ASU CSE TR ; June 2006.

Multi-Objective Query Processing for Data Aggregation

Jianchun Fan & Subbarao Kambhampati

Department of Computer Science and Engineering

Arizona State University

Tempe, AZ 85287, USA

email: {jcf,rao}@asu.edu

Abstract

Most fielded data integration systems focus on data aggregation applications, where individual data sources all export fragments of a single relation. Given a query, the primary query processing objective in these systems is that of selecting the appropriate subset of sources so as to optimize various user objectives regarding the completeness and quality of the answers and the response time. In this paper we consider three specific objectives: coverage, density and latency. To handle the often conflicting nature of these objectives, we introduce a joint optimization model for source selection that supports a spectrum of trade-offs between them. We also introduce our techniques in defining query classes for different types of statistics and learning statistics with respect to corresponding query classes from the autonomous data sources. We present comprehensive evaluation to demonstrate the effectiveness of our multi-objective query processing model for data aggregation scenarios.

Index Terms

Data Integration, Multi-Objective Optimization, Coverage, Density, Latency

I. Introduction

The availability of structured information sources on the web has recently led to significant interest in query processing frameworks that can integrate information sources available on the Internet. Data integration systems [13], [21], [22], [28], [31] provide a uniform interface to a multitude of information sources, query the relevant sources automatically and restructure the information from different sources.

Many fielded data integration systems focus on "data aggregation" applications (such as comparison shopping systems, bibliography mediators) where individual data sources all export possibly overlapping fragments of a single relation. In such systems, users pose selection queries on the mediator schema, which will be answered by accessing individual sources. The obvious approach — of relaying each query to all the sources known to the mediator and collating the answers — can quickly get out of hand. This is due to the limitations on mediator and data source capacity, network resources, users' tolerance of execution latency, and so on. Therefore in data aggregation systems, the primary query processing task of the mediator is to select an appropriate subset of sources that are most relevant to a given query and the specific user objectives.

Source selection in data aggregation system needs to be adaptive to both the characteristics of the sources and the objectives of the users. Often the users have multiple and possibly conflicting objectives. Examples of such objectives include coverage objectives, cost-related objectives (e.g. response time [17]), and data quality objectives (e.g. density [25]). For example users may like to have many answers (high *coverage*) that are of good quality (high *density* or fewest attributes with missing values) and be retrieved fast (low *latency*). However these objectives are often conflicting: the data sources that respond fast may not have good coverage, the sources that give many answers may have poor answer quality, etc. The following two example scenarios further illustrate some of the challenges here:

1) In an online bibliography aggregation system such as *Bibfinder* [3], a user may not care about whether *all* possible bibliography entries regarding her query are retrieved. She may be satisfied to get most answers back without waiting for a long time. Unfortunately, the sources that provide the most number of answers may not be the ones that respond with the shortest delay. Therefore, the mediator needs to make trade-offs between the coverage and latency objectives when selecting sources.
2) In an online used car trading aggregation system, a user who is looking for a used car may not only want many query results, but they may also want the individual answers to be of good quality (e.g. provide as much detailed information about the cars — body style, configuration, price range, accident history, maintenance history availability, etc. — as possible). Often sources that are populated either by automated information extraction techniques, or by direct user entry, tend to have many tuples but also tend to have a high percentage of missing (null) values in the tuples. Most users also prefer the query to be processed quickly. In this scenario, the mediator has to accommodate the possibly conflicting objectives of coverage, density and latency.

In both the scenarios above, different users may have very different preferences in terms of the importance of the different objectives. For instance, some may want to have a few answers really fast, and others may be willing to wait a little longer for more answers or for better quality answers, etc. This requires the meditator to be adaptive to such user preferences and support trade-offs among multiple objectives. There are two broad challenges in satisfying these requirements: (i) how to estimate the quality of different query plans (source selections) w.r.t. the various objectives, and (ii) how to select the optimal query plans, given the quality estimates.

Estimating the quality of a query plan: To estimate the quality of a query plan, the mediator needs several types of source- and query- specific statistics. For example, it would need coverage, overlap, density and latency estimates for various sources and source combinations.

Unfortunately, the autonomous and decentralized nature of the data sources constrains the mediators to operate with very little information about the structure, scope, content, and access costs of the information sources they are trying to aggregate.

Finding Plans Optimal w.r.t. Multiple Objectives: Even after estimating the quality of the candidate query plans (sets of selected sources), the mediator still needs to decide which specific query plan(s) to execute in response to the user's query. The classical approach to supporting multi-objective optimization is to compute the pareto set of solutions. The pareto set has the property that no solution in it is dominated by any other solution across all objectives. An instance of this approach, going under the name of *skyline query processing*, has been used to support database tuple retrieval under multiple conflicting selection criteria [10]. Unfortunately, the approach is often infeasible for selecting optimal query plans. Specifically, we will see that the pareto sets can often contain a large number of query plans, and we cannot expect lay users to be able to select the appropriate query plan from such large sets. An alternative approach is to bundle the preferences over multiple objectives into a single "utility" metric. The challenge here lies in finding appropriate models for defining utility in terms of the various objectives.

Our Approach: We have been developing a source- and user- adaptive query processing framework for data aggregation that addresses the challenges discussed above. Figure 1 shows the architecture of this framework which supports coverage, density and latency query objectives.

For estimating the quality of query plans, we developed methods to automatically learn coverage, overlap, latency and density statistics of the sources. To keep the learning tractable while maintaining the accuracy of the estimates, these statistics are learned and stored with respect to query classes (rather than individual queries or entire sources). The definition of query classes depends on the type of statistic being learned (as we shall see). While our methods for learning density and latency statistics are new, the approach for learning coverage and overlap statistics is borrowed from our previous work [29]. We learn density statistics with respect to

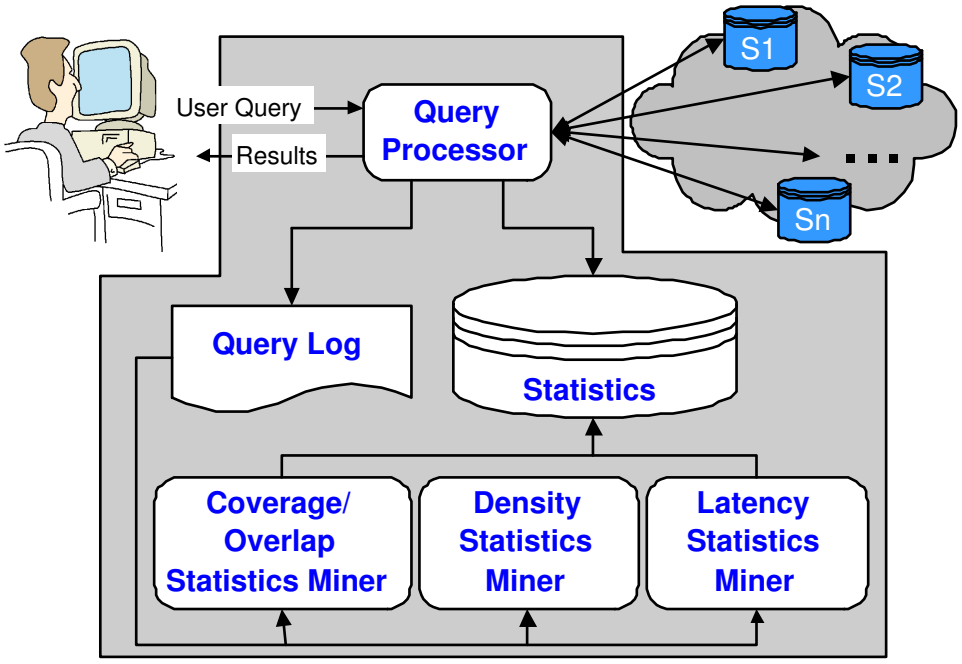

Fig. 1. The Architecture of the Adaptive Data Aggregation.

the *projection attribute sets* of the queries (see below), and learn latency statistics with respect to the *binding patterns* of the queries.

To support optimization, we develop models to combine the coverage, latency and density into an aggregate quality (utility) metric. Specifically, we combine coverage and overlap statistics into "residual coverage." The residual coverage and density are combined into a statistic called 2D-coverage. This 2D-coverage is then combined with the latency statistic using a discounted model that weights tuples expected from a source in terms of the latency associated with that source. The combinations are done in a parameterized fashion allowing the user ample flexibility to change the relative importance associated with the various objectives.

Evaluation: We empirically evaluate the effectiveness of our multi-objective query processing approach. The evaluation is performed both on controlled sources and in the context of *Bibfinder* [3], an online bibliography aggregation system. Our experiments show that our query class based statistics learning model is able to effectively capture the unique characteristics of each type of statistics, and the source selection procedure is able to make flexible trade-offs among multiple

objectives during query processing.

