ADAPTIVE QUERY PROCESSING FOR DATA AGGREGATION: MINING, USING AND MAINTAINING SOURCE STATISTICS

by

Jianchun Fan

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

May 2006

ADAPTIVE QUERY PROCESSING FOR DATA AGGREGATION:

MINING, USING AND MAINTAINING SOURCE STATISTICS

by

Jianchun Fan

has been approved

April 2006

APPROVED:

______________________________________, Chair

____

____

Supervisory Committee

ACCEPTED:

____

Department Chair

____

Dean, Division of Graduate Studies

ABSTRACT

Most 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 is to select the appropriate subset of sources to optimize conflicting user preferences. We develop an adaptive data aggregation framework to effectively gather and maintain source statistics and use them to support multi-objective source selection.

We leverage the existing techniques in learning source coverage/overlap statistics, and develop novel approaches to collect query sensitive source density and latency statistics. These statistics can be used to optimize the coverage, density and latency objectives during source selection. We present a joint optimization model that supports a spectrum of trade-offs between these objectives. We also present efficient online learning techniques to incrementally maintain source statistics.

To my family.

ACKNOWLEDGMENTS

I would like to thank my advisor Professor Subbarao Kambhampati for his constant support, guidance and encouragement throughout my graduate studies in Arizona State University. I am very grateful for the time and effort he spent on working with me on my research. I deeply admire his devotion to his responsibilities and his enthusiasm for teaching and research. His vision and wisdom gave me great inspiration not only in my research but also in so many aspects of everyday life.

Special thanks to Professor Yi Chen for her wonderful collaboration and inspiring conversation. A big thanks goes to Professor Huan Liu for his encouragement generous time spent helping with my thesis.

I would also like to thank my friends in the AI lab: Thomas Hernandez, Ullas Nambiar, Zaiqing Nie, Hemal Khatri and Garrett Wolf for their many helpful critiques and discussions. Special thanks to my friends at Arizona State University, Yang Xiao, Feng Guo, Mengying Li and Guofeng Deng for their invaluable friendship.

I have been and will always be grateful to my parents for their unconditional love, support and encouragement.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

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 [17, 24, 25, 30, 34] are being developed to provide a uniform interface to a multitude of information sources, query the relevant sources automatically and restructure the information from different sources. Not only must a data integration system adapt to the variety of sources available, it must also take into account multiple and possibly conflicting user objectives. Such objectives can include overall coverage objectives, cost-related objectives (e.g. response time [19]), and data quality objectives (e.g. density [27], provenance of results [13]).

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. For this purpose, the mediator needs to access several types of source- and query- specific statistics, and use them to select sources in a way that is sensitive to the users' conflicting objectives.

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. While some types of statistics may well be voluntarily publicized by the individual data sources, many of these statistics need to be learned/gathered, and actively maintained by the mediator. This raises three challenges: 1) how to automatically gather various statistics from autonomous sources and use them to optimize individual goals, 2) how to combine the gathered statistics to support multi-objective query processing[1], and 3) how to maintain the gathered statistics to keep up with the continuously evolving system. We aim to provide a comprehensive framework for addressing these challenges.

To ground further discussion of the challenges as well as our solutions, let us consider the concrete example of a bibliographic mediator system such as *Bibfinder* [3]. According to its web page, *Bibfinder* aggregates several computer science bibliography sources including *ACM Digital Library* [1], *ACM Guide* [2], *Network Bibliography* [8], *IEEE xplorer* [7], *DBLP* [5], *CSB* [4] and *Sciencedirect* [9]. *Bibfinder* has a global schema $Paper(title, author, conference/journal, year)$. Each source only covers a subset of the global schema. *Bibfinder* presents a unified and more complete view for the users to query these bibliography sources. The queries submitted to *Bibfinder* are selection queries on the *Paper* relation. These queries are directed to the most relevant data sources, and the

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

tuples returned by the sources are presented to the users.

BibFinder offers interesting contrasts with respect to other bibliography search engines like *CiteSeer* [15] and *Google Scholar* [6]. First of all, because it uses online aggregation approach (rather than a data warehouse one), user queries are sent directly to the integrated Web sources and the results are integrated on the fly to answer a query. This obviates the need to store and maintain the paper information locally. Secondly, the sources integrated by *BibFinder* are autonomous and partially overlapping. By combining the sources, *BibFinder* can present a unified and more complete view to the user. These features require the *Bibfinder* to tackle the challenges of mining, using and maintaining various types of source statistics as discussed above. The following illustrates some of the specific challenges:

1. Not all sources have answers for a given query, and different sources may export the same answers. To prevent an unnecessary workload on the sources, the mediator should select only a relevant subset of the sources to be called for a given query. To make this source selection, *Bibfinder* needs to know the degree of relevance of each source in regards to the query (*coverage*), and the size of intersection of answer sets among multiple sources (*overlap*).

2. Users want to have quality answers that give valid information on all requested attributes. The mediator needs to know for a query, which sources tend to give better answers (with high *density*).

3. In web scenarios, users may want to have answers fast. Thus the mediator needs to know for a query, which source is able to respond with the shortest delay (*latency*).

4. Source selection may need to make reasonable trade-offs between multiple conflicting

objectives. For example the users might want a quick response to their query in addition to expecting it to return many high quality answers. Therefore the mediator needs to combine the multiple types of statistics together and try to jointly optimize the multiple objectives.

5. Online aggregation systems such as *Bibfinder* are constantly evolving. New bibliography entries are inserted into the individual sources on a daily basis; the focus of interest of the user population shifts rapidly with the emergence of new "hot spots" of computer science literature; the sources may upgrade their server and network topology might change. To keep up with them, the corresponding statistics (coverage/overlap, density and latency) learned by the mediator have to be updated accordingly.

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 approach 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. First, 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. Second, 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.

We have been developing an adaptive data aggregation framework to address the

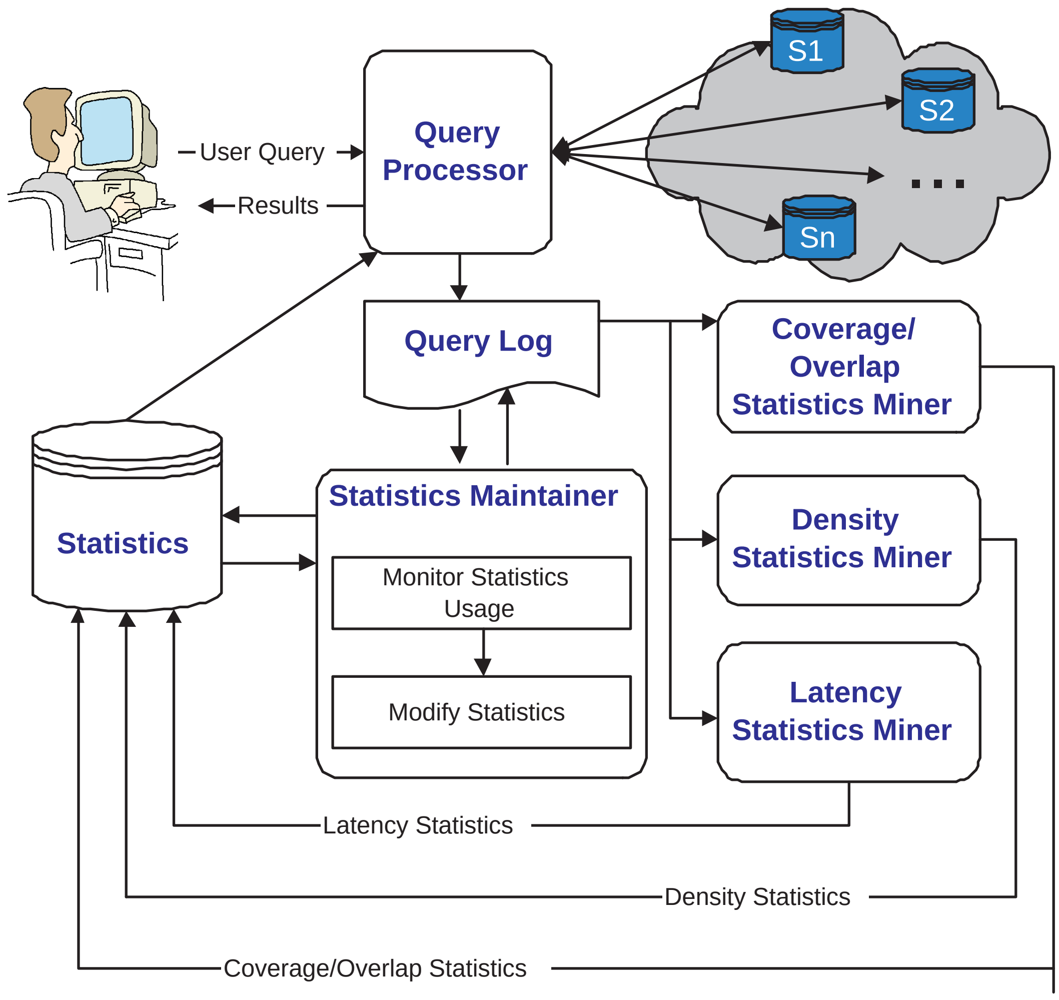


Figure 1. The Adaptive Data Aggregation Architecture

challenges of collecting multiple types of statistics, combining them for multi-objective query optimization and dynamically maintaining the various statistics. Figure 1 shows the architecture of this framework. Our work integrates the coverage/overlap statistics miner developed by Nie et al [30], which focuses on collecting coverage/overlap statistics from data sources and using them to optimize *coverage* for the given queries. It essentially partially addresses the first of the challenges listed above. This thesis makes four specific contributions:

Mining Source Density Statistics: We propose a source density statistics model to measure the database tuple level completeness of the sources. We develop techniques to efficiently collect such density statistics with respect to *projection attribute sets* of queries. We also extend the concept of coverage to 2D coverage to measure the completeness of query results on both the level of result sets and the level of individual tuples.

Mining Query Sensitive Latency Statistics: We propose a solution to gather source latency profiles in a query sensitive way by learning them with respect to the *binding patterns* of the queries.

Supporting Multi-Objective Query Processing: We develop two joint optimization models to help users handle conflicts among multiple objectives and optimize the query plans on *coverage*, *density* and execution *latency* at the same time.

Incremental Statistics Maintenance: Maintenance of statistics is not always a trivial task of relearning, considering the impact of learning cost on the scalability of the system. We introduce an incremental statistics maintenance approach for the purpose of reducing maintenance cost while keeping statistical accuracy.

For each of the above, we empirically evaluate the effectiveness of our solutions. Some of the evaluation is performed in the context of the *Bibfinder* online aggregation system. To facilitate the evaluation, we acquired the *Bibfinder* query log. We also set up a synthetic data aggregation system to evaluate some of our approaches that are not applicable in *Bibfinder*. Our experiments demonstrate that (1) density statistics allow the mediator to find the sources that provide the highest quality answers, and the joint optimization of density and coverage helps to maximize the 2D coverage of the answer set, (2) learning query sensitive latency statistics helps the mediator to identify fastest sources more accurately for specific queries, (3) the joint optimization models make flexible trade-offs among coverage, density and latency objectives, and (4) the incremental statistics maintenance approach significantly reduces the maintenance cost while keeping the statistics accurate.

The rest of this thesis is organized as follows. In the next chapter, we will first discuss some of the related work in source statistics learning in data integration literature. In Chapter 3 we will briefly introduce the existing work [29, 31, 30, 32] in learning source coverage/overlap statistics to optimize *coverage* of query plans. Chapter 4 focuses on learning source density statistics and combining density and coverage into 2D coverage. In Chapter 5 we will discuss a novel method for learning query sensitive source latency statistics to optimize the execution cost of query plans. Chapter 6 focuses on our joint optimization model considering coverage, density and latency in source selection. In Chapter 7 we will discuss the motivation and will present a solution to incremental statistics maintenance. We will present the conclusion and future work in Chapter 8.

CHAPTER 2

RELATED WORK

Several research efforts have focused on effective query processing in data integration scenarios [25, 23, 24, 34, 29, 36]. 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, rather than on selecting among multiple sources exporting similar information. In contrast our work focuses on data aggregation, which foregrounds the "vertical" aspects of data integration. Specifically, the sources export possibly overlapping fragments of a single relation, and the users pose selection queries to the mediator. The emphasis is thus on selecting sources appropriately. Data aggregation systems can thus be thought of as the structured-data analogues of the text meta-search engines [22, 14]. Although this problem seems simpler compared to the general data integration problem, effective solutions for it are much more important in practice. This is because to-date, a majority of the fielded data integration systems have been of the "data aggregation" variety.

