BibFinder/StatMiner: Effectively Mining and Using Coverage and Overlap Statistics in Data Integration*

Zaiqing Nie Subbarao Kambhampati Thomas Hernandez

Department of Computer Science and Engineering
Arizona State University,Tempe, AZ 85287-5406
{nie, rao, th}@asu.edu

Abstract

Recent work in data integration has shown the importance of statistical information about the coverage and overlap of sources for efficient query processing. Despite this recognition there are no effective approaches for learning the needed statistics. In this paper we present StatMiner, a system for estimating the coverage and overlap statistics while keeping the needed statistics tightly under control. StatMiner uses a hierarchical classification of the queries, and threshold based variants of familiar data mining techniques to dynamically decide the level of resolution at which to learn the statistics. We will demonstrate the major functionalities of StatMiner and the effectiveness of the learned statistics in *BibFinder*, a publicly available computer science bibliography mediator we developed. The sources that *BibFinder* integrates are autonomous and can have uncontrolled coverage and overlap. An important focus in *BibFinder* was thus to mine coverage and overlap statistics about these sources and to exploit them to improve query processing.

1 Introduction

With the vast number of autonomous information sources available on the Internet today, users have access to a large variety of data sources. Data integration systems 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. In a data integration scenario, the user interacts with a mediator system via a mediated schema. A mediated schema is a set of virtual relations, which are effectively stored across multiple and potentially overlapping data sources, each of which may only contain a partial extension of the relation. Query optimization in data integration thus requires the ability to figure out what sources are most relevant to the given query, and in what order those sources should be accessed [FKL97, NLF99, NK01, DH02]. For this purpose, the query optimizer needs access to statistics about the coverage of the individual sources with respect to the given query, as well as the degree to which the answers they export overlap. We illustrate the

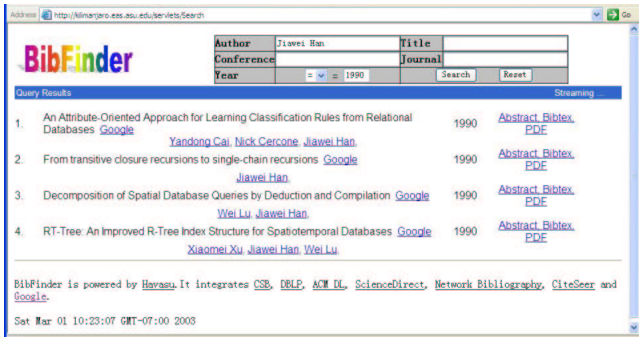

Figure 1: *The BibFinder User Interface*

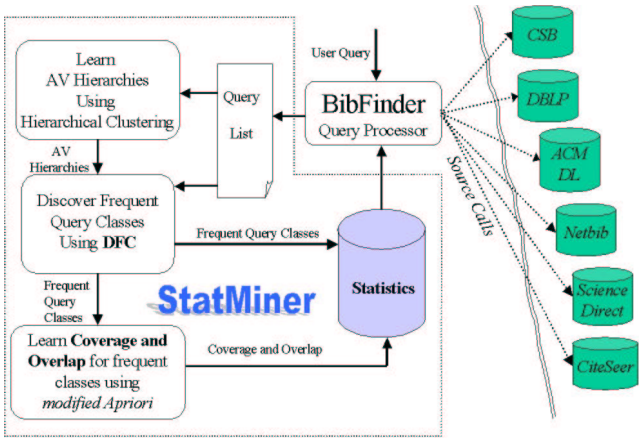

Figure 2: *Bibfinder/StatMiner Architecture*

*This research is supported in part by the NSF grant IRI-9801676 and the ASU ET-I[3] initiative grant ECR A601. We thank Ullas Nambiar, Sreelakshmi Vaddi for comments as well as help in a previous implementation of *Statminer*, and Louiqa Raschid, Huan Liu, K. Selcuk Candan for many helpful