The rest of this paper is organized as follows. In the next section, we will discuss some of the related work in multi-objective query optimization and source statistics learning in data integration literature. Section III introduces the concepts of coverage, density and latency statistics and our methods of learning them. Section IV focuses on our joint optimization model that uses these statistics to make flexible trade-offs between multiple objectives. An empirical evaluation of our framework is presented in Section V. We will conclude the paper and discuss some future work in Section VI.

II. Related Work

Several research efforts have focused on effective query processing in data integration scenarios [22], [20], [21], [31], [27], [33]. Most of these efforts however have concentrated on the "horizontal" aspects of data integration, where individual sources export different relations of the mediator schema. The emphasis is on supporting joins across multiple sources. Since a majority of the fielded data integration systems have been of the "data aggregation" variety, our work focuses on selecting sources appropriately in such scenarios.

Multi-Objective Optimization: Multi-objective query planning has received very little attention in data integration literature. Some exceptions include the work by Nauman et. al. [26]. Although they focus on multiple quality metrics for the plan, our work differs from theirs in significant ways. First, we focus on automatically computing the quality metrics through source statistics learning. Second, unlike their work which defines utility as a simple inner product of weight and quality vectors, we consider more sophisticated models for combining quality metrics into utility metrics. Finally, we provide a systematic evaluation of our techniques in both controlled and real-world data aggregation scenarios.

Some efforts in data integration literature attempt to handle multi-objective query planning

through decoupled strategies. For example, the *Information Manifold* system [22], [12], [20] attempts to optimize coverage of the query plans first, and then the response time. Earlier work by Nie and Kambhampati [27] showed that such decoupled strategies can lead to highly suboptimal plans, and motivated join optimization strategies. In contrast to [27], our work focuses on a larger variety of statistics–coverage, overlap, density and response time, and on automatically learning the statistics.

In contrast to multi-objective query planning, multi-objective retrieval has received more attention in the database literature [10]. The work here, going under the name of "sky-line query processing" focuses on efficiently computing the pareto-optimal set of database tuples given a set of selection functions. Other researchers, including Agrawala and Wimmers [9] have focused on frameworks for supporting expression and combination of user preferences.

Learning Coverage/Overlap Statistics: Some existing efforts consider the problem of text database selection [19], [34] in the context of keyword queries submitted to meta-search engines. To calculate the relevance of a text database to a keyword query, most of the work ([16], [35], [23], [11]) uses the statistics about the document frequency of each single-word term in the query. The document frequency statistics are similar to our coverage statistics if we consider an answer tuple as a document. Although some of these efforts use a hierarchy of topics to categorize the Web sources, they use only a single topic hierarchy and do not deal with computation of overlap statistics. In contrast, we deal with classes made up from the Cartesian product of multiple attribute value hierarchies, and are also interested in modeling overlap. This makes the issue of storage space consumed by the statistics quite critical for us, necessitating our threshold-based approaches for controlling the resolution of the statistics.

Learning Source Density Statistics: Naumann [24] introduces the concept of density that is used to measure how well the attributes stored at a source are filled with actual values. We

use a similar definition of density in our work, however we propose methods to effectively and efficiently *learn* query-dependent density statistics. Our 2D coverage model is similar to the completeness measure proposed by Naumann [24] which combines coverage and density. However, our model uses a configurable parameter to allow users to set their own preferences on coverage and density.

Learning Source Latency Statistics: There has also been previous work on learning response-time statistics in data integration literature [17], [33]. [36] discusses collecting and managing time-dependent latency distributions between individual clients and servers for wide area applications. Our approach differs from these efforts by considering the impact of the query pattern on the source latency value and acquiring finer granularity source latency estimation.

III. Learning Source Statistics

To select sources with respect to coverage, density and latency objectives during query processing, the mediator needs corresponding source statistics to make an approximate estimate of the degree of relevance for each source in terms of the given objective(s). In this section we introduce these statistics and how the mediator can automatically learn them from the autonomous data sources.

To learn each type of statistic for every possible query is too costly (the space of all possible queries is exponential to the number of attributes) and usually infeasible. The general statistics learning model we advocate is *to group individual queries into query classes and learn statistics with respect to query classes*. The advantage of this approach is two fold.

1. The mediator can manage the number of query classes to learn statistics on by making query classes more abstract (less accurate statistics) or more specific (more accurate statistics), depending on the learning and storage cost it is willing to pay, and the degree of statistical accuracy it wants to achieve.

2. With statistics on query classes, the mediator may be able to estimate the statistics for new unseen queries by using the statistics of the query classes into which these queries are classified.

The interesting challenge here is that the definition of appropriate query class may well depend on the type of statistics.

A. Coverage/Overlap Statistics

We leverage the existing work of *StatMiner* [28], [29] on frequency based source coverage/overlap statistics mining. *StatMiner* automatically discovers frequently accessed query classes and learns source coverage/overlap statistics with respect to these query classes. We will briefly review the basic concepts and algorithms here to the extent they are needed to understand our contributions. For more detailed information, we refer the readers to to the discussion in [28].

The mediator in *StatMiner* keeps track of a query log which contains the past user queries. For each query it keeps information of how often it is asked and how many answers came from each source and source set. The query log is used in learning source statistics.

1) Coverage and Overlap: In data aggregation scenarios, the naive way of answering user queries is to direct each query to all the data sources, collect the answers, remove the duplicates and present the answers to the user. This will increase the query processing time and tuple transmission cost as well as the work load of the individual sources. A more efficient and polite approach is to send the queries to only the most relevant sources.

To figure out which sources are the most relevant to a given query Q, the mediator needs the *coverage* statistics for each source with respect to Q, i.e. $P(S|Q)$, the probability that a random answer tuple for query Q belongs to source S. The source with the highest coverage value for Q will be the first source to be called. Considering the overlap between data sources, the second source to be called will be the source that has the largest *residual coverage*, which provides the maximum number of answers that are not provided by the first source. In other words, the

mediator needs to find the source S' that has the next best coverage but minimal overlap with the first source S. Thus it needs the *overlap* information about sources S and S' with respect to query Q, i.e. $P(\{S, S'\}|Q)$, the probability that a random answer tuple for query Q belongs to both S and S'.

2) AV Hierarchies and Query Classes: If we have coverage statistics for each source-query combination as well as the overlap statistics of each possible source set with respect to each possible query, we are able to compute the optimal order in which to access the sources for each query. However this is unrealistic because the cost to learn and store such statistics with respect to every possible query is extremely high (exponential in the number of data sources and linear in the number of possible queries, which is an enormous number). To keep such costs low, *StatMiner* learns and keeps statistics with respect to a small set of "frequently asked query classes" instead of individual queries. The idea is to group the queries into query classes and only learn coverage/overlap statistics for those frequently accessed query classes. When a new query is issued on the mediator, it is mapped to the most relevant frequent query classes for which statistics are kept, and these statistics are then used to estimate the coverage/overlap information of sources with respect to the new query.

To group the queries into query classes, *StatMiner* classifies them in terms of their bound attributes and their binding values in selection queries. To further abstract the classes it constructs "attribute value hierarchies" for the attributes of the mediated relation. An "attribute value hierarchy" (AV hierarchy) over an attribute A is a hierarchical classification of the values of the attribute A. The leaf nodes represent the individual concrete values of A and the non-leaf nodes represent the abstract values that correspond to the union of the values of their descendants. Figure 2 shows two very simple AV hierarchies for the $conference$ and $year$ attributes of the $paper$ relation for the *Bibfinder* scenario. The actual AV hierarchies are much more complicated. For example, there are more than $9,500$ nodes in *author* hierarchy and around 800 nodes in

conference hierarchy. *StatMiner* has a method to automatically construct these AV hierarchies. This method essentially uses the agglomerative hierarchical clustering algorithm [18] to build a hierarchical classification of each attribute. The distance measure used in this clustering algorithm is the cosine distance of the vectors which represents the query result distribution information (among the sources) for each individual attribute value.