Source Density Statistics: Naumann [26] introduces the concept of density that is used to measure how well the attributes stored at a source are filled with actual values. The source density is defined as average of the density of all attributes in a universal relation, and the query dependent density is defined as average of the density of the attributes in the set of query answers. We use a similar definition of density in our work, however we propose methods to *learn* query dependent density statistics. We control the cost and complexity of such learning by classifying queries according to their projection attribute set and only assessing the source density value with respect to such classes. Our 2D coverage model is similar to the completeness measure proposed by Naumann [26] 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 Statistics: As we already mentioned earlier, our work on learning statistics is related in spirit to the work by Nie et. al. [30]. Since we adopt their approach for learning coverage and overlap statistics, we will discuss it in a bit more detail in Chapter 3. There has also been previous work on learning response-time statistics in data integration literature [19, 36]. [37] 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. In addition, we use these query sensitive source latency statistics in conjunction with other types of source statistics in order to adapt to multiple user objectives.

Multi-Objective Optimization: Multi-objective optimization has been studied in database literature [20, 12] to use multiple data-oriented objective functions in ranking query results. In contrast, our work focuses on using source specific statistics in selecting relevant sources for given queries. In the data integration community, some existing query optimization approaches [25, 28, 16, 34, 12, 23, 35] use decoupled strategies by attempting to optimize multiple objectives in separate phases. In general, they cannot achieve overall optimality of multiple objectives due to the objectives' conflicting nature. In contrast, we present a novel joint optimization model to combine coverage/overlap,density and latency statistics.

Incremental Statistics Maintenance: There has been some work on learning selectivity or sampling data in the relational database literature [11, 18]. For example, [11] introduced a self-tuning approach to progressively refine the histogram of selectivity of range selection operators. Our work differs from theirs in at least the following aspects. First, our work aims to facilitate source selection in aggregating autonomous data sources, where query feedback is the only source of statistics learning. Second, instead of maintaining selectivity histograms on single attributes, our work directly learns and maintains statistics about query classes which can have multiple bound attributes. Last, our work cuts down the statistics learning and storage cost by keeping statistics only with respect to the frequent query classes and updates the statistics according to recent user queries. As a result, our work has an obvious user adaptive flavor. Our work is similar to [18] in this sense but differs from it by collecting different types of statistics (coverage and overlap) for different purposes (source selection in data aggregation scenario).

CHAPTER 3

MINING COVERAGE AND OVERLAP STATISTICS

We leverage the existing work of *StatMiner* [30] 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 introduce the basic concepts and algorithms here to the extent they are needed to understand our contributions. For more detailed information, we suggest the readers to refer to the discussion in [30].

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

3.1. Coverage and Overlap Statistics

In the *Bibfinder* scenario, 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 for 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'.

3.2. 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 should be able to compute 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 statics 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 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 the sources with respect to the new query.

To group the queries into query classes, *StatMiner* classifies them in terms of the attributes bound and the binding values in the 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 unions of the values of their descendants. Figure 2 shows two very simple AV hierarchies for the $conference$ and *year* attributes of the *paper* relation. 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 [21] to build a hierarchical classification of each attribute. The distance measure used in this clustering algorithm is the cosine distance of the vectors that represent query result distribution information (among the sources) of each individual attribute values.

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 of the classification attributes. For example Figure 2 shows an example class hierarchy for a simple mediator with AV

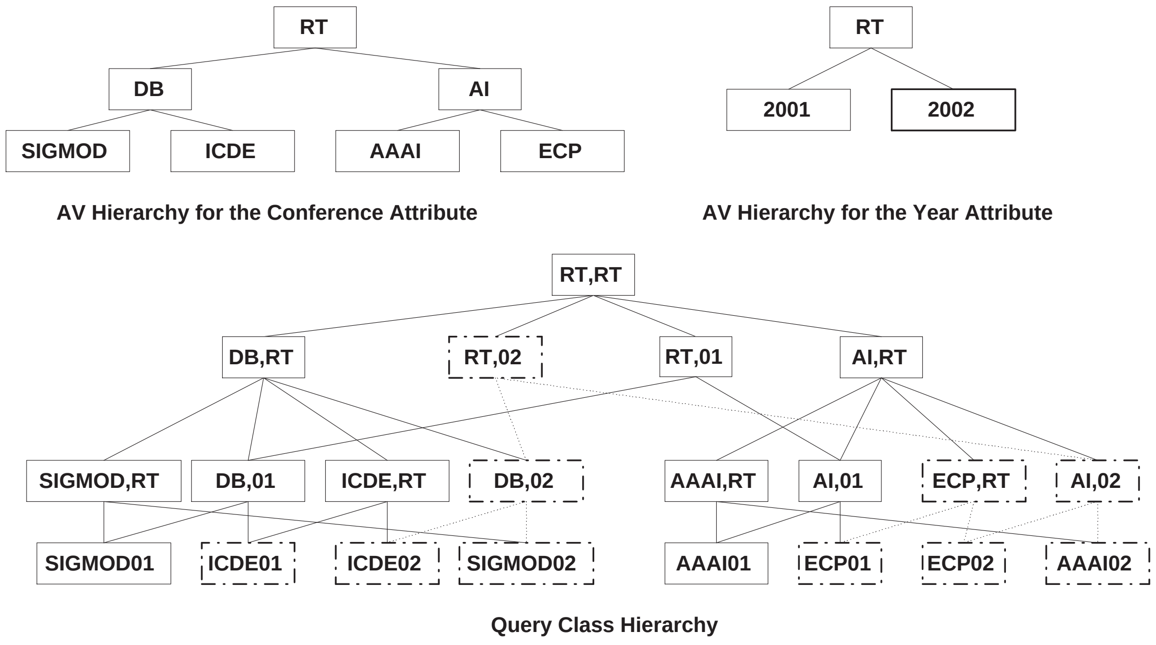


Figure 2. AV hierarchies and corresponding query class hierarchy.

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.

3.3. 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 reduces 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 that query belongs. The *access frequency* of a query class is defined as the number of queries that are mapped to it during a given period of time. A *frequent query class* is a query class that has access frequency above a given threshold. Nie et al [30] 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 of the classification attributes. The statistics of these frequent query classes are combined to estimate the coverage/overlap information for the new query. The mediator can then 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.

CHAPTER 4

LEARNING SOURCE DENSITY STATISTICS

The coverage statistics discussed in previous chapter essentially measure the "vertical completeness" of a source or source set with respect to a given query. For a given query Q, if we view the set of all qualified tuples from all sources as a database table, the coverage of a source or source set S for Q is the portion of the vertical edge that is covered by the tuples that come from S. Vertical completeness can be viewed as the completeness on the level of result set for a given query.

On the other hand, in data aggregation systems, incompleteness on the database record level is inevitable too. 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 usually not all attributes are supported by every individual source. For example, a global schema on 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 if users want quality answers, it 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". We first formally define density of a source or source set with respect to a given query. This definition is similar to the one proposed by Naumann [26]. We then propose a method to efficiently *learn* source density statistics of queries with respect to their projection attribute set. Such statistics can be used to make density oriented source selection. We further extend the concept of coverage to 2D coverage which is a joint measurement of the completeness on both vertical and horizontal directions, and be used to select sources that maximize the overall completeness of the result table of a given query. Compared to the completeness model proposed by Naumann [26], our 2D coverage model uses a configurable parameter to allow more flexible trade-offs between coverage and density to reflect users' own preferences.

4.1. Definition of Density

The density of a source depends on what attributes the users are interested in. For example, in a used car trading aggregation system where the global schema is *cars(make,model,year,price,milage,location,color)*, suppose data source S_1 has lots of missing values on attribute *color* and *model* but has values on every other attribute for every database record. If user A wants to know all the information of the cars, user B does not care about the color, and user C does not care about model and color, then source S_1 has 100% density for user C, lower density for user B, and even lower density for user A.

We thus define the density of a data source for a given user query with respect to the projection attribute set of the query result set returned by the source. Formally, let $\{A_1, A_2, \ldots, A_n\}$ be the set of projection attributes of a given query Q, S be one of the data sources in the aggregation system, and $RS_S(Q)$ the result set of Q returned from S. For any given result tuple $t \in RS_S(Q)$, 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 $RS_S(Q)$ may have missing values on different attribute(s). Obviously there are 2^n possible *density patterns* of tuples in $RS_S(Q)$, 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 $DP(RR_S(Q))$ be the set of 2^n density patterns for result set $RS_S(Q)$. If we can assess the probability of the appearance of each of the 2^n density patterns $P(dp|RS_S(Q))$, then we can define the overall density of source S for query Q as following:

$$density(S|Q) = \sum_{dp \in DP(RS_S(Q))} P(dp|RS_S(Q)) * density(dp)$$

This model essentially uses the weighted average of the 2^n density patterns to measure the overall density of the result set, where the weight of each density pattern is the likelihood of it being appeared in the results.

4.2. Learning Density Statistics

As stated earlier, the density of a data source for a given query is defined with respect to the result set. It may look as if density only depends on the projection attributes of the given query and has nothing to do with its selection predicates; however, this may not always be true. For example, in a used car trading database, it may be the case that most newer (e.g. $year \geq 2003$) cars are being sold by professional car dealers, and older cars are more likely being sold by lay users who tend to miss lots of attribute values when they enter the information into the database. In this case, the density of the attribute *year* may as well depend on the range of *year* values that are constrained by the selection predicates of the query.

Ideally for each possible query Q and data source S, the mediator can keep such density statistics regarding its projection attribute set and its selection predicates. When the query is submitted to the mediator again, the mediator can simply find out which subset of sources gave answers with the highest density for that query and dispatch the query on to those sources.

However this is obviously an unrealistic approach, considering the huge amount of possible queries that can be submitted to the mediator schema. In fact, if the mediator supports arbitrary selection predicates as standard SQL, the number of possible queries on a given database relation is infinite. One may argue that in the data aggregation system where the user query interface is very limited, and the selection constraints are usually simple conjunctions or disjunctions of the form *"attribute op value"* (where *op* is the value comparison operator such as =, >, <=, etc). However, the number of possible queries is still exponential to the number of attributes and linear to the domain size of each attribute.

Similar to the ideas of previous work on learning coverage/overlap statistics, an obvious alternation is to group queries into query classes where queries in same query class share similar density characteristics. This way the mediator will only learn density statistics with respect to the query classes.

4.2.1. Defining Query Class for Density Statistics. To avoid the complicated task of learning density statistics that depend on the selection predicates, as shown in the example above, we first make the following assumption:

Assumption 1. *If a database tuple t represents a real world entity E, then whether or not t has missing value on attribute A is independent to E's actual value of A.*

This is intuitive in many applications. For example, in a used car database, tuple t_1 describes a car that is white and tuple t_2 describes a car that is red. In this case, t_1's chance of having a missing *color* value shouldn't be significantly higher or lower than that of t_2 when these records are added to the database.

With this assumption, we can avoid collecting density statistics for every possible query. Specifically, since the density of source S on a single attribute is independent to the values of that attribute, the density of source S on a query is independent of its selection predicates, which are used to put constraints on the values. Therefore we can classify queries with respect to their set 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. Thus we define the "query class" of density statistics as the set of projection attributes of the queries. Therefore, we will only learn density statistics with respect to these query classes.

4.2.2. Learning Density Statistics. For a database relation $R(A_1, A_2, \ldots, A_n)$, the set of all possible projection attribute set 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 $\sum_{1 \leq m \leq n} C_n^m 2^m$, which is still a significantly large number. To efficiently learn density statistics, we further make the following assumption:

Assumption 2. *Let A_1 an A_2 be two arbitrary database attributes, then the probability of tuple t having a missing value on A_1 is independent of whether or not t has a missing value on attribute A_2.*

This assumption is also intuitive in most scenarios. For example, in a used car database, whether a tuple has a missing "model" value is not likely to be correlated to whether or not its "color" value is missing.

This independence assumption greatly reduces the number of probability values we need to assess. Specifically, for relation $R(A_1, A_2, \ldots, A_n)$, the mediator only needs to assess n probability values: $\{P(A_1 = null), P(A_2 = null), \ldots, p(A_n = null)\}$. Let qc be any given query class (projection attribute set), which is simply a subset of $\{A_1, A_2, \ldots, A_n\}$, namely $\{A_1, A_2, \ldots, A_m\}$ $(1 \leq m \leq n)$. The joint probability for each of the 2^m density patterns of qc can be estimated by the product of each attribute in qc being (or not being) missing. For example, for density pattern $dp = (A_1, \neg A_2, ..., A_m)$,

$$\begin{aligned} P(dp) &= P(A_1 \neq null \ \wedge \ A_2 = null \ldots \ \wedge \ A_m \neq null) \\ &= P(A_1 \neq null) \ * \ P(A_2 = null) \ * \ \ldots \ * \ P(A_m \neq null) \\ &= (1 - P(A_1 = null)) \ * \ P(A_2 = null) \ * \ \ldots \ * \ (1 - P(A_m = null)) \end{aligned}$$

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, it is able to figure out these probability values easily. To collect the sample data for a given source, the mediator can simply issue probing queries to the source and use the union of the results returned as the sample database. This however requires the mediator to send a relatively large group of queries to the source. A less intrusive way of collecting a sample database is 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 for a specific 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.

