way when selecting the source plan for a given query: the source with the largest utility is added to the plan, then the source with next largest utility, and so on. Note that except for the selection of the first source, the coverage used in both utility functions to choose the next best source is actually the residual coverage with respect to the set of already selected sources.

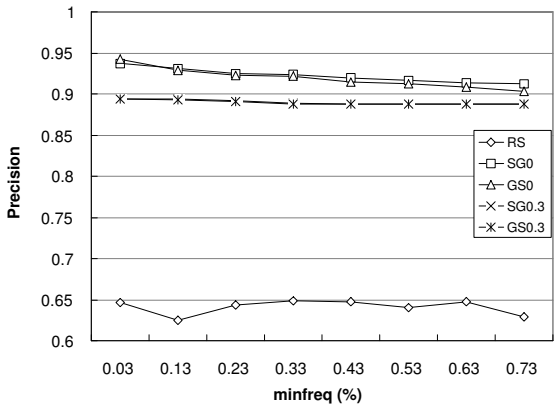

Figure 5: Precision for *top-3* query plans

4. EXPERIMENTS

4.1 Impact of Coverage and Overlap Statistics

We conducted an evaluation of the effectiveness of the coverage and overlap statistics in *BibFinder*[11]. The learned statistics were used in concert with *Simple Greedy* and *Greedy Select* algorithms to select *top-K* sources for any given query. The effectiveness of the source selection was compared against both a random source selection algorithm for selecting *top-K* sources, as well as the actual *top-K* sources for the query (as found by sending the query to *all* the sources, and using a *post-facto* analysis to figure out the real top-K sources). Figure 5 shows the comparative results for top-3 sources. The plans using our learned statistics have high precision, and their precision decreases slowly as we change the *minfreq* and *minoverlap* thresholds (to reduce the amount of stored statistics). The plot here shows that *Greedy Select*, which uses overlap statistics in addition to coverage statistics, does about the same as *Simple Greedy* which uses coverage statistics alone. More evaluation results are available in [8].

4.2 Impact of Multi-Objective Optimization

[11] A prototype version of *BibFinder* which uses coverage and overlap statistics has been developed, and is available at http://sagarmatha.eas.asu.edu

	ACM: coverage, RT	***CSB: coverage, RT***	***DBLP: coverage, RT***	***NET: coverage, RT***	***SCI: coverage, RT***	***Greedy Select plan***	***Discount plan***	***Weighted Sum plan***
q1	0.11 3986ms	0.51 1702ms	0.55 1143ms	0.0 2906ms	0.0 3411ms	DBLP, CSB, ACM	DBLP, CSB, ACM	DBLP, CSB, ACM
q2	0.13 3986ms	0.52 1702ms	0.49 1143ms	0.0 2906ms	0.0 3411ms	CSB, DBLP, ACM	DBLP, CSB, ACM	DBLP, CSB, ACM
q3	0.12 1443ms	0.35 2919ms	0.53 4462ms	0.0 4277ms	0.15 30221ms	DBLP, CSB, SCI	CSB, ACM, DBLP	ACM, CSB, DBLP

Table 1: Plans generated for 3 sample queries using different source utility measures for combining coverage and response time (RT).

Preliminary experiments were also conducted to measure the effectiveness of our multi-objective utility functions. Table 1 shows a sample of three queries for which plans were generated using *Greedy Select* (i.e. not taking into account the response time statistics of the sources), the $utility_D$ function, and the $utility_W$ function. Both the discount γ and the coverage weight δ used in the utility functions were set to 0.5.

All three methods agree on the first query q1, where DBLP has highest coverage as well as best average response time, followed by CSB. For query q2 however, *Greedy Select* will first call CSB because of its high coverage, but the Discount and Weighted Sum utility functions will actually choose DBLP over CSB because for approximately the same coverage, it has a lower response time. For query q3, which is a year range query, all three methods disagree. *Greedy Select* chooses DBLP first even though it has the second to last response time, and even includes SCI in its plan despite the 30221ms average response time for this kind of query on SCI. The multi-objective utility functions agree on the set of sources to call – note that they both eliminate SCI – but not on the order in which to call them. In this case the Weighted Sum function will end up choosing the source with best response time, whereas the Discount function will essentially determine that CSB has a high enough coverage to compensate for not being the fastest source.

Note that the tests described here determined which response time statistics to use based only on whether the query was a range query (since this was the attribute which impacted the response times the most).

5. CONCLUSION

In the process of selecting relevant sources for a given query, a data integration system must have source-specific statistics available. Although several types of statistics can be useful, our approach specifically gathers and uses coverage and overlap information in conjunction with response time statistics, and in doing so greatly improves the source call plans. We are also currently investigating how to include other statistics such as density into our approach and how these could be gathered and used effectively.

Even though response time is generally less query-dependent than coverage, we have seen that the specific bindings of a query can considerably influence the overall response time of a source. By keeping more statistics and refining the query analysis to determine which statistics to use, a better response time estimate could be obtained. One could for instance use the data from Figure 4**c.** to go one level deeper than in our experiments and have different statistics for queries that are not range queries and that bind (or not) the author. Nonetheless, this issue goes back to the traditional tradeoff issue between number of statistics to gather, store, and use, and the performance of the system. More experiments are currently in progress to further determine the impact of our utility functions given different types of queries.

Another issue raised by these query-dependent response time statistics is whether we could consider reformulating a selection query by splitting it on attributes that are not bound, on the basis that, for a particular source, binding specific attributes results in improved response time and/or coverage. The learned statistics could definitely help to determine which attributes should be bound when possible.

6. REFERENCES

[1] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of VLDB*, Santiago, Chile, 1994.

[2] P. Buneman, S. Khanna, and W-C. Tan. Why and where: A characterization of data provenance. In *Intl. Conf. on Database Theory (ICDT)*, 2001.

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

[4] J. Gruser, L. Raschid, V. Zadorozhny, and T. Zhan. Learning response time for websources using query feedback and application in query optimization. *VLDB Journal.*, 9(1):18–37, 2000.

[5] Ulrich Junker. Preference-Based Search and Multi-Criteria Optimization. *AAAI/IAAI 2002*, 34-40.

[6] F. Naumann and J.C. Freytag and U. Leser. Completeness of Information Sources. Workshop on Data Quality in Cooperative Information Systems. 2003.

[7] Z. Nie and S. Kambhampati. Joint optimization of cost and coverage of query plans in data integration. In *Proceedings of ACM CIKM*, Atlanta, Georgia, 2001.

[8] Z. Nie and S. Kambhampati. Frequency-based Approach for Mining Coverage Statistics in Data Integration. In *Proceedings of ICDE*, 2004.

[9] C.H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proceedings of PODS*, 2001.

[10] R. E. Steuer Multiple Criteria Optimization: Theory, Computation and Application. John Wiley, New York. 1986.

[11] Qiang Zhu and Per-Ake Larson. Developing regression cost models for multidatabase systems. In *Proceedings of PDIS*, 1996.