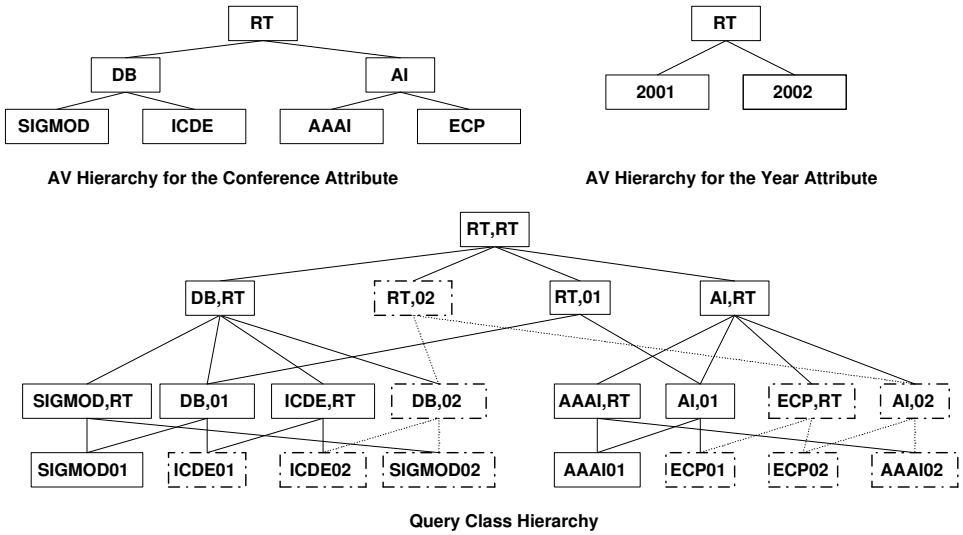

Fig. 2. AV hierarchies and corresponding query class hierarchy.

Since the selection queries on the mediated relation have one or more attributes, *StatMiner* groups them using the AV hierarchies of the attributes. A query feature is defined as the assignment of a classification attribute to a specific value from its AV hierarchy. Sets of features are used to define query classes. Specifically, 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 for each of the classification attributes. For example Figure 2 shows a class hierarchy for a simple mediator with AV hierarchies for two classification attributes. 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 classification attributes in Q are equal to, or are specializations of, the features defining C.

3) Learning and Using Coverage/Overlap Statistics: Since query classes represent an abstraction of the queries that belong to them, *StatMiner* uses the statistics of the query classes as the approximation of individual queries and thus cuts the cost of learning and storing the statistics. A query submitted to the mediator can be mapped to a group of query classes to which it belongs. The *access frequency* of a query class is defined as the number of queries that are mapped to the query class during a given period of time. A *frequent query class* is a query class that has an access frequency above a given threshold. In our previous work [28], we have proposed methods to automatically identify the *frequent query classes* from the entire query class hierarchy (the Cartesian product of all AV hierarchies) based on the query log. The coverage/overlap statistics of the individual queries that are mapped to the frequent query classes are then aggregated to represent the class statistics.

After the source coverage/overlap statistics with respect to the frequent query classes are gathered, they can be used in source selection for future queries. Specifically, when a new query is submitted to the mediator, it is mapped to one or more *frequent query classes* according to its binding values on the classification attributes. The statistics of these frequent query classes are combined to estimate the coverage/overlap information for the new query so that the mediator can select the most relevant sources accordingly.

This approach has been demonstrated to be very efficient and effective in coverage/overlap statistics mining. These statistics have been shown to be useful in optimizing the overall coverage during source selection.

B. Density Statistics

In data aggregation systems, incompleteness at the database record level is also inevitable. First, web databases are often input by individuals without any curation. For example, web sites, such as *cars.com*, and *autotrader.com*, obtain car information from individual car owners, who

may or may not provide complete information for their cars, resulting in a lot of missing values in databases. Second, data aggregation systems usually assume a single global relation, but in many cases, not all the attributes are supported by every individual source. For example, a global schema for used car trading has an attribute *body style*, which is supported in *autotrader.com*, but not *cars.com*.

As a result, when a query is posted to the mediator schema and dispatched to the individual sources, the answers returned from different sources are usually not equally good in their quality - some sources may give better answers that have very few missing values, and others may return answers with many missing values, or even have certain attributes missing all together. Naturally such properties have to be captured by the mediator, so that the mediator can select the sources that give answers with fewest missing values. Source density can be used to capture such database tuple level completeness, or "horizontal completeness".

The density of a tuple is simply the fraction of its attribute values that are not missing. Similar to the definition proposed by Naumann [24], the density of a source with respect to a given query can be defined as the average of the density of all answer tuples. We make an assumption that whether an attribute value of a tuple is missing is independent to its actual value [14]. Therefore, the density of source S on a query Q is independent of the selection predicates in Q[1]. Thus we can classify queries with respect to their projection attributes. Queries with the same set of projection attributes share the same density value for a given source S, no matter what their selection predicates are.

We define the density of a data source for a given query class (projection attribute set) as

[1]Admittedly this assumption does not always hold. For example, in a used car data aggregation system, suppose one of the sources is specialized in Toyota cars. For the queries that are requesting Toyota cars, this source will have high density on the *make* attribute but zero density for queries that are looking for cars made by other manufacturers. However in most cases where each source covers a reasonably large portion of the domain of each attribute and follows similar "generic model", this assumption approximately holds.

follows. Let $qc = \{A_1, A_2, \ldots, A_n\}$ be a set of attributes and S be one of the data sources in the aggregation system. For any given tuple[2] $t \in qc$, the density of t is the number of missing values in $\{t(A_1), t(A_2), \ldots, t(A_n)\}$ divided by n. Different tuples in qc may have missing values on different attribute(s). Obviously there are 2^n possible *density patterns* of tuples in qc, in terms of which attribute values are missing. For example, a tuple may belong to the density pattern $(A_1, \neg A_2, A_3)$, meaning that it has concrete values on attributes A_1 and A_3 but is missing the value on attribute A_2. For all tuples that belong to a same density pattern dp, source S has the same density $density(dp)$ which is simply the number of missing attribute values divided by the total number of attributes in the projection attribute set. Let $DP(qc)$ be the set of 2^n density patterns, if we can assess the probability of the appearance of each of the 2^n density patterns $P(dp)$, then we can define the overall density of source S for query class qc as: $density(S|qc) = \sum_{dp \in DP(qc)} P(dp) * density(dp)$

This model essentially uses the weighted average of the 2^n density patterns to measure the overall density of the given projection attribute set, where the weight of each density pattern is the likelihood of it being appearing in the database.

For a database relation $R(A_1, A_2, \ldots, A_n)$, the set of all possible projection attribute sets is the power set of $\{A_1, A_2, \ldots, A_n\}$ which has 2^n members. For each projection attribute set, suppose m is the number of attributes contained in the set, then we need to assess the joint probability of all the 2^m density patterns. Overall, the number of joint probability values we need to assess will be $\sum_{1 \leq m \leq n} \binom{n}{m} 2^m$, which is a significantly large number. By making an independence assumption among database attributes [14], we reduce the problem down to assessing the n probabilities of each attribute having a missing value, and *computing* the joint probability values based on the independent assumption. Collecting these n probability values is straightforward. Based on Assumption 1, as long as the mediator has a certain amount of sample records from each source,

[2]A tuple $t \in qc$ means that t is one answer tuple for a query Q that belongs to query class qc.

it is able to figure out these probability values easily. During the bootstrap stage, the mediator can simply dispatch user queries to all the sources instead of a selected subset (at this stage, not enough source statistics are available for smart source selection anyway). The results returned from each source are stored as sample records of that source. The density statistics can then be collected once there are a reasonable amount of sample records for each source.

C. Latency Statistics

In data aggregation scenarios, the latency (response time) of a data source is the time needed to retrieve the answers from the source. Naturally, with latency statistics, the mediator is able to select the fastest sources to call to satisfy the users who want get to *some* answers as quickly as possible.

In data aggregation scenarios, sources on the internet often differ significantly in their latency. For example, we sent 200 random queries from the *Bibfinder* query log to all the sources, and their average response time varied from 1100 milliseconds to 6200 milliseconds [3]. Some existing work has been done in learning source latency profiles. For example, [17] defines latency (response time) as a source specific measure with some variations across several dimensions (such as time of a day, day of the week, quantity of data, etc.). Such a definition assumes that under similar circumstances, the execution latency of a source is *query insensitive* and different queries submitted to a source have similar latency values. However, we found that such an assumption is questionable in many cases. For example, when we monitored queries on *Bibfinder*, we found that the same data sources sometimes give very different latency values for different queries. There are many obvious reasons for such variance. For example, the source database may have indices on some attributes but not on others, and thus selection queries on the indexed attributes can be much faster than the ones on the other attributes.

[3]We use *latency* and *response time* interchangeably throughout this paper.

This observation suggests that to accurately estimate latency for the new queries, *the mediator has to learn the latency values in a query sensitive manner*. In other words, for any single data source, we have to learn latency values for different types of queries. This leads to the next challenge: how to properly classify the queries so that queries within the same category have similar latency values for a given data source.