For any given query Q that belongs to query class $\{A_1, A_2, \ldots, A_m\}$, the density of a source or source set for Q can be estimated as

$$density(S|Q) = \sum_{dp \in DP\{A_1, A_2, \ldots, A_m\}} P(dp) * density(dp)$$

where $DP\{A_1, A_2, \ldots, A_m\}$ is the set of 2^m density patterns.

4.3. Using Density Statistics

With the density statistics, the mediator can direct a user query to the sources with the highest estimated density for that query. This density sensitive source selection plan is aimed to return the most quality tuples which have the fewest number of missing values. We call this type of source selection plan a *density-oriented* plan since it optimizes on the density of the answers. Similarly, the previous work using coverage/overlap statistics in source selection generates *coverage-oriented* plans.

High coverage and high density are both desirable in data aggregation systems. However, sometimes it is impossible to achieve both. Some sources may return lots of answers but each of the answers has many missing values (Figure 3 (a)), and others may return only a few quality answers with most attribute values fulfilled (Figure 3 (b)). Therefore, the system needs to be able to accommodate the trade-off and jointly consider both coverage and density in source selection. 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. In this sense, 2D coverage is also 2D density. Some sources may have high 2D coverage without having the highest value on either of the individual dimensions (Figure 3 (c)).

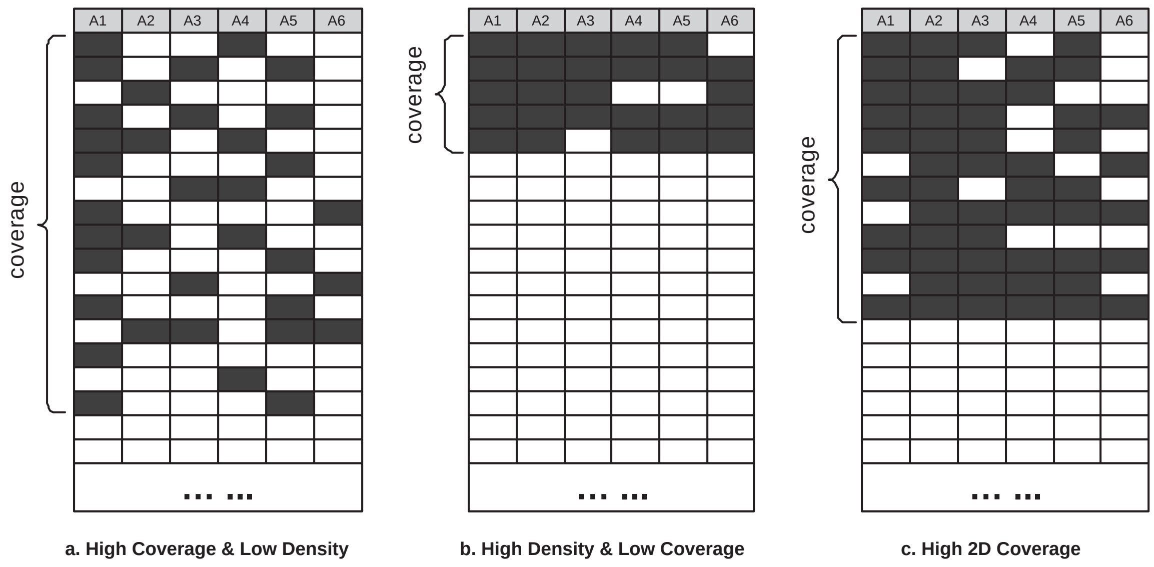

A1
A2
A3
A4
A5
A6
coverage
... ...
a. High Coverage & Low Density
b. High Density & Low Coverage
c. High 2D Coverage

Figure 3. 2D coverage combines measurement of coverage and density

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:

$$\begin{aligned} 2DCoverage(S|Q) &= coverage(S|Q) * density(S|Q) \\ &= coverage(S|Q) * \sum_{dp \in DP\{A_1, A_2, \ldots, A_m\}} P(dp) * density(dp) \end{aligned}$$

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 on 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:

$$2DCoverage(S|Q) = coverage(S|Q)^{\kappa} * density(S|Q)^{1-\kappa} \ (0 \leq \kappa \leq 1)$$

We call κ the *scale factor* in this formula since it is used to scale the measure of coverage and density in computing the 2D coverage. This scale factor allows the users to set their own preferences on coverage and density.

With the 2D coverage estimation of the sources for a user query Q, the mediator is able to make source selection to maximize the 2D coverage of the answer set with a limited number of source calls. Recall that in the previous work on source selection where coverage was maximized, the residual coverage with respect to the sources that are already selected was used to take into consideration the overlap between the sources. Similarly, we can introduce the concept of residual 2D coverage by simply replacing the $coverage(S|Q)$ component in the above formula with $residualCoverage(S|Q)$. The source selection follows the same procedure: first the source with the highest 2D coverage is selected, then the one with next highest 2D coverage but lowest overlap with the first selected source, and so on. We call such a source selection plan a *2 dimensional* plan.

To execute a 2 dimensional source selection plan, the mediator sends the query to all the selected sources. The answers returned are merged based on their primary key (we assume the primary key is never missing and will always be returned). Such merging has two advantages: (1) multiple sources might return the exact same tuples due to the overlap between the sources. As a result of the merging process, all redundant information is filtered out. (2) Different tuples that represent the same entity (having same primary key) may be returned by multiple sources, but the attribute values are complementary. The merging thus gives a result tuple that is more complete (less missing attribute values) than the ones given by any individual sources.

In the next section we will show that this joint optimization model helps to maximize the 2D coverage for a given query with limited source calls. It tends to provide the most number of attribute value cells when compared to coverage-oriented and density-oriented plans.

4.4. Empirical Evaluation

We evaluated our work on density statistics mining and joint optimization of coverage/density using a synthetic test bed, since the realistic *Bibfinder* test bed does not show enough interesting variation in source density characteristics. The evaluation is focused on using the statistics learned to generate source selection plans, and verifying that the learned source statistics provide useful guidance in maximizing the density or 2D coverage.

4.4.1. Experiment Setting. The synthetical test bed we used in the evaluation is set up in the following way. We used the Yahoo! autos [10] database as the base data. This database has around 100,000 used car records. The schema of the database is *cars(make,model,year,price,milage,location,color)*. We inserted a primary key field for the convenience of comparing coverage and overlap. We set up 30 artificial data sources for the purpose of evaluating our source selection techniques. Each of the 30 source contained a random portion of the car database, ranging from 10% to 50% in size. For each of the

30 database we made it horizontally incomplete. To do so, 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 5% random portion of each of the 30 databases as their sample to learn the density statistics. As for coverage statistics, instead of using the existing approach discussed in Chapter 3, we directly made the estimation from the sample databases. The reason is that we did not have a real query log to apply the techniques used in Chapter 3, and the main purpose here is to evaluate the density statistics and joint optimization. In fact, the coverage learning method and statistics model can be smoothly plugged into the 2D coverage model.

We used a group of realistic queries on the car database in our evaluation. We evaluate our techniques by generating top K source selection plans and comparing the measurements of the results in terms of coverage, density and 2D coverage. Table 1 lists some of the representative queries.

Table 1. Some representative queries on car databases for the evaluation.

Query	K
select make, model, color, year, mileage where mileage<25000 and color="white"	3
select model, color, mileage where make="ford"	3
select make, color, price, mileage where year>2002	5
select make, model, year, price where location="chicago"	3
select color, year, price, mileage where model="civic" and price<3000	3
select model, price, mileage where location="boston" and color≠"black"	3
select make, price, color where location="los angeles"	4
select make, model, price where color="red"	3
select model, price, location where year>2001 and color≠"green"	1

4.4.2. Evaluation Metrics. We use the measurements of the result set returned from the selected source to judge the effectiveness of the density statistics and source selection. A source selection plan is simply a subset of all sources that are selected to be called. For a given query Q and a source selection plan p, we define the following metrics to evaluate the plan.

Result Set Coverage: The result set coverage of the plan p is the number of distinct tuples returned from the sources in p, divided by the total number of distinct tuples returned from all sources.

Result Set Density: The density of a tuple returned for Q is the number of concrete values (not missing) divided by the total number of projection attributes of Q. The result set density of plan p is the average of all result tuples returned from the sources in p, which is actually the sum of the density of the tuples divided by the total number of tuples.

Result Set 2D Coverage: The result set 2D Coverage is the number of database values of all distinct tuples, divided by the product of the total number of distinct tuples from all sources and the number of projection attributes of Q.

4.4.3. Experimental Results.

4.4.3.1. *Optimizing Density and 2D Coverage.* We generated coverage-oriented plans, density oriented plans and 2 dimensional plans for 10 test queries. In our experiment, we used scale factor $\kappa = 0.5$ in combining coverage and density, assuming that both dimensions are equally important. We evaluated the query results using each of the above metrics. Figure 4 shows the result set density values for each of the 3 plans given the test queries. The results demonstrate the effectiveness of our density statistics learning approach. When using the learned density statistics, the density oriented plans almost always select the sources with highest density for the given queries. It also shows that by jointly optimizing density and coverage, the 2 dimensional plans usually result in higher density answers than plans where only the overage of the sources is considered.

Figure 5 shows the result set coverage values for each of the 3 plans given the test queries. Not surprisingly, in most cases the coverage-oriented plans select the sources with highest coverage for the given queries. It also shows that 2 dimensional plans result in higher coverage than plans where only density of the sources is considered.

Figure 6 shows the result set 2D coverage values for each of the 3 plans given the test queries. It shows that in most cases, 2 dimensional plans maximize the 2D coverage and select the sources that give the highest 2D coverage.

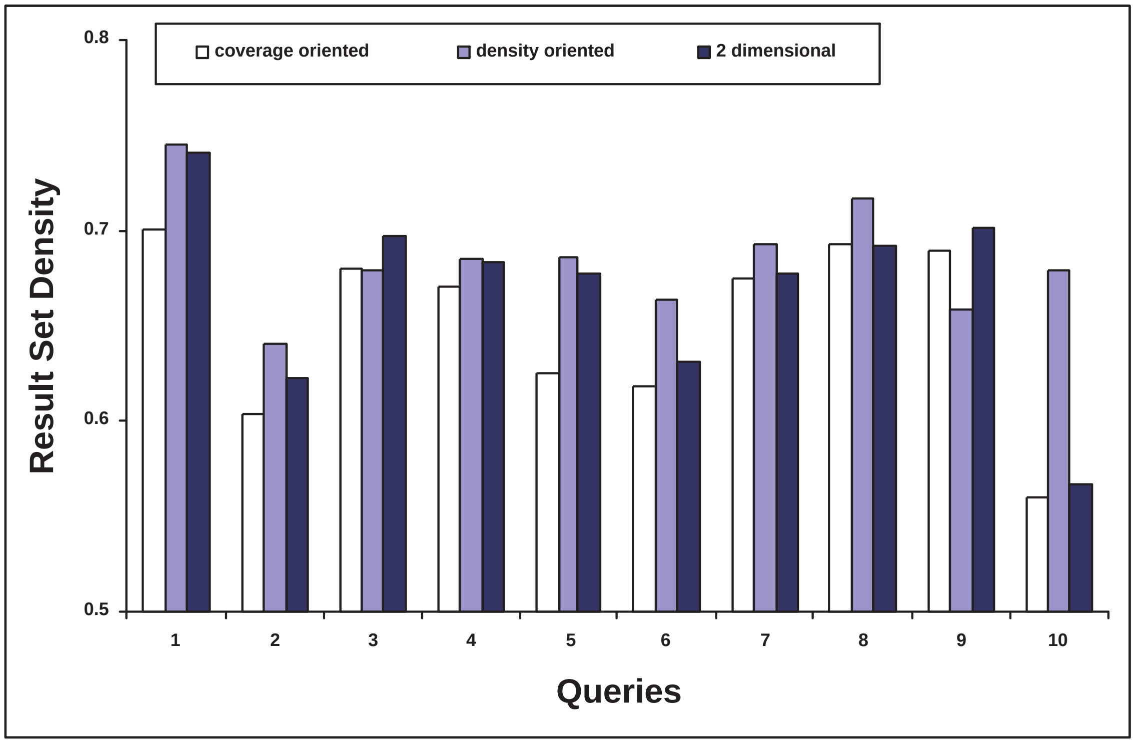


Figure 4. Result set density values each of the 3 plans.

We also summarize the average result set coverage, result set density and result set

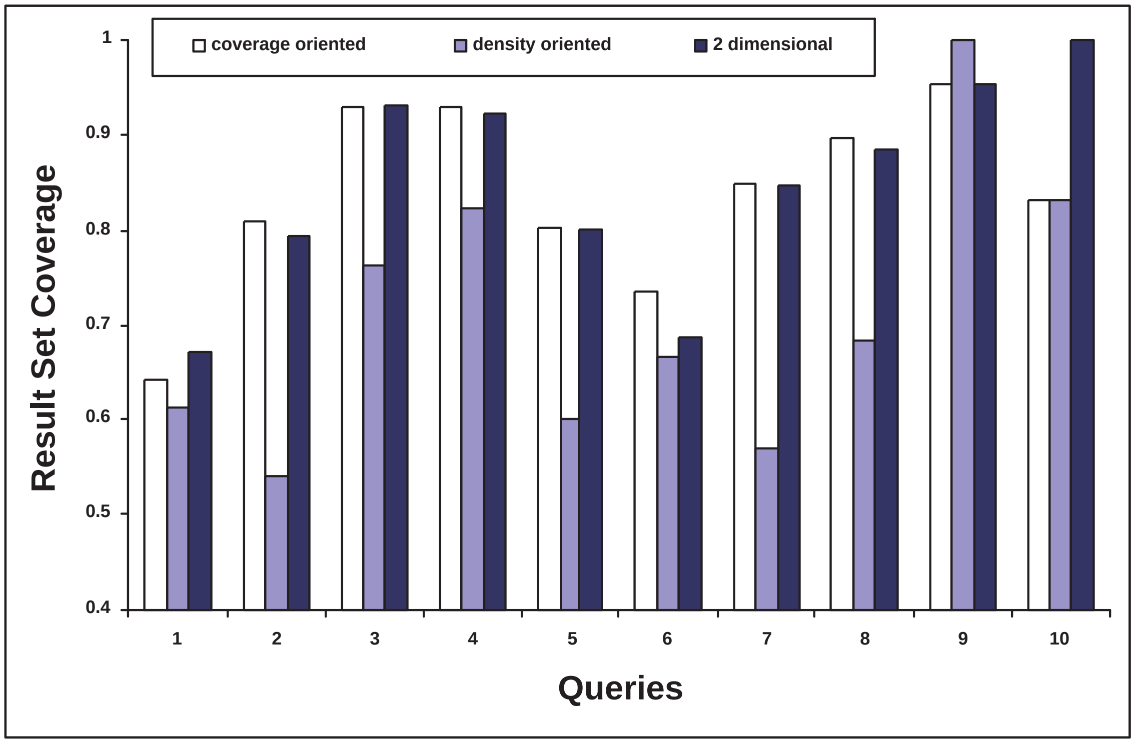


Figure 5. Result set coverage values for each of the 3 plans.

2D coverage of all 3 types of plans for 20 testing queries in Table 2.

Table 2. Result set metrics for 3 types of plans.

	result set coverage	result set density	result set 2D coverage
coverage oriented	0.807	0.635	0.661
density oriented	0.668	0.673	0.578
2 dimensional	0.802	0.655	0.684

These experiments show that our source density statistics model captures the density feature of the data sources, and the density statistics we learn gives an accurate estimate of the source density values for new queries. Such estimates effectively guide the mediator allowing it to optimize the source selection in terms of the horizontal completeness. These experiments also show that the joint optimization model on coverage and

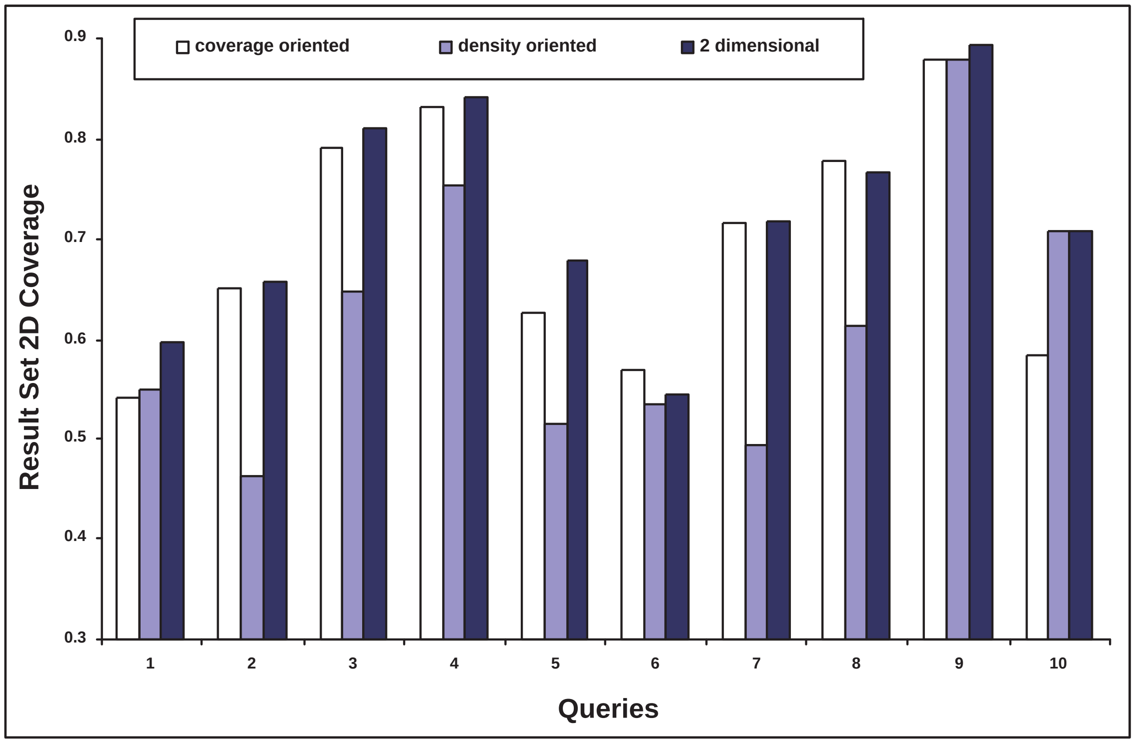


Figure 6. Result set 2D coverage values for each of the 3 plans.

density successfully addresses the problem of optimizing the 2D coverage of the result set.

4.4.3.2. *Flexible Trade-off between Coverage and Density.* Another important aspect we evaluated was whether or not the 2D coverage measurement, with changing scale factors, can make flexible trade-offs between coverage and density to adapt to the preferences of different users.

We therefore made the following experiment. We chose 20 realistic queries on the car database, made source selections with 2 dimensional plans, and computed the average result set coverage and result set density for each of the 20 queries. We repeated this experiment multiple times with a different scale factor κ value used in the 2D coverage formula. Table 3 shows the affect on the average result set coverage and result set density values when scale factor κ varies from 0.0 to 0.5.

Table 3. The effect of changing scale factor values.

scale factor value	0.0	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50
result set coverage	0.668	0.740	0.770	0.779	0.789	0.806	0.809	0.810	0.817	0.813	0.813
result set density	0.673	0.670	0.666	0.664	0.663	0.660	0.656	0.654	0.649	0.648	0.648

We plotted the values of Table 3 in Figure 7. When the scale factor value increases, the sources with high coverage are more favored during source selection and the result set coverage increases correspondingly. Similarly, with the decrease of the scale factor value, the sources with high density are more favored in source selection and thus the result set density increases too.

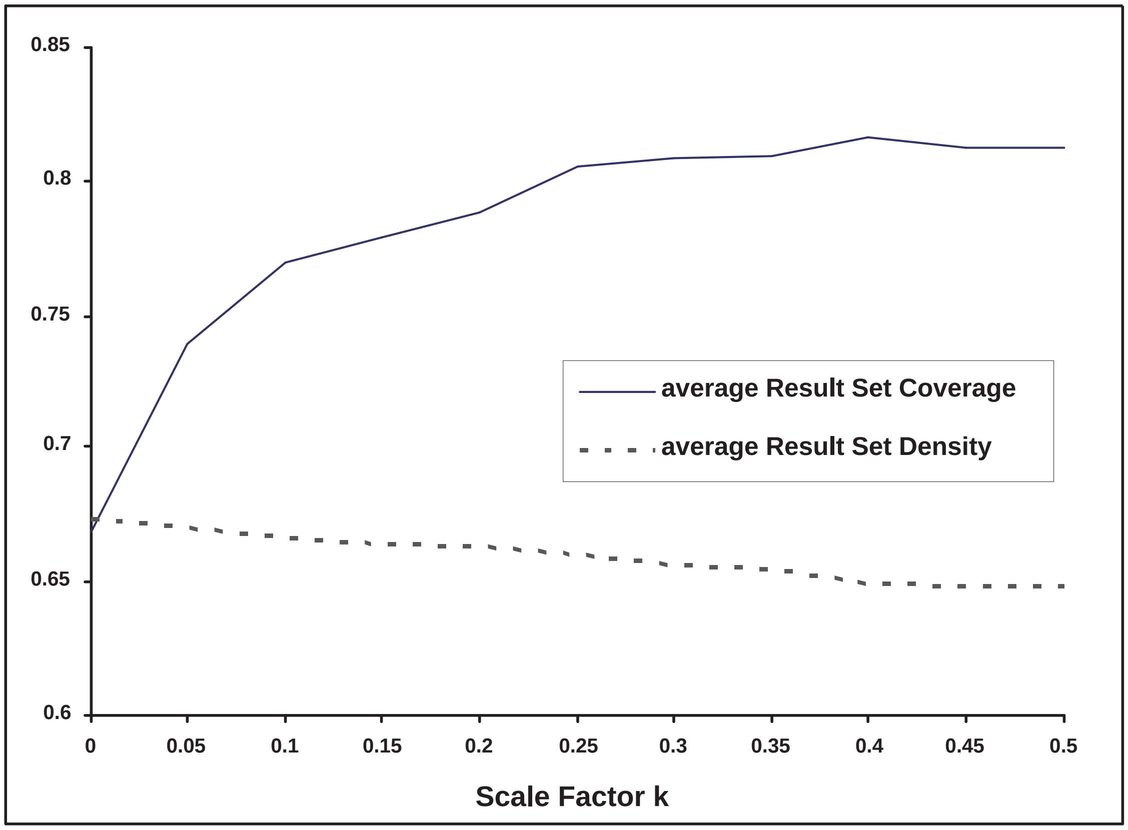


Figure 7. The effect of varying scale factor value.

This experiment clearly shows that the joint optimization model of coverage and density, with configurable scale factor on the 2 dimensions, makes flexible trade-offs according to different users' coverage and density preferences.

CHAPTER 5

MINING SOURCE LATENCY STATISTICS

In data aggregation scenarios, the speed of retrieving the answer tuples is also a very important measure of the goodness of a source selection plan. 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. To optimize the execution cost, the mediator needs the source latency[1] statistics.

5.1. Query Specific Nature of Latency

Because of the autonomous and dynamic nature of the data sources and the network topology, the latency statistics of individual sources have to be learned by the mediator. The first issue is how to define the latency. Some work has been done in learning source latency profiles. For example, [19] 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

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

in many cases. For example, when we monitored queries on *Bibfinder*, we found that the same data sources sometimes give very different latency values on 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.

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 that data source.

We already have the “query class” model used by *StatMiner*, so a natural idea would be to use such query classes to classify the queries and learn source specific latency values for each query class. However, this query class model is based on AV hierarchies and thus such a classification depends on the binding values of the queries. Based on our observations, query execution latency is usually less sensitive to the binding values[2], but rather more dependent on the *binding pattern*[3] of the queries. Specifically, we found that for a given data source, the latency values are usually quite different when the binding patterns of the queries are different.

In the *Bibfinder* test bed, we found that for the categorical attributes *title*, *conference* and *author*, whether or not these attributes are bound highly differentiates the latency

[2]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.

[3]Binding pattern of a selection query describes which attributes are bound to concrete values in the query and how they are bound (e.g. equality binding, range binding, 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”$)$.

for most data sources. For the numeric attribute *year*, we found that for a query with the *year* attribute bound to a range of numbers (e.g. 1994 to 2003), the latency is usually very different from that of a query with the *year* attribute bound to a single number, and both of them differ highly from the latency of a query that has the *year* attribute free. Therefore, in the *Bibfinder* scenario, there are $(2 \times 2 \times 2 \times 3 - 1)$, or 23 different binding patterns.

We also need to decide how to quantify the latency value. In the *Bibfinder* scenario, the latency is intended to measure how fast a source can answer a query. 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 shown in Figure 8, 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 experiment shows that at least in the *Bibfinder* scenario, 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. This can be observed in Figure 9. These observations indicate that the "query class" model defined in Chapter 3 may be an overkill for classifying the queries in learning latency statistics. Instead, we decided to learn source latency statistics based on another type of "query classes" - the binding patterns of queries. For every [source, binding pattern] combination we learn the latency statistics and use them to predict the latency values of the future queries that are bound in the same pattern.