Based on our observations, query execution latency is usually less sensitive to the binding values[4], but rather more dependent on the *binding pattern*[5] of the queries. We define the time required for a web data source to export the first page of answers as its latency value. Some sources may export answers in multiple pages, but the time to retrieve the first page (which includes query processing time of the sources and the time for the mediator to parse the returned answer pages) is very small compared to the overall time. Obviously, for users who care less about the completeness of the answer, the speed of retrieving the first page is more important. Moreover, when comparing different queries with the same binding patterns on the same data source, we found that the time to retrieve the first page is reasonably stable. Thus, we decided to take this time as the latency value since it is more likely to be a good estimate for future queries. Admittedly, different data sources choose to export different number of answer tuples within each page. Thus, the time for the mediator to parse those pages would be different. However, such differences in the size of pages can be ignored, as we noted that the source-side query processing time almost always significantly dominates the mediator-side parsing time. In other words, we consider the answers within the same page to have same latency values. As

[4]Naturally, for selection operation on certain attributes, different binding values may result in different number of answer tuples and thus the transmission cost from the source to the mediator may be different. However we will discuss later that such difference is very small compared to the query processing and connection setup cost.

[5]Binding pattern of a selection query describes whether each attribute is bound to a concrete value in the query, and if so, how it is bound (equality, range, etc). For example, in the *Bibfinder* scenario, query $q1($"$data\ integration$", _, "$VLDB$", _) and $q2($"$planning\ graph$", _, "$AAAI$", _) have same binding patterns, but they have different binding patterns from query $q3($_, "$EF\ Codd$", _" < 1980").

shown in Figure 3, queries with different binding patterns tend to have very different latency values. For queries from same binding pattern, their latency values may or may not be close to each other, depending on what concrete values are bound in the queries. We can always refine this classification of queries to get finer latency values, but our experiments in the *Bibfinder* scenario shows that the actual data sources we integrate have rather stable latency values for queries with the same binding pattern, regardless of what the bound values are, as shown in Figure 4. Therefore, we decided to learn source latency statistics based on the binding patterns of queries.

Our method for learning the latency statistics is relatively straightforward. For each binding pattern, we measure the average response time of a source given a group of testing queries which conform to a specific binding pattern. We take the average response time over the set of test queries as the latency value for this [source, binding pattern] combination. Note that if there are n attributes and we only care whether or not each attribute is bound, there will be 2^n number of binding patterns. In many applications, n is rather small and we can learn latency statistics for all 2^n binding patterns. However, in cases where n is large, it will be an interesting problem to automatically learn which of the binding patterns show more latency difference and are worth learning and keeping statistics for.

Our experiments on the *Bibfinder* test bed validates our model of defining the "query class" of latency statistics with respect to the binding pattern of the queries. In fact, our experiments, given a random query set, show that when the learned latency statistics are used to predict the expected latency for new queries, the prediction has an estimation error ($\frac{|estimatedlatency - reallatency|}{reallatency}$) which is less than 20% in most cases.

The major difference between our latency statistics learning model and most of the existing work is that we collect *query sensitive latency* values for each [source, binding pattern] combination, instead of more coarse, *query insensitive* source latency values. As shown in Figure

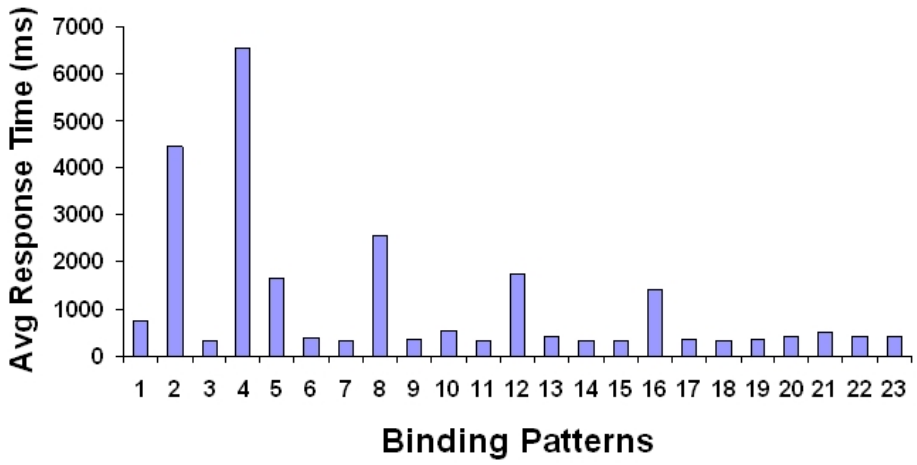

Fig. 3. The average latency of the source DBLP shows significant variance between different binding patterns.

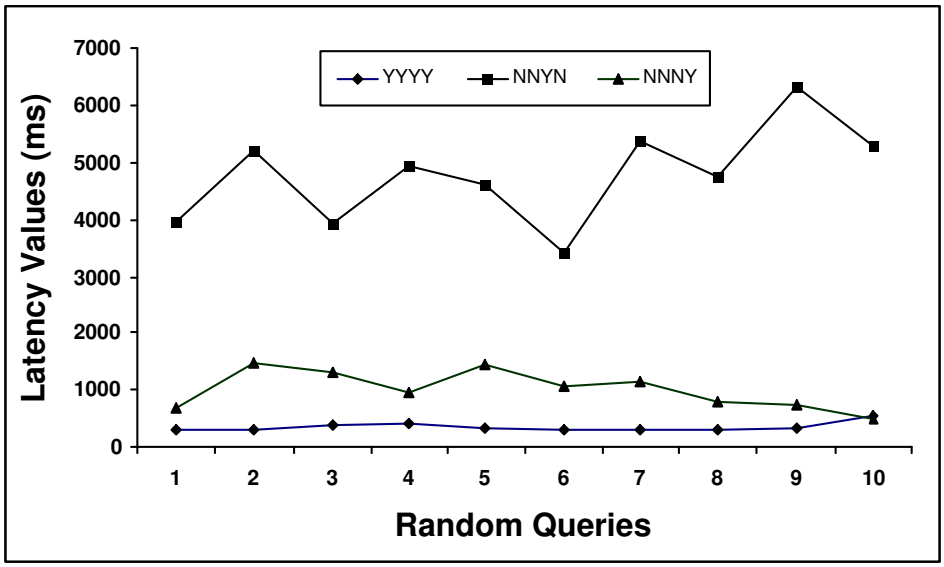

Fig. 4. The latency values for the source DBLP with 10 randomly selected queries for each of the 3 different binding patterns. The queries that fall in same binding pattern have similar latency values.

5, our approach estimates the latency values much better and more accurately identifies the fastest data sources.

D. Summary of Statistics Learning Model

In summary, our framework learns all of the statistics with respect to the query classes. Our general statistics learning model focuses on appropriately defining query classes for different

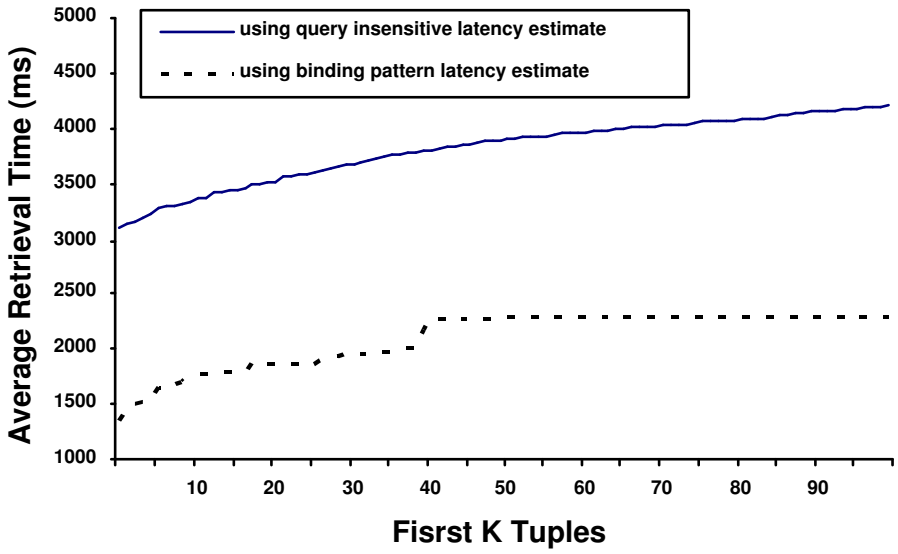

Fig. 5. By using query sensitive latency values based on binding patterns, our approach identifies the fastest sources for different queries more accurately and optimizes the retrieval cost much better.

types of statistics.