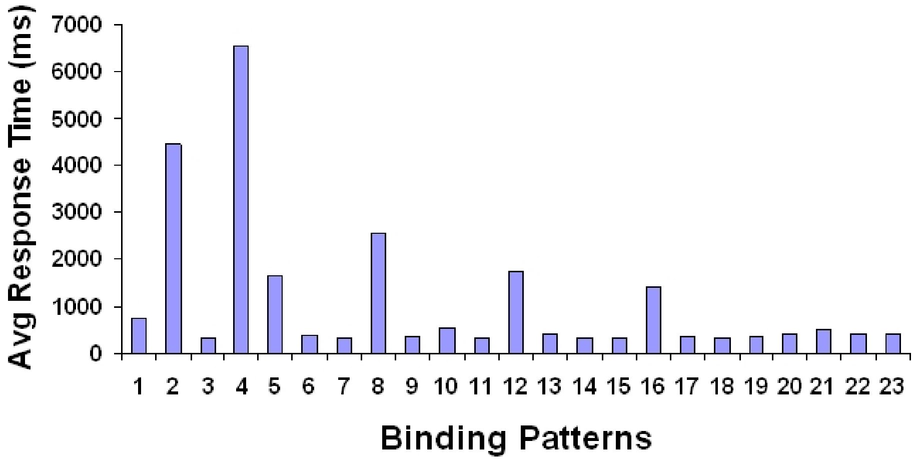


Figure 8. The average latency of the DBLP on different binding patterns.

5.2. Learning Latency Statistics

As stated above, we need to learn the latency statistics for each [source, binding pattern] combination. We randomly selected up to 20 real user queries for each of the 23 binding patterns from the query log of *Bibfinder* and sent each of them to each of the 5

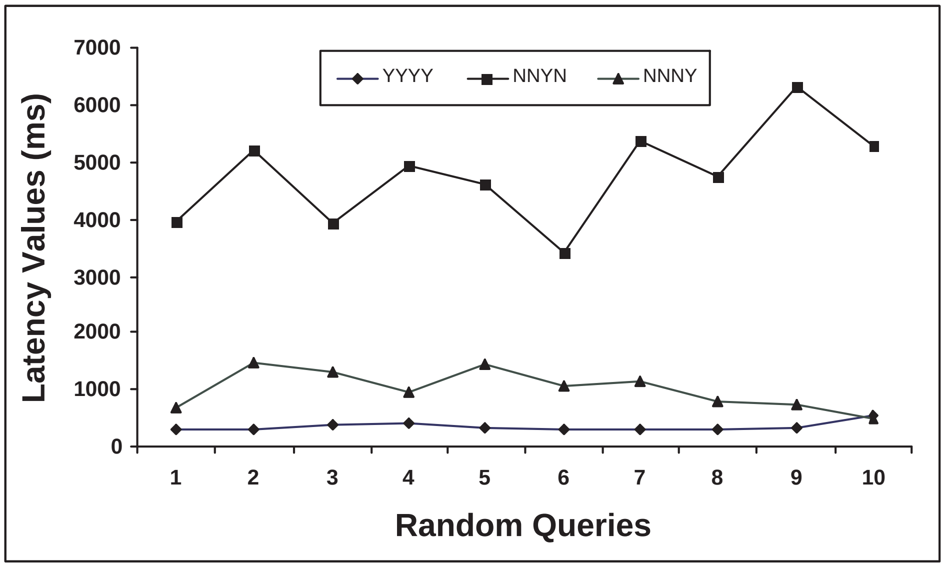


Figure 9. The queries with same binding pattern have similar latency values.

sources. We used the average response time as the latency value for each [source, binding pattern] combination. We conducted the probing several times at different times of the day to minimize the influence of the difference in the source work loads during different time frames throughout the day[4].

The learning results show that although the latency value for a given data source may vary significantly across different binding patterns, the queries with same binding pattern usually have very similar latency values for a given data source. This validates our model where learned latency values are used to predict the expected latency values for the future queries. In fact, our test on a random query set shows that the prediction has estimation error ($\frac{|estimatedlatency - reallatency|}{reallatency}$) which is less than 20% in most cases.

Not all sources show significant difference in the latency values for different binding patterns. For example the source *CSB* is very stable and its response time for any query

[4]In fact in contrast to the results in [19], in our setting no obvious differences in latency values at different times of the day are observed. The nature of the source servers and network topology largely decide the average response time of the sources.

is almost a constant value. However since this is not true for other sources, this does not compromise our method of differentiating queries based on binding patterns.

As stated earlier, the queries we used for probing the source latency values were from the actual query log of *Bibfinder*. Some query patterns, such as queries like "give me all the papers published in 1995" (q(__, __, __, "=1995")), are so rarely asked that we can only find very limited number of queries with such binding patterns from the query log to probe the sources. This implies that the latency values we learned for such binding patterns might be highly inaccurate. However, this is consistent with our basic philosophy of "keeping more accurate statistics for more frequently asked queries", and will rarely harm the processing of the real user queries.

5.3. Effectiveness of Query Sensitive Latency Statistics

With the learned latency statistics we are able to optimize the query plan in terms of retrieval cost, in other words, to find the faster data sources to call for a given query.

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 *query insensitive*, more coarse source latency values which do not differentiate the binding patterns. When a new user query is submitted to the mediator, it finds the specific binding pattern of this query *bp*, retrieves all the [source, *bp*] values and then chooses the fastest sources to call.

We compare this approach with the existing *query insensitive* source latency values (learned by probing the sources with a set of random queries from the query log, not differentiating their binding patterns). As shown in Figure 10, our approach estimates the latency value much better and more accurately identifies the fastest data sources.

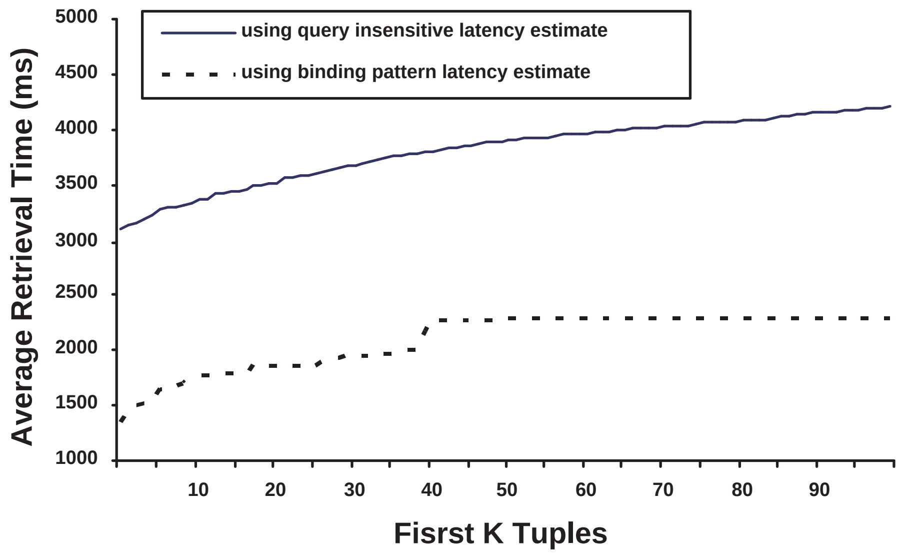


Figure 10. By using query sensitive latency values are much more acurate.

CHAPTER 6

MULTI-OBJECTIVE QUERY OPTIMIZATION

Using coverage/overlap statistics helps the mediator with source selection to maximize the coverage with limited resources. Similarly, the density statistics help to identify the sources that give the highest quality answers. On the other side, the binding pattern based latency statistics we learned help to optimize the query plan in terms of expected response time.

However, in data aggregation systems, the users usually prefer query plans that are optimal in terms of both coverage, density and latency. Unfortunately, these objectives often conflict. A source selection plan optimal with respect to one objective cannot in general be post-processed to account for the other objectives. For example, sources with high coverage are not necessarily the ones that respond quickly or provide quality answers. Therefore, the mediator will need to generate plans which optimize coverage, density and latency so that the users can get most quality answers as soon as possible. Moreover, different users usually have different preferences on the different objectives. For example, some 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 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 coverage/overlap, density and latency statistics gathered.

In data integration scenarios, some existing query optimization approaches [25, 28, 16, 34, 12, 23, 35] use decoupled strategies by attempting to optimize multiple objectives in separate phases during the query processing. However, due to the conflicting nature of these objectives, such decoupled strategies usually can not achieve overall optimality and are very hard to adapt to different users' unique preferences.

In this chapter we first present two models that can jointly optimize coverage and latency in source selection. We will then show that 2D coverage, as discussed in Chapter 4, can be easily plugged into either of these 2 models and therefore supports the joint optimization of coverage, density and latency.

6.1. Combining Coverage and Latency Optimization

6.1.1. Joint Optimization Models. The goal of multi-objective query optimization is to find the "overall best" query plans in terms of multiple user objectives. In *Bibfinder* and most web data aggregation scenarios, *coverage* and *latency* are two of the most important objectives. We need to determine what "overall best" means in these scenarios. For a given query Q, each query plan $p(Q)$ corresponds to a vector in a two dimensional space, with *coverage* and $\frac{1}{Latency}$ (since lower latency is preferred) as its two dimensions. The multi-objective query optimization then becomes the problem of picking the best vectors among these vectors. This type of problem is one of the central problems addressed in operations research [12]. The first principle solution to such a problem is to present all non-dominated vectors[1] (each of the vectors is called a "pareto optimal" solu-

[1]a vector $v1$ is dominated by vector $v2$ if $v1$ is worse than $v2$ in all dimensions

tion) to the user (so they can select one). This is often infeasible if the pareto set contains many vectors [33].

A popular alternative is to convert the multi-objective problem to a single objective one by aggregating the different dimensions of each vector into a scalar by using a utility function. In this approach, only the vector with the best utility value is presented to the user. We adopt this second method and propose two different models to simultaneously use coverage/overlap statistics and latency statistics. The first model is a *discount model*, which emphasizes the expected 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/overalap statistics and the latency statistics:

$$Util(S_i) = ResidualCoverage(S_i|Q) \times \gamma^{Latency(S_i|Q)}$$

In the formula above, for a given query Q, the coverage estimate 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 Chapter 3. 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 coverage, and the source selection plan generated only optimizes on coverage. By reducing γ, the sources that have shorter latency are favored and thus the plan is able to jointly optimize both coverage and latency. By adjusting γ, individual users can express their preferences on coverage and latency. We will show in next section that this combined utility function does make reasonable trade-offs between multiple objectives.

One minor problem with this model is that it is slightly asymmetric. We can optimize the coverage only 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 [29]:

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

This *weighted sum* model simply combines the coverage measure and latency measure. We take logarithm of the coverage value so that it would be comparable to the latency value and 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 "coverage only" to "latency only" or somewhere in-between. We will show next that both models can easily accommodate the trade-off between the coverage and latency objectives and optimize the query plan in both directions.

When a new query is submitted to the mediator, the corresponding latency statistics of each source with respect to its binding pattern are retrieved. At the same time, the query is mapped to one or more frequent query classes to get the coverage/overlap estimation of it. These two types of statistics are then fed into one of the joint optimization models to generate a source selection plan that tries to achieve optimality in both coverage and latency.

6.1.2. Experimental Evaluation. To evaluate the multi-objective query optimization approach, we experimented with *Bibfinder* to see if it can make reasonable trade-offs between latency and coverage and generate query plans that can reflect the users' preference for different objectives. We evaluated the utility function to see if the plans generated using combined statistics can reward both high coverage and low latency appropriately. 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 $COP(Q)$. 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, 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 in query planning. Figure 11 shows the average time for retrieving the first K tuples of the 200 testing queries together with their average *relative coverage* (ARC), with different discount factors. Figure 12 shows the results of using the weighted sum model.

In both figures, when 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 more and more discounted with the latency. As a result, the plans generated tend to favor the faster sources, but 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 fastest on average.

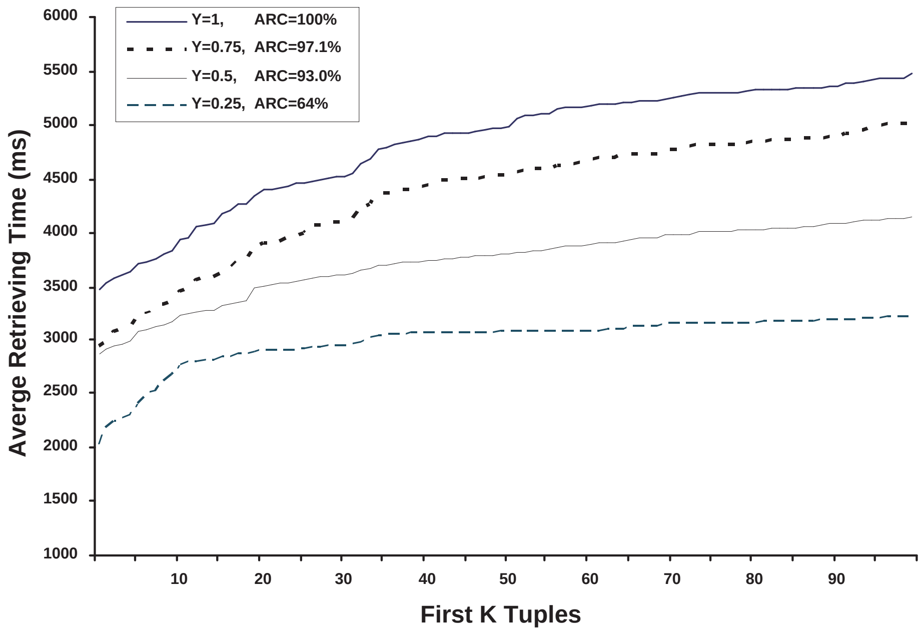


Figure 11. Effect of the discount model and various discount factor values.

We can plot the query plans in Figure 11 and Figure 12 in the two dimensional space of *average relative coverage* and $\frac{1}{Average\ Latency}$, as shown in Figure 13. The plots show that the plans are non-dominated by each other.

In summary, this experimental evaluation shows that the latency values of different [source, binding pattern] combinations are a sound estimate of the future queries, and the joint optimization approach presented here is able to optimize the query plan in both cov-

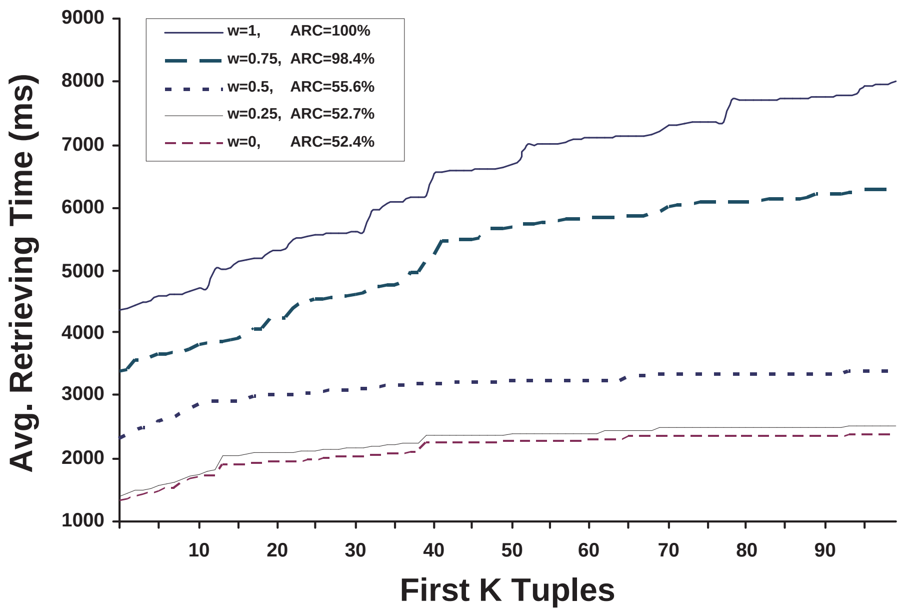


Figure 12. Effect of the weighted sum model and various weight factor values.

erage and latency and make flexible trade-offs according to the users' preferences regarding different objectives.

As both models can accommodate different user's preference on coverage and latency, either of them can be used in joint optimization in data aggregation systems. However, we noted some minor pros and cons of each them. The primary issue would be which parameter - γ or ω - is more natural and comfortable for the users to set. As shown in Figure 11 and Figure 13, when discount model is used, the average relative coverage of the query plans decreases smoothly with the linear decrease of the discount factor γ. When weighted sum model is used, the average relative coverage value decreases more sharply with the linear decrease of the weight factor ω (possibly due to the fact that we used the logarithm of the coverage value in the utility function). In web data aggregation systems, a user's choice of the γ and ω value is rather ad hoc and inaccurate. As a result, in appli-

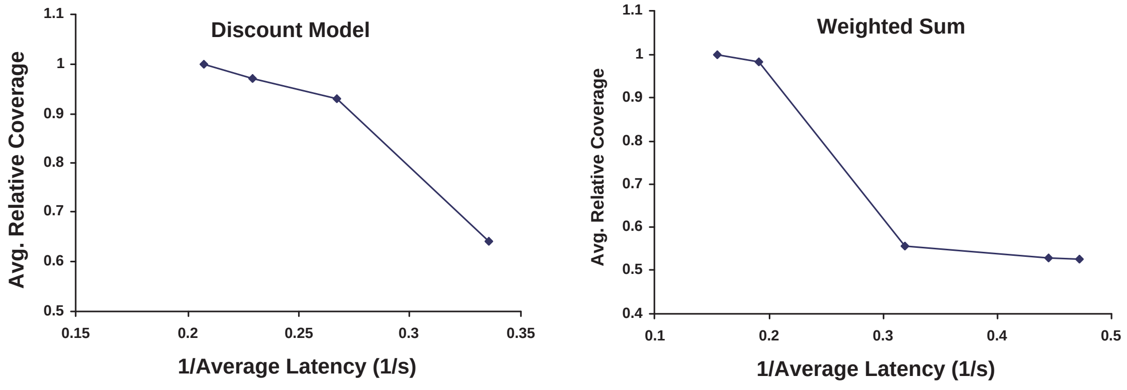


Figure 13. On average, the plans are non-dominated by each other.

cations where coverage is generally more important than latency, discount model may be safer for lay users, since the coverage of the query plans is less likely to drop drastically with small variations of γ value.

6.2. Joint Optimization of Coverage, Density and Latency

6.2.1. Plugging in 2D Coverage. We have shown in Chapter 4 that the concept of coverage can be easily extended to 2D coverage by jointly considering coverage and density. Naturally we can simply plug the 2D coverage into our joint optimization models of coverage/latency.

Specifically, the discount model using 2D coverage will be:

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

Similarly the weighted sum model using 2D coverage will be:

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

6.2.2. Experimental Evaluation. We conducted an experimental evaluation of joint optimization of coverage, density and latency on the synthetical test bed as used in Chapter 4. To simulate the source latency in this synthetic test bed, each of the sources is given a constant latency value. This latency value is randomly generated when the source is created, ranging from 100 to 8000 milliseconds. When the selected sources process the given queries, they intensionally make a delay before they send the results back to the mediator. Note that the way we simulate latency values for the sources are not query sensitive, since the purpose here is mostly to verify multiple objective trade-offs. However our finer granularity, binding pattern based latency statistics (as discussed in Chapter 5) can be seamlessly used here too.

Since we have already validated the method jointly optimizing and making trade-offs between coverage and density, the problem is reduced down to evaluating whether or not the joint optimization model makes flexible trade-offs between the latency and 2D coverage objectives. We therefore use experiments that are similar to the ones in Section 6.1.2.

We chose 20 realistic queries on the used car database, made query plans using each of the two utility functions, 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 in query planning. Figure 11 shows the average time for retrieving the first K tuples of the 200 testing queries together with their average *relative coverage* (ARC), with different discount factors. Figure 12 shows the results of using the weighted sum model.

In both figures, when 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 decreases in the discount model, the 2D coverage estimates of the sources are more and more discounted with the latency. As a result, the plans generated tend to favor the faster sources, but the average 2D 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/overlap and density statistics and only looks for the fastest sources. As shown in Figure 12, such plans retrieve the first K tuples fastest on average.

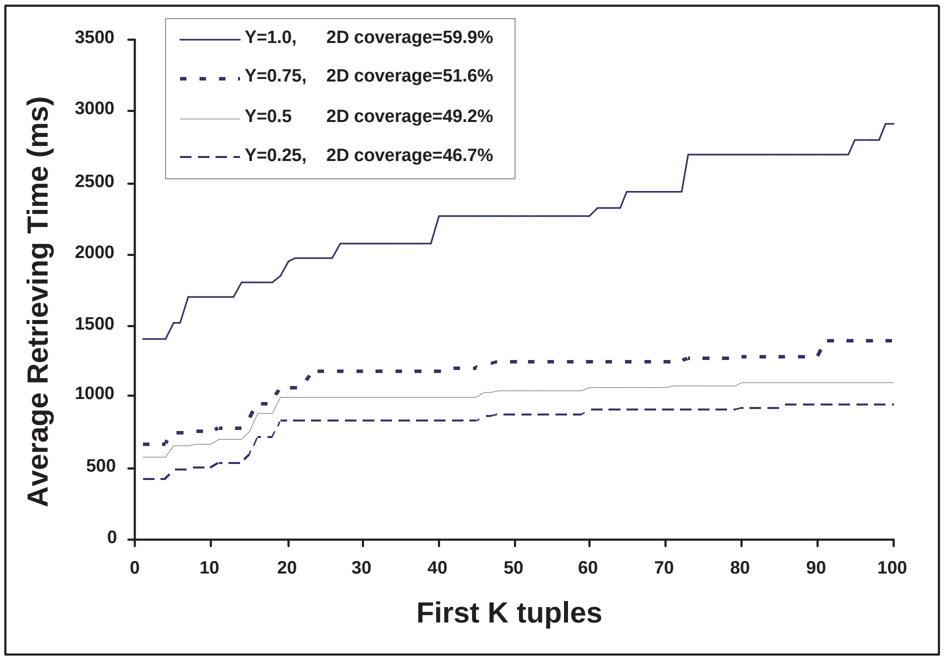


Figure 14. Effect of the discount model and various discount factor values.

In summary, this experimental evaluation shows that the joint optimization approach presented here is able to optimize the query plan in both 2D coverage and latency. Therefore it handles the problem flexible trade-offs between coverage, density and latency

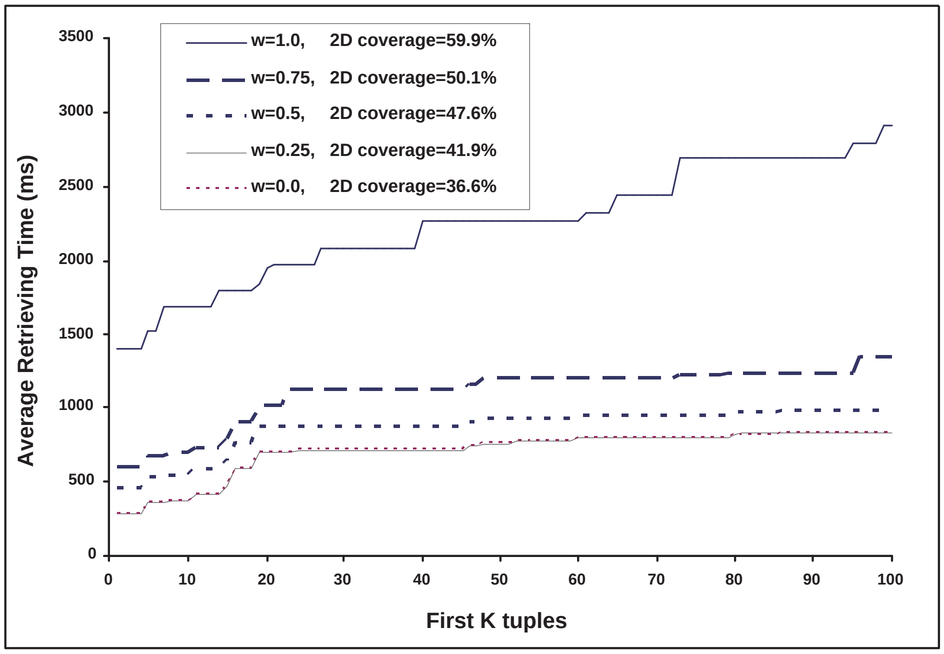


Figure 15. Effect of the weighted sum model and various weight factor values.

objectives according to the different users' own preferences regarding these objectives.

CHAPTER 7

MAINTAINING SOURCE STATISTICS

Online data sources and the mediator systems usually evolve continuously, thus the statistics of the sources kept by the mediator have to be updated accordingly to keep their accuracy. For example, the data sources might upgrade their services and the network topology might change, and as a result the source latency statistics will also change accordingly. Similarly, the content of the data sources or user interest patterns might change over time and thus the coverage/overlap statistics have to be updated to capture the *recent* frequent query classes.

All these changes bring up the need for maintaining statistics over time. In our framework, maintaining source latency statistics is straightforward. The way latency statistics are learned is a simple aggregation of the latency values of individual queries. Naturally, the maintenance of such statistics can be integrated into the regular query processing of the mediator.

Similarly, maintaining source density statistics is simple too. There is no interaction between the density statistics from different sources, and because of the two assumptions we made in Chapter 4, source density statistics can be learned efficiently as long as the mediator maintains some sample records for each database[1]. Such samples can be obtained

[1]Admittedly, the sample of the database could be biased based on the sampling techniques that are

during the daily query processing of the mediator. The mediator just needs to periodically refresh the samples that it maintains for each source to update the density statistics accordingly.