1) For the coverage statistics, the query classes are defined as the Cartesian product of attribute value hierarchies according to the binding value found in the selection predicates of the queries.
2) For the density statistics, the query classes are defined in terms of sets of projection attributes within the queries.
3) For the latency statistics, the query classes are defined in terms of the binding patterns within the queries.

When a query is submitted, it is mapped to appropriate query classes with respect to each type of objective. The statistics we learn for these query classes are used as estimates for the new query to guide the source selection.

Incremental Maintenance: In the preceding sections, we focused on how to learn the source statistics. A related issue is that of maintenance. Given that the data sources on the web are

autonomous and can change without notice, source statistics can become stale if they are not actively maintained. This can happen either because of the change of the user interests, change of the source contents or addition/deletion of the sources accessible to the mediator. Our statistics learning framework is highly amenable to incremental maintenance technqiues, and we have in fact developed and evaluated strategies for incremental maintenance. Because of the way query classes are defined for the different statistics, density and latency statistics are relatively straightforward to maintain, while coverage and overlap statistics present more interesting technical challenges. For the latter, we develop an incremental approach where statistics are calibrated with respect to the most recent window of user queries. Space considerations preclude a complete discussion of these results; interested readers are referred to [14, Chapter 7].

IV. Multi-Objective Query Optimization

As discussed in the last section, by using coverage/overalp, density and latency statistics, the mediator is able to optimize the individual objectives during source selection.

However, in data aggregation systems, the users usually prefer query plans that are optimal for multiple objectives. Unfortunately, these objectives often conflict. For example, sources with high coverage are not necessarily the ones that respond quickly or provide quality answers. As a result, a source selection plan which is optimal with respect to one objective cannot in general be post-processed to account for the other objectives. For example, if for a given query, three data sources have the following coverage and density statistics: S_1($coverage = 0.60$, $density = 0.10$), S_2($coverage = 0.55$, $density = 0.15$), S_3($coverage = 0.50$, $density = 0.50$). If we try to find only one source to call using a decoupled strategy [22], [26], [12], [31], [10], [20], [32], S_3 will be eliminated first because it has the worst coverage, and then S_1 will be eliminated next because it has the worst density. S_2 will be selected although in reality, S_3 may be the overall best source to call, assuming coverage and overlap are equally important.

Therefore, the mediator needs to generate plans which jointly optimize coverage, density and latency so that users can get many high quality answers as soon as possible. Moreover, different users usually have different preferences on the different objectives. For example, some users would rather get a few answers as fast as possible, some may be willing to wait a little longer for higher coverage and others may only like the good quality answers. The query processor in our framework adopts a novel joint optimization model in source selection to resolve conflicting user requirements with respect to coverage, density and latency and make flexible trade-offs between them, using the various statistics gathered.

The goal of multi-objective query optimization is to find the "overall best" query plans in terms of multiple user objectives. For a given query Q, each query plan $p(Q)$ corresponds to a vector in a three dimensional space, with $coverage$, $density$ and $\frac{1}{Latency}$ (since lower latency is preferred) as its three dimensions. The multi-objective query optimization then becomes the problem of picking the best vectors among these vectors. The first principles solution to such a problem is to present all non-dominated vectors (called the "*pareto set*")[6] to the users and let the users select one. Although such a technique is standard in sky-line query processing (c.f. [10]), as we argued earlier, it is not well suited for query planning where the lay users may be overwhelmed by the possibly large number of plans in the pareto set.[7]

A popular alternative is to convert the multi-objective problem to a single-objective problem by aggregating the different dimensions of each vector into a scalar by using a utility function. In this approach, only the plan with the best utility value is presented to the user. The challenge

[6]A vector v_1 is dominated by vector v_2 if v_1 is worse than v_2 in all dimensions.

[7]If there are n data sources and we are looking for query plans containing k best sources, then there are $\binom{n}{k}$ query plans. To see that in the worst case, the pareto set can be as large as $\binom{n}{k}$, consider a scenario where we only have two objectives o_1 and o_2, with o_1 defining a strict total order over the k-size query plans, and o_2 defining a total order which is the opposite of that defined by o_1. In this case, it is easy to each that every pair of $\binom{n}{k}$ query plans are non-dominated by each other. Consequently, *all of them* are in the pareto set. It is also easy to see that as the number of objectives increases, the chance that every query plan will be in the pareto set increases

here is finding appropriate and natural models to combine the various objectives.

Since coverage and density describe vertical and horizontal completeness respectively, we naturally extend the joint measurement between them as “two dimensional coverage” (2D coverage). We represent the set of all possible answers for a given query Q as a table, with each row being an answer tuple and each column an attribute from the projection attribute set of Q. For any source or source set S, the 2D coverage of S for Q is the portion of the table cells that are covered by the answers from S. Some sources may have high 2D coverage without having the highest value on either of the individual dimensions (Figure 6 (c)).

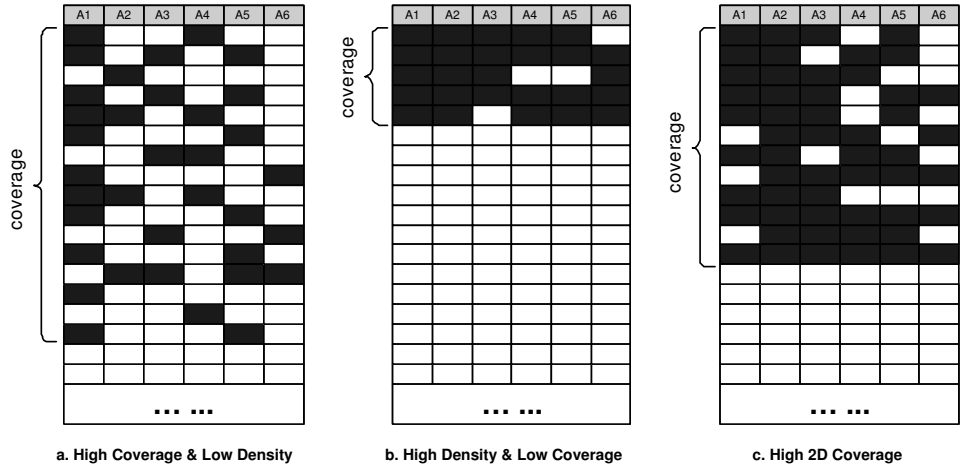

Fig. 6. Joint measurement of coverage and density with 2D Coverage.

2D coverage of a source or source set for a given query represents the area covered by its answers, which can be simply estimated as:

$$2DCoverage(S|Q) = coverage(S|Q) \times density(S|Q)$$

Note that here 2D coverage is exactly the percentage area of the table cells that are covered by the answer tuples of S, assuming that coverage and density are of equal importance to the user. In fact, different users may have different preferences in terms of coverage and density - some may simply prefer more answers and others may like a few “good” answers that each have complete information. Therefore, the above definition can be extended to a more general

form as following:

$$2DCoverage(S|Q) = coverage(S|Q)^{\kappa} \times density(S|Q)^{1-\kappa}$$

We call κ $(0 \leq \kappa \leq 1)$ the *scale factor* in this formula since it is used to scale the measure of coverage and density in computing the 2D coverage. Compared to the completeness model proposed by Naumann [24], our 2D coverage model uses a configurable parameter to allow more flexible trade-offs between coverage and density to reflect users' own preferences.

With the concept of 2D coverage, the problem of jointly optimizing the coverage, density and latency objectives is reduced to combining 2D coverage and latency statistics. The first model we use is a *discount model*, which emphasizes the expected 2D coverage of the data sources, but discounts those estimates with the corresponding latency estimates. Specifically, we use the following source utility model which takes into account both the coverage/overlap, density and the latency statistics:

$$Util(S_i) \quad = \quad ResidualCoverage(S_i|Q)^{\kappa} \times density(S_i|Q)^{1-\kappa} \times \gamma^{Latency(S_i|Q)}$$

In the formula above, for a given query Q, the 2D coverage estimate (combination of coverage and density) of source S_i on Q is discounted by the latency value of source S_i for queries that have same binding pattern as Q. The mediator uses the utility value in a greedy way to select the sources to call. The source with the largest utility value is the first source added to the plan, and then the source with the next largest utility, and so on. Note that to maximize the overall coverage of the plan, we use *residual coverage* with respect to the set of already selected sources as discussed in Section III-A. For the selection of the first source, the *residual coverage* is just the regular coverage value.

The utility function has the *discount factor* $\gamma(0 < \gamma \leq 1)$. When γ is 1, the utility is just a function of the 2D coverage, and the source selection plan generated only optimizes on 2D coverage. By reducing γ, the sources that have shorter latency are favored and thus the plan

is able to jointly optimize both 2D coverage and latency. By adjusting γ, individual users can express their preferences on 2D coverage and latency. In the next section, we will show that this combined utility function does support natural trade-offs between multiple objectives.

One minor problem with this model is that it is slightly asymmetric. We can optimize only the 2D coverage by setting the discount factor γ to be 1, but there is no way that we can optimize only the latency (although when $\gamma \approx 0$, it does essentially optimize the latency). We also looked at another model which uses a weighed sum utility function [27]:

$$\begin{aligned} Util(S_i) &= \omega \times \log(ResidualCoverage(S_i|Q)^{\kappa} \times density(Si|Q)^{1-\kappa}) \\ &\quad + (1-\omega) \times (-Latency(S_i|Q)) \end{aligned}$$

This *weighted sum* model simply combines the 2D coverage measure and latency measure. We take logarithm of the coverage value so that it is comparable to the latency value and we take the negative of the latency value to reward the faster data sources. The resulting utility function can easily adapt to the full spectrum of optimization from "2D coverage only" to "latency only" or somewhere in-between. Next we will show that both models can easily accommodate the trade-off between the 2D coverage and latency objectives and optimize the query plan in all three dimensions.

A. Parameter Estimation

Our model uses configurable parameters to accommodate different users' preferences on the importance of coverage, density and latency during source selection. In our current work, we provide users sliding scales to set the values of these parameters. It can be argued that while the users may have a qualitative understanding of the impact of these parameters on the query plans, it is hard to expect them to have the quantitative understanding needed to make fine adjustments to the parameters. An interesting future direction would be to adapt machine learning methods

such as [15] to learn these preference parameters through user interactions.[8]

V. Experimental Evaluation

We have experimented with a controlled test bed and a real *Bibfinder* data aggregation test bed to see if our joint optimization model can make reasonable trade-offs between coverage, density and latency objectives.

A. Experimental Set Up

1) Controlled Test Bed: The controlled synthetic test bed we used in the evaluation is set up in the following way. We used the Yahoo! autos database as the base data which has around 100,000 used car records. The schema of the database is *cars(make,model,year,price,milage,location,color)*. We set up 30 artificial data sources, each of which contains a randomly selected portion of the car database, ranging from 10% to 50% in size. To introduce horizontal incompleteness, each attribute of each database had values removed at a random percentage, ranging also from 10% to 50%. This way the sources all have different coverage and density features for any given query.

In data aggregation scenarios, the mediator usually only has access to a small portion of the data from the autonomous sources for the purpose of knowledge mining. Therefore, we used a 5% random portion of each of the 30 databases as their sample to learn the density statistics with the techniques presented in Section III. As for coverage statistics, since we did not have a query log for the synthetic test bed, we were not able to apply the techniques proposed in our previous work [29]. Instead, we used a simplified way to collect coverage statistics by querying

[8]One approach would be to have an off-line training phase where representative users are shown plans from the pareto set, estimates of their coverage, density and latency, as well as the results obtained by executing those plans. The users are then allowed to choose a plan, and regression methods are used to fit the parameters of the utility model to make it conform to the user choices.

the sample databases and using the coverage/overlap values as the estimated coverage statistics. Specifically, for a given query, the mediator first directs it to the samples of all of the data sources to figure out the coverage and overlap information by looking at the query results from each of the samples. This information is then used as an estimate of the real coverage/overlap statistics of the actual sources in the source selection.

To simulate the source latency, each of the sources is given a randomly generated latency value, ranging from 100 to 8000 milliseconds. The sources intensionally make a delay before they send query results back to the mediator[9].

2) Bibfinder Test Bed: We have also partially experimented with the real *Bibfinder* test bed. *Bibfinder* aggregates several computer science bibliography sources including *ACM Digital Library* [1], *ACM Guide* [2], *Network Bibliography* [7], *IEEE xplorer* [6], *DBLP* [5], *CSB* [4] and *Sciencedirect* [8]. *Bibfinder* has a global schema $Paper(title, author, conference/journal, year)$. For this test bed, we learned source coverage/overlap statistics with techniques proposed in [29] by using the real user queries logged in *Bibfinder*. These user queries are also used in probing the individual sources to learn the binding pattern based source latency statistics. In *Bibfinder*, some sources do not export some of the attributes found in the global schema, which indicates a lower density on the projection attribute set that contains those missing attributes. However, the attributes that a given source does export are usually complete, i.e, the tuples usually do not have missing values on these attributes. Therefore, in the *Bibfinder* scenario, there was not much of an interesting variation in terms of the source density statistics for user queries. Thus, for this test bed, we could only evaluate the joint optimization of coverage and latency. This is done by setting the density component as a constant 1 in both of the utility functions.

[9]Since the focus here is to evaluate the joint optimization model, we use same latency statistics for different binding patterns. The binding pattern based latency statistics are used in our evaluation on the *Bibfinder* test bed.

B. Evaluation Results

1) Evaluation Results on Controlled Test Bed: We first evaluated on the synthetic test bed to determine whether the 2D-coverage model is a reasonable and flexible way to combine coverage and density statistics. We first chose a group of test queries, then generated source selection plans with 2D-coverage statistics and executed these plans. We used the results to compare the average coverage and average density measurements using different scale factor values. As shown in Figure 7, when the scale factor value increases, the sources with high coverage are more favored during source selection and the coverage measure of the answers increases correspondingly. Similarly, with the decrease of the scale factor value, the sources with high density are more favored and thus the density of the answers also increases.

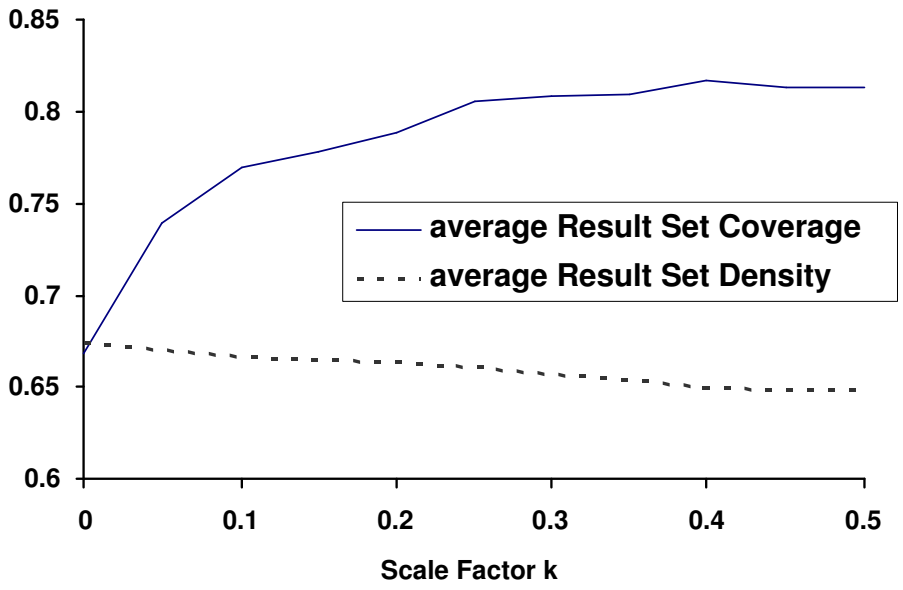

Fig. 7. Supporting flexible trade-offs between coverage and density.

To evaluate the joint optimization model, we first chose 20 test queries on the car database, then made query plans using discount utility functions, and finally executed the plans recording the time needed to retrieve the first K tuples (K=1, 2, 3, ...). We varied the discount factor γ to see its influence on query planning. Figure 8 shows the average time for retrieving the

first K tuples using the 20 testing queries together with their average *2D coverage* for varying discount factor values. Figure 9 shows the results of similar experiments using the weighted sum model. In both experiments, when the discount factor γ (or weight factor ω) is 1, we have a plan that only uses coverage/overlap and density statistics without considering the latency. Thus, on average the speed of retrieving the first K answers is the lowest (because the faster sources are not rewarded for being faster) and the average *2D coverage* is highest. On the other hand, when the discount factor (or weight factor) decreases, the 2D coverage estimates for each of the sources are increasingly discounted with respect to the latency. As a result, the plans generated tend to favor the faster sources, while the average *2D coverage* is getting lower. In addition, when the weight factor ω in the weighted sum model is set to 0, the mediator effectively ignores the coverage and density statistics and only looks for the fastest sources. Our experiment shows that such plans retrieve the first K tuples fastest on average.