On the other hand, the frequency based coverage/overalap learning method of *StatMiner* relies on looking for certain patterns from a rather large group of queries and learning from their group characteristics. Thus it is more challenging to efficiently maintain these statistics. We will first motivate the need for active maintenance of these statistics and then introduce an incremental update approach to reduce the cost of maintenance but at the same time keep the accuracy of the statistics.

7.1. Motivation

StatMiner controls the cost of learning and storage of coverage/overlap statistics by only learning them on "frequent query classes". Therefore at least two types of change in the system will have impact on the accuracy of the learned coverage/overlap statistics. First, the content of the data sources changes all the time and consequently their actual coverage/overalp statistics of a given query will change also. Second, when the mediator's user interest patterns change, the actual *frequent query class set* will change accordingly, i.e., some query classes may no longer be accessed as frequently and some previously infrequent query classes may suddenly get more attention.

We focus on methods to handle the second type of changes in this paper, since the effectiveness *StatMiner* approach is more sensitive to this type of changes. Moreover, changes in user interest are very common in *Bibfinder* scenario because new "hot spots" in computer science tend to emerge constantly. Meanwhile, since our statistics maintenance

used. However based on Assumption 1 (the density statistics of an attribute is insensitive to its values), the biased samples is not likely to cause serious deterioration of the density statistics.

method uses the most recent queries in the query log to update the statistics, the change in source content that is projected onto the recent queries will be eventually integrated into the statistics. Therefore it also handles the source content changes partially and passively.

To have a more quantitative picture of whether and how user interest changes, we used *StatMiner* to identify frequent query classes over different time slices of the query log in *Bibfinder*. Using the same threshold values, *StatMiner* identified 1129 frequent query classes from the first 15% queries from the query log and 458 from the next 15% queries. Among the latter 458 frequent query classes, 84.5% of them also appear in the first 1129 frequent query classes, but the other 15.5% are new frequent query classes. This shows a drift of user interest over time.

To further illustrate the necessity of updating the statistics with such changes, we used *StatMiner* to learn the coverage/overlap statistics from the first $20,000$ queries in the query log. These statistics are used as the baseline statistics. We updated the statistics periodically by reinvoking *StatMiner* with the most recent queries as the training data. For every 200 queries along the time line of the query log, we measured the average *estimation distance* between the estimated and the real coverage/overlap values, using baseline statistics and updated statistics respectively. The *estimation distance* is computed using the following formula:

$$\frac{\sum_{Q \in TestQuerySet} \sqrt{\sum_i [P'(\widehat{S}_i|Q) - P(\widehat{S}_i|Q)]^2}}{|TestQuerySet|}$$

$\widehat{S}_i$ denotes the i^{th} source set of all possible source sets in the mediator, $P'(\widehat{S}_i|Q)$ denotes the estimated overlap (or coverage) of the source set $\widehat{S}_i$ for query Q, $P(\widehat{S}_i|Q)$ denotes the real overlap (or coverage, when $\widehat{S}_i$ contains only one source)[2] of the source set $\widehat{S}_i$ for query

[2]Each test query is sent to all the data sources, and the number of tuples returned by each source and the number of common tuples among each possible "source set" are recorded. We then use these numbers to compute the real coverage and overlap statistics for the queries

Q, and $TestQuerySet$ refers to the set of 200 queries at each test point along the time line of query log.

Figure 16 shows the comparison of the average estimation distances for the first 8% queries in the query log time line. We can see that without updating the statistics, the average estimation distance became higher as time passed. By updating the statistics, the mediator gets more accurate estimates for the future queries.

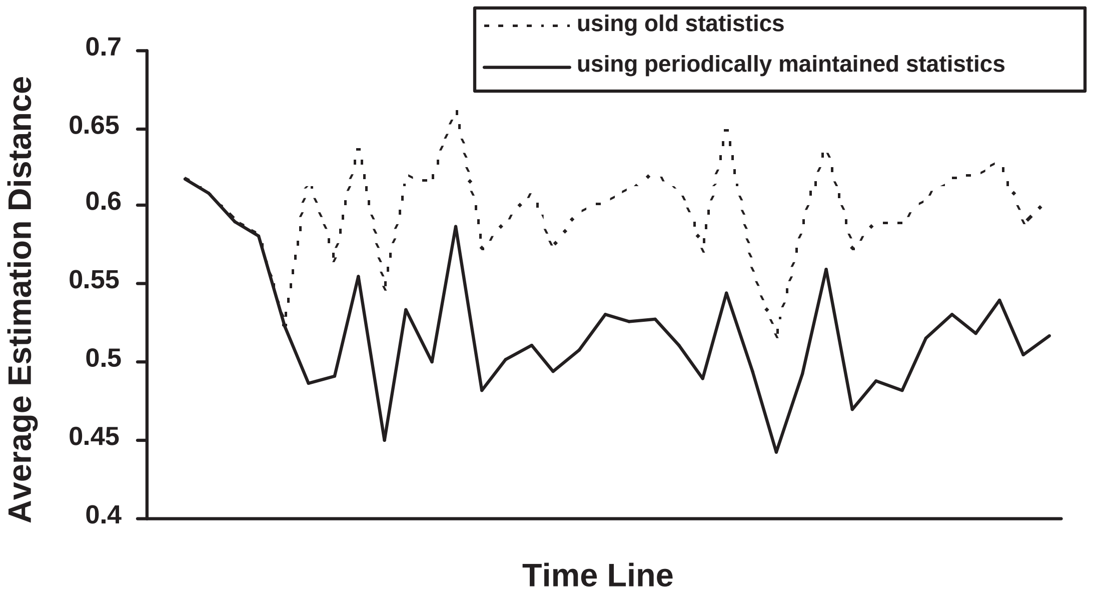


Figure 16. Average estimation distance.

7.2. Approach of Incremental Statistics Maintenance

The tests above show that to keep statistics accurate, the mediator must update the previously mined *frequent query class set* to track the user interest. The effectiveness of *StatMiner* is based on the assumption that the mediator can predict the near future with the statistics learned from the immediate past. Therefore, as used in the second test above, a straightforward updating mechanism is to reinvoke the learning process periodically, using the queries form a recent “time window” as the training data. The queries in this window represent the recent past and the statistics learned from them capture the recent interests of the user population. How frequent such update happens is also important, so we introduce a “sliding window” update model as shown in Figure 17. In this model, the *window period* represents the recent fragment of the query log from which the statistics are mined. The *shift interval* represents the time interval between consecutive updates. The window period needs to be large enough so that the learning technique can mine useful statistics and the shift interval has to be small enough so that the recent change of user interests can be captured in a timely manner. We will discuss briefly later about how the size of window period and shift interval are chosen in our setting.

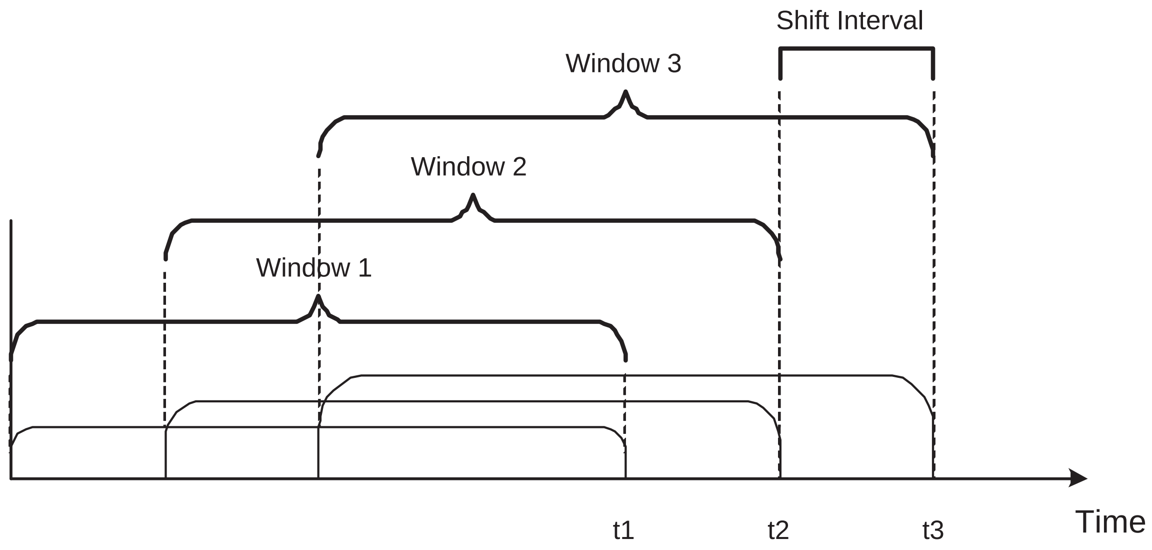


Figure 17. Sliding window in query log.

One remaining problem is that with a relatively large window period, the *StatMiner* learning algorithm is still costly. Moreover, we can observe that when the window period is much larger than the shift interval, there is a very big overlap between two consecutive windows. Intuitively the statistics we learn from two consecutive windows should be very similar. Redoing the entire learning process on the new window would largely repeat the work that has been done in the learning process on the previous window. This observation motivates our investigation on the incremental learning algorithm. Specifically, we keep the statistics learned previously and use only the queries in the most recent shift interval as the training data to calibrate the existing statistics and get the new statistics of the current window. Since the shift interval is much smaller than window period, such training could be less costly than regular learning on the entire window. More importantly, since it only partially changes the existing statistics, it is less likely to have the problem of introducing too much noise (due to the small training set) to the statistics.

In Section 7.3, we will show that incremental maintenance of the statistics will be less costly but still retains the quality of the statistics in terms of their effectiveness in query planning.

StatMiner controls the cost of learning and storage by using query classes as the abstraction of individual queries. Recall from Chapter 3 that the space of query classes is the cartesian product of all AV hierarchies, and that the *frequent query class set* is just a subgraph of this cartesian product. Updating the statistics requires changing the composition of this subgraph to reflect the current user interest patterns.

StatMiner identifies such a subgraph by searching for the frequent classes in the entire class. The size of that space is exponential in the number of attributes and linear in the domain size of each attribute. Such a large space makes the searching costly. Our incremental learning approach tries to limit the space of searching in the vicinity from the existing frequent class hierarchy. Specifically we can observe that:

- If a query class C has been very frequently accessed lately but it is not currently considered a frequent class, then the closest ancestors of this class which are already in the frequent class set must be very frequently accessed too because the queries that belong to C are mapped to the ancestors instead.

- If an existing frequent class is no longer frequently accessed, then its descendants that are also in frequent class set are not frequently accessed lately either.

Figure 18 shows a very simple query class hierarchy and its change over time. To use these features to incrementally update the frequent class set, the mediator needs to maintain the following information for each of the previously discovered frequent query classes:

- The coverage and overlap statistics.
- The estimated access frequency of this class, which is the access frequency during the previous learning cycle and is used as the expected access frequency from then on.
- The real access frequency of this class since the previous learning cycle is over. This frequency is accumulated as the mediator keeps processing the new queries posted to it, and mapping the queries to the existing frequent classes to make query plans.

The mediator invokes the algorithm 1 at the end of each shift interval to update the composition of frequent query class set and statistics of the classes in it.

This algorithm takes the existing frequent query class subgraph as the base of the update. It takes all the query classes in this subgraph as the candidates of the infrequent classes to be removed, and the children of them as the candidates of new frequent classes to be added. It then decides the new frequent query class subgraph by checking the real access frequency of these candidates, and calibrates the statistics for the qualified ones.