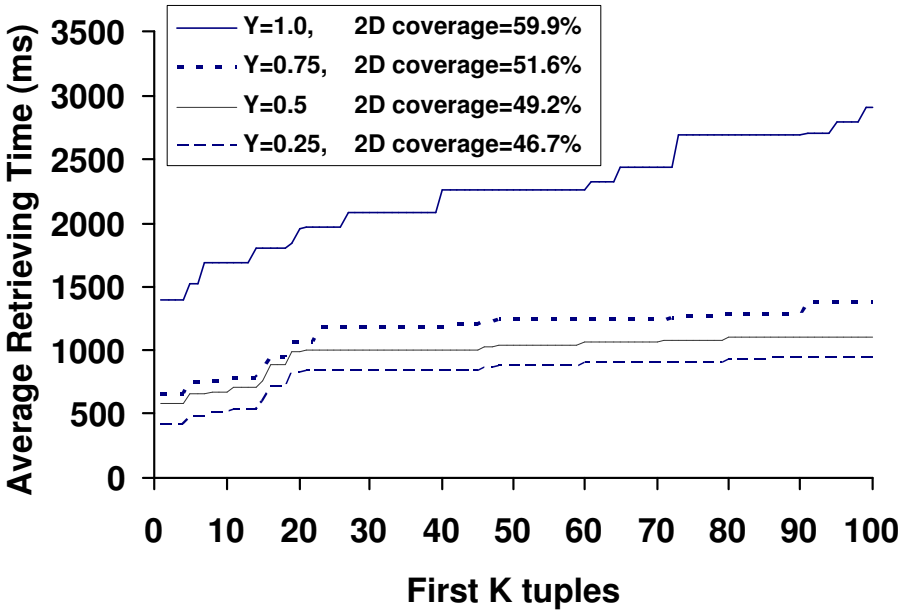

Fig. 8. Combining 2D coverage and latency with different discount factor values, using discount model on controlled test bed.

We can plot the query plans in Figure 8 and Figure 9 in the two dimensional space of

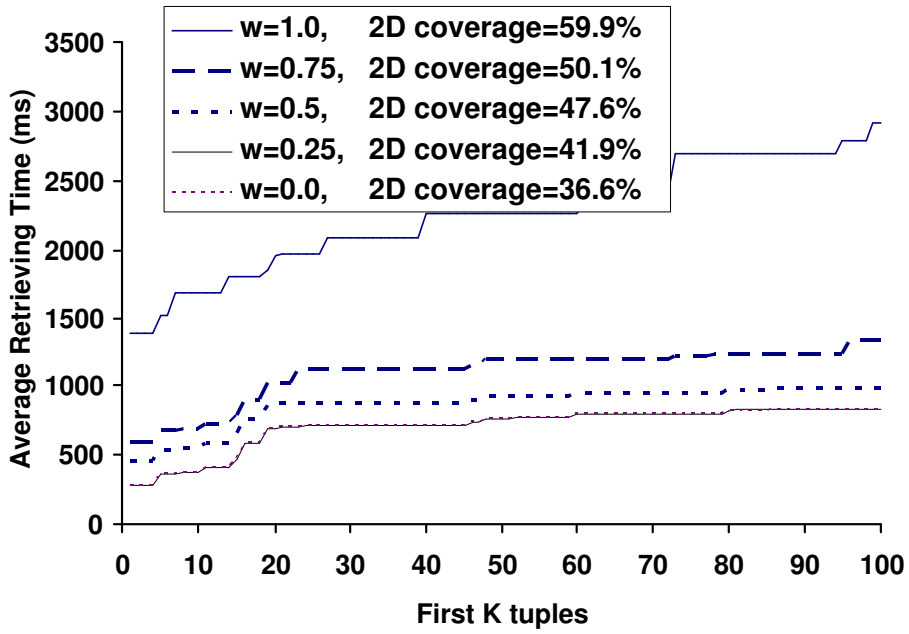

Fig. 9. Combining 2D coverage and latency with different weight factor values, using weighted sum model on controlled test bed.

$average\ 2D\ coverage$ and $\frac{1}{Average\ Latency}$, as shown in Figure 10. The plots show that the plans are non-dominated by each other.

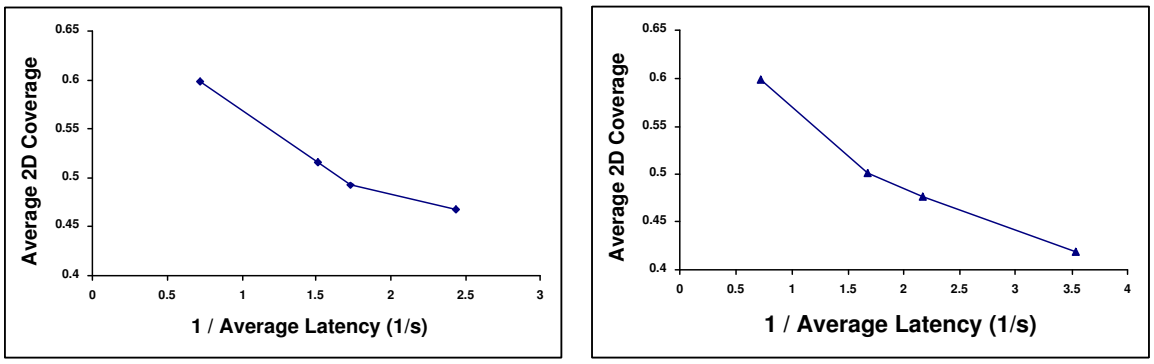

Fig. 10. On average, the plans generated with different discount/weight factor values are non-dominated by each other, using either the discount model or the weighted sum model

2) Evaluation Results on Bibfinder Test Bed: We conducted a similar experiment on the real *Bibfinder* test bed, where currently only coverage and latency objectives are supported. For the purpose of the evaluation, we define *relative coverage* of a query plan as following. Suppose

for a given query Q, its *coverage-only plan* $COP(Q)$ is the source selection plan that has the discount factor $\gamma = 1$ or the weight factor $\omega = 1$, depending on which utility function is used. $COP(Q)$ is a plan that only optimizes the coverage. For any given query plan $p(Q)$ of Q, with its own discount factor or weight factor, the *relative coverage* of $p(Q)$ is the size of the result set of $p(Q)$ divided by the size of $COP(Q)$. The relative coverage measures how much coverage can be achieved by executing plan $p(Q)$, compared to a coverage-only plan $COP(Q)$. The reason why we use relative coverage of a plan instead of its absolute coverage (i.e, the fraction of all possible answers that are covered by executing this plan) is that the latter requires the *Bibfinder* mediator to execute the naïve plans that simply call all the sources available. Our expectation on both of the joint-optimization models is that by varying the parameters of the utility functions the users' degree of preference on coverage and latency can be properly reflected in the query plan generated.

For evaluation, we randomly chose 200 real user queries from the query log, made query plans using each of the two utility functions, and executed the plan and recorded the time to retrieve the first K tuples (K=1, 2, 3, ...). We varied the discount factor γ and the weight factor ω to see their influence on query planning. Figure 11 shows the average time for retrieving the first K tuples using the 200 testing queries together with their average *relative coverage* (ARC) for varying discount factor values. Figure 12 shows the results of using the weighted sum model.

In both figures, when the discount factor γ or weight factor ω is 1, we have a plan that only uses coverage/overlap statistics without considering the latency. Thus, on average the speed of retrieving the first K answers is the lowest (because the faster sources are not rewarded for being faster) and the average *relative coverage* is highest. On the other hand, when the discount factor decreases in the discount model, the coverage estimates of the sources are increasingly discounted with respect to the latency. As a result, the plans generated tend to favor the faster sources, while the average *relative coverage* is getting lower. Similarly, in the weighted sum model, when the

weight factor ω is set to 0, the mediator effectively ignores the coverage statistics and only looks for the fastest sources. As shown in Figure 12, such plans retrieve the first K tuples the fastest on average.