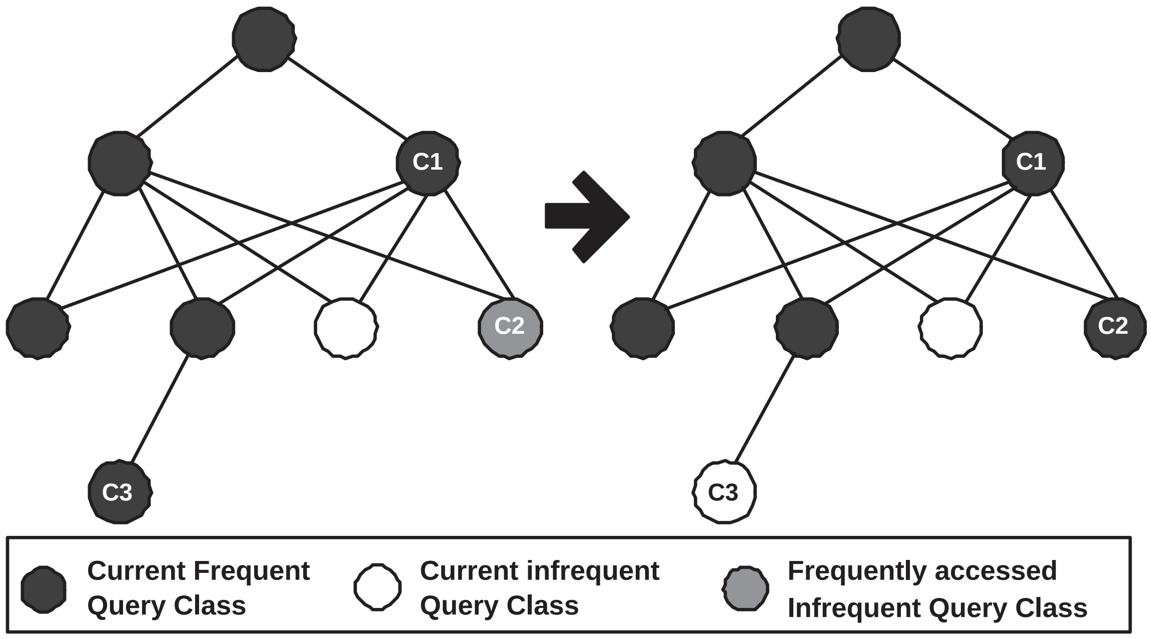

C1
C2
C3
C1
C2
C3
Current Frequent Query Class
Current infrequent Query Class
Frequently accessed Infrequent Query Class

Figure 18. A simple example showing the change of *frequent query class set*.

Once the learning process is over, the mediator resets the bookkeeping of the frequent query classes and trigger another update at the end of next shift interval. Note that this algorithm is a threshold based one and the thresholds are set manually at present. In most of our testing T_{split} and T_{merge} are set as 10, and $T_{minFreq}$ is set as 0.0023. As shown in [30] and our experiments, the effectiveness of the statistics is not very sensitive to the values of these thresholds.

Since we only inspect the *direct* children of current frequent classes instead of all their descendants, the new frequent query class set may not include all the classes that have been accessed with a frequency higher than given threshold. Therefore, this approach is not globally optimal in terms of identifying all the recent frequent classes. We also considered more complex variations of this algorithm. For example, instead of searching only the direct children, we can search the grandchildren also; or we can delay the removal of the infrequent query classes from the frequent class set until the classes have been infrequent

Algorithm 1 Incremental Update of Statistics

```
Require: $FQCS$: Frequent Query Class Set; $QList$: Query Log of the recent window;
  $T_{split}$: Threshold to Split; $T_{merge}$: Threshold to Merge; $T_{minFreq}$: Minimum Access
  Frequency Threshold;
  begin
  Let $addList = \emptyset$, $removeList = \emptyset$;
  for all $C \in FQCS$ do
    $C.realFrequency = \frac{C.numOfAccess}{|QList|}$;
    if $C.realFrequency > C.estimatedFrequency * T_{split}$ then
      $addList = addList \cup C.children$;
    else if $C.realFrequency < \frac{C.estimatedFrequency}{T_{merge}}$ then
      $removeList = removeList \cup \{C\}$;
    end if
  end for
  $FQCS = (FQCS \cup addList) - removeList$;
  $map\ Q \in QList\ to\ FQCS\ and\ compute\ coverage/overlap$
  end
```

for several consecutive updating cycles (to accommodate the natural fluctuation of certain query classes). However, our experiments show that these variations cost more but the statistics learned do not show very significant improvement in their quality. Thus we decided to just keep the incremental maintenance algorithms simple. This of course results from our trade-off between optimality and cost of learning. We will show in the next section that this approach cuts the cost significantly but does not sacrifice too much quality of the statistics.

7.3. Empirical Evaluation

We empirically evaluated the effectiveness of the incremental statistics maintenance approach on the *Bibfinder* test bed. We will demonstrate that the incremental learning approach is able to reduce the learning cost while retaining accuracy of statistics.

7.3.1. Experiment Setting.

7.3.1.1. *Data Sources.* *Bibfinder* currently integrates 7 structured web bibliography sources. We use a subset of 5 sources in our evaluation: *ACM Digital Library*, *DBLP*, *CSB*, *Network Bibliography* and *ScienceDirect*. We used the real user queries query log to train and test our algorithms.

7.3.1.2. *Evaluation Metrics.* We compared the cost of incremental learning and regular *StatMiner* learning method in terms of time spent in learning. The effectiveness of the statistics is evaluated according to how good the source selection plans are. The goodness of a plan, in turn, is evaluated by calling the sources specified in the plan as well as all the other sources available to the mediator. We define the *plan precision* of a plan to be the fraction of sources in the source selection plan, which turn out to be the true top K sources after we execute the query. We also use *plan coverage* to measure the fraction of the answers of the real top K sources the plan is able to cover. Let $Num(p)$ be the number of answers returned by executing plan p and $Num(topK)$ be the number of answers returned by the real top K sources for a query. We define *plan coverage* to be $\frac{Num(p)}{Num(topK)}$.

7.3.2. Experimental Results.

7.3.2.1. *Efficiency of Incremental Updating.* We used the statistics learned from the first 3% queries from the query log as the baseline statistics. We use 3% of the size of the query log as the size of window period, and 0.5% of the size of the query log as the size of shift interval. We updated the statistics using the sliding window model using regular *StatMiner* learning and the incremental learning algorithm on parallel. Figure 19 shows the comparison of learning cost at each of 7 consecutive update points. With the given sizes of window period and shift interval, the cost of incremental learning is always a very small fraction of that of regular learning (less than 20% overall in this setting).

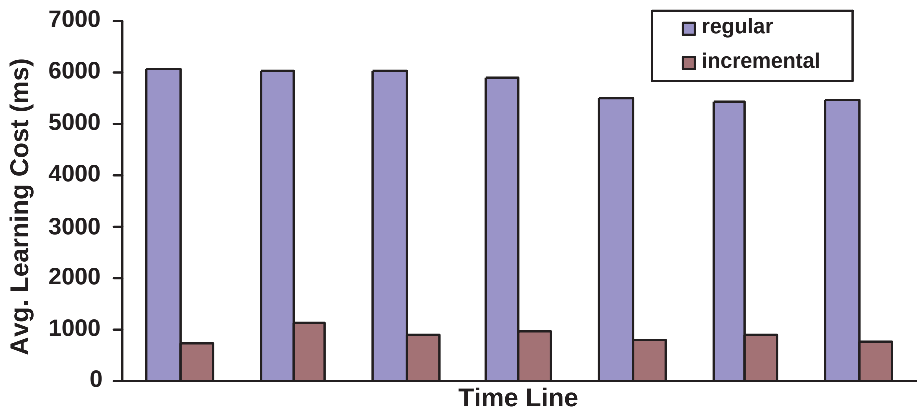
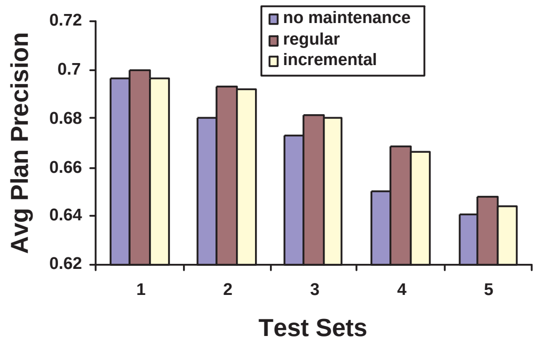


Figure 19. Average learning cost.

7.3.2.2. *Quality of Updated Statistics.* We evaluate the quality of the updated statistics by looking at the quality of the source selection plans generated by using these statistics. In this experiment, we took the statistics learned from regular *StatMiner* learning algorithm on the first 14% as the baseline statistics, and used the next 5% of the query log as the training data for regular *StatMiner* to get the updated statistics. We used the same training data for incremental update approach where the size of shift interval is 1% of the query log size (thus the statistics were updated 5 times incrementally at the end of the 5% training data) to get the incrementally updated statistics. We then used the next 1% (around $2,000$) queries as the test queries and divided them into 5 test sets with 400 queries in each. We used the three kinds of statistics above to generate source selection plans (asking for top 2 sources) for the test queries and compared the average plan coverage plan precision for each of the test sets.

Figure 20 shows the results. In all of the 5 test sets, the mediator gets source selection plans with higher average coverage and precision by using updated statistics instead of old statistics. Moreover, when using incrementally maintained statistics, the mediator gets source selection plans that are almost as good as the ones generated by using statistics updated with regular *StatMiner* learning approach, but with much lower costs.



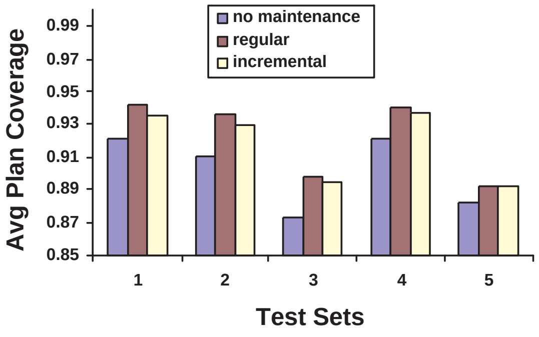


Figure 20. Average plan precision and plan coverage using different statistics.

In our evaluation, the size of the window period controls the size of the training data for the regular learning algorithm. The size of shift interval controls the frequency of the update for both regular and incremental update algorithm, as well as the size of the training data for the incremental update algorithm. Both of the sizes are currently set manually. Our evaluation above uses two different pairs of values of them, and both of the evaluation shows the effectiveness of the incremental updates. We have tried a wider range of the values of the sizes of window period and shift interval (in terms of the percentage of the entire query log) and the results are not significantly affected by these values. We are also looking into using techniques similar to the ones used in text database monitoring [22] to dynamically and systematically decide the values of the sizes of window period and shift interval.

In summary, the comparison on the metrics above is consistent with our initial expectation of incremental statistics maintenance, i.e. the cost of learning is significantly lower but the quality of statistics learned is almost as good as learning everything from scratch periodically.

CHAPTER 8

CONCLUSION AND FUTURE WORK

In this paper we described a comprehensive adaptive data aggregation framework that can effectively mine, use and maintain various types of source statistics. This framework is adaptive to the dynamic and autonomous nature of the data sources, varying user preferences, as well as the shifting interest and access patterns of the user population.

We first presented the statistics mining approaches in this framework. We leverage the existing *StatMiner* approach to automatically learn source coverage/overlap statistics with respect to a set of frequently accessed query classes. We proposed a source density model to measure the database tuple level completeness of the sources. We also developed new techniques to learn *query sensitive* source latency statistics. By using these statistics, the mediator is able to make quality estimates for the new queries and optimize different query objectives (specifically, high coverage, high density or low cost) respectively.

To be adaptive to multi-objective query processing requirements, we developed a novel multi-objective optimization model that can use coverage/overlap, density as well as latency statistics simultaneously to jointly optimize coverage, density and execution latency in source selection.

Moreover, to be adaptive to the dynamically changing system, we developed an incremental statistics maintenance algorithm to handle the change of interest in the user population. This approach significantly reduces the cost of maintaining coverage/overlap statistics while retaining their accuracy.

We provided compelling empirical evaluations showing the effectiveness of these techniques in learning, using and maintaining a wider range of statistics to support flexible and adaptive query processing.

Our framework addresses three of the most important aspects of query processing in data aggregation systems: *coverage*, *density* and *latency*. Density is correlated to the incompleteness of autonomous databases caused by missing attribute values. In related work, we have also been working on the problem of retrieving highly relevant uncertain answers when critical attribute values are missing. The uncertain answers retrieved in such a scenario therefore have the characteristic of *confidence* measure. When a mediator also takes *confidence* into the procedure of source selection, how to jointly optimize confidence and all other objectives simultaneously presents a challenging problem to be further investigated.

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] Google scholar. http://scholar.google.com/.

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

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

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

[10] Yahoo! autos. http://autos.yahoo.com.

[11] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *SIGMOD Conference*, pages 181–192, 1999.

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

[13] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[14] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *SIGIR*, pages 21–28, 1995.

[15] CiteSeer. Computer and information science papers. http://www.citeseer.org.

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

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

[18] V. Ganti, M.-L. Lee, and R. Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *VLDB*, pages 176–187, 2000.

[19] 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.

[20] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *VLDB*, pages 419–428, 2000.

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

[22] P. G. Ipeirotis, A. Ntoulas, J. Cho, and L. Gravano. Modeling and managing content changes in text databases. In *ICDE*, pages 606–617, 2005.

[23] 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.

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

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

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

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

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

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

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

[31] Z. Nie, S. Kambhampati, and T. Hernandez. Bibfinder/statminer: Effectively mining and using coverage and overlap statistics in data integration. In *VLDB*, pages 1097–1100, 2003.

[32] 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*, 2004.

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

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

[35] 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.

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

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