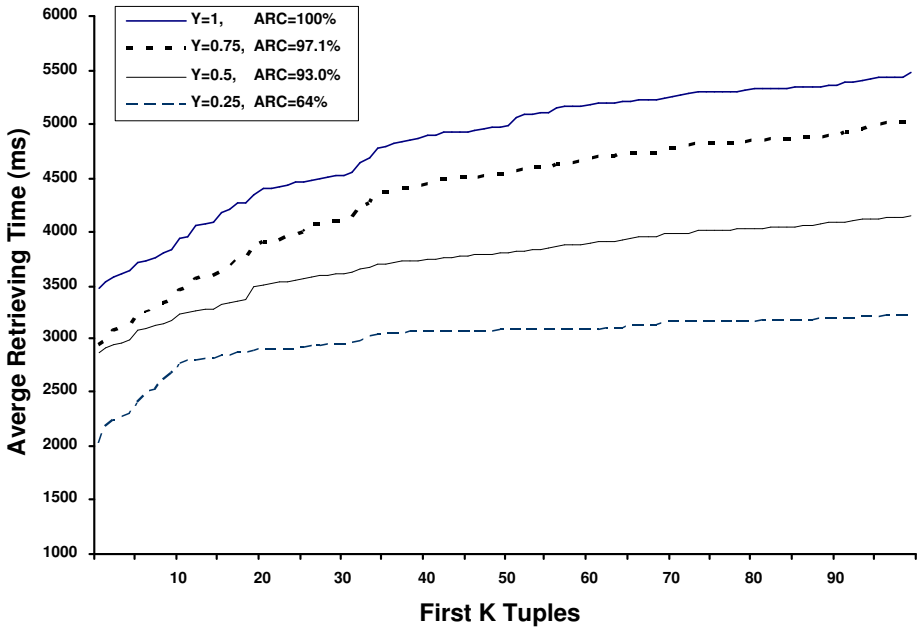

Fig. 11. Effect of the discount model and various discount factor values on *Bibfinder* test bed.

3) Summary of Evaluation: In summary, the experimental evaluation on both the test bed with synthetic sources as well as the one with online bibliographic sources shows that:

1) The query class based statistics learned from the sources help to make accurate estimates of the coverage/overlap, density and latency values for the new queries. These estimates give very useful guidance during source selection.
2) The joint optimization approach presented here is able to handle the coverage, density and latency objectives at the same time and make flexible trade-offs among them according to the users' preferences.

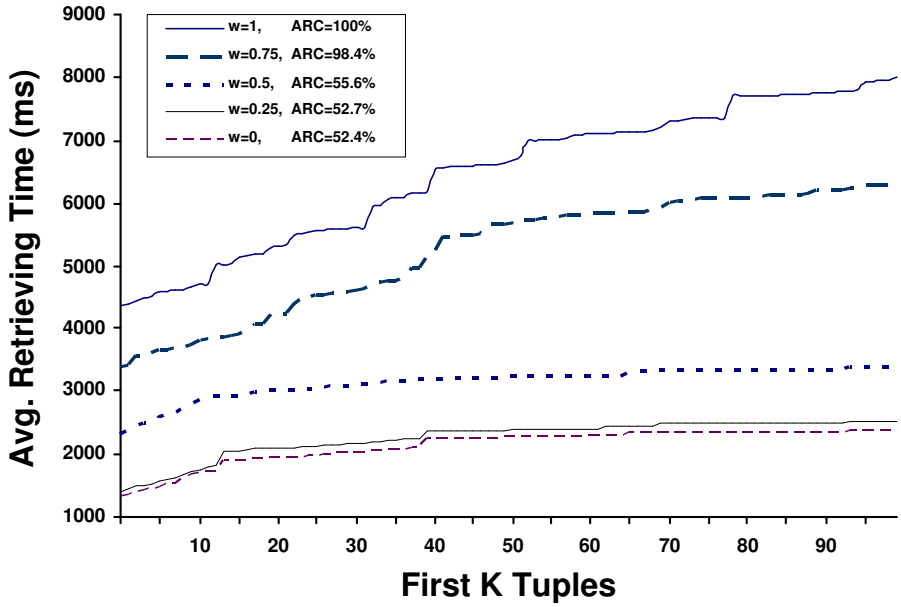

Fig. 12. Effect of the weighted sum model and various weight factor values on *Bibfinder* test bed.

VI. Conclusion and Future Work

In this paper we described an adaptive data aggregation framework that can effectively mine various types of source statistics to support various user query objectives. This framework has been shown to be effective in resolving the often conflicting nature of these objectives, supporting a spectrum of trade-offs between them and achieving multi-dimensional optimality during source selection. Finally, the joint optimization model we propose supports configurable parameters to adapt to users' own preferences in terms of the different objectives.

Acknowledgements:

We thank Zaiqing Nie and Ullas Nambiar for the discussions that lead to this work, and Garrett Wolf for comments on the current draft. This research was supported by ECR A601, the ASU Prop 301 grant to ETI3 initiative.

References

[1] Acm digital library. http://portal.acm.org/dl.cfm.

[2] Acm guide to computing literature. http://portal.acm.org/guide.cfm.

[3] Bibfinder: A computer science bibliography mediator. http://kilimanjaro.eas.asu.edu/.

[4] The collection of computer science bibliographies. http://liinwww.ira.uka.de/bibliography.

[5] Dblp bibliography. http://www.informatik.uni-trier.de/ ley/db.

[6] Ieee xplore. http://ieeexplore.ieee.org.

[7] Network bibliography. http://www.cs.columbia.edu/ hgs/netbib.

[8] Sciencedirect. http://www.sciencedirect.com.

[9] R. Agrawal and E.L. Wimmers. A Framework for Expressing and Combining Preferences. In *SIGMOD*, pages 297-306. 2000.

[10] W.-T. Balke and U. Güntzer. Multi-objective query processing for database systems. In *VLDB*, pages 936–947, 2004.

[11] J. Callan. *Distributed Information Retrieval. In W.B. Croft, editor, Advances in information retrieval*, chapter 5, pages 127–150. Kluwer Academic Publishers, 2000.

[12] A. Doan and A. Levy. Efficiently ordering plans for data integration. *IJCAI-99 Workshop on Intelligent Information Integration*, 1999.

[13] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive query plans for data integration. *J. Log. Program.*, 43(1):49–73, 2000.

[14] J. Fan. Adaptive query processing for data aggregation: Mining, using and maintaining source statistics. *Master's Thesis, Arizona State University*, 2006. *http://rakaposhi.eas.asu.edu/jcf-thesis.pdf*.

[15] K. Gajos and D. S. Weld. Preference elicitation for interface optimization. In *UIST*, pages 173–182, 2005.

[16] L. Gravano and H. Garcia-Molina. Generalizing gloss to vector-space databases and broker hierarchies. In *VLDB*, pages 78–89, 1995.

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

[18] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmman Publishers, 2000.

[19] P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *VLDB*, pages 394–405, 2002.

[20] Z. G. Ives, D. Florescu, M. Friedman, A. Y. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *SIGMOD Conference*, pages 299–310, 1999.

[21] E. Lambrecht, S. Kambhampati, and S. Gnanaprakasam. Optimizing recursive information-gathering plans. In *IJCAI*, pages 1204–1211, 1999.

[22] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, pages 251–262, 1996.

[23] W. Meng, K.-L. Liu, C. T. Yu, W. Wu, and N. Rishe. Estimating the usefulness of search engines. In *ICDE*, pages 146–153, 1999.

[24] F. Naumann. *Quality-driven query answering for integrated information systems*. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[25] F. Naumann, J. C. Freytag, and U. Leser. Completeness of integrated information sources. *Inf. Syst.*, 29(7):583–615, 2004.

[26] F. Naumann, U. Leser, and J. C. Freytag. Quality-driven integration of heterogeneous information systems. In *VLDB*, pages 447–458, 1999.

[27] Z. Nie and S. Kambhampati. Joint optimization of cost and coverage of query plans in data integration. In *CIKM*, pages 223–230, 2001.

[28] Z. Nie and S. Kambhampati. A frequency-based approach for mining coverage statistics in data integration. In *ICDE*, pages 387–398, 2004.

[29] Z. Nie, S. Kambhampati, and U. Nambiar. Effectively mining and using coverage and overlap statistics for data integration. *IEEE Transactions on Knowledge and Data Engineering*, May 2005.

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

[31] R. Pottinger and A. Y. Levy. A scalable algorithm for answering queries using views. In *VLDB*, pages 484–495, 2000.

[32] J. Shanmugasundaram, K. Tufte, D. J. DeWitt, D. Maier, and J. F. Naughton. Architecting a network query engine for producing partial results. In *WebDB (Selected Papers)*, pages 58–77, 2000.

[33] S. Viglas and J. F. Naughton. Rate-based query optimization for streaming information sources. In *SIGMOD Conference*, pages 37–48, 2002.

[34] W. Wang, W. Meng, and C. T. Yu. Concept hierarchical based text database categorization in a metasearch engine environment. In *WISE*, pages 283–290, 2000.

[35] J. Xu and J. P. Callan. Effective retrieval with distributed collections. In *SIGIR*, pages 112–120, 1998.

[36] V. Zadorozhny, A. Gal, L. Raschid, and Q. Ye. Arena: Adaptive distributed catalog infrastructure based on relevance networks. In *VLDB*, pages 1287–1290, 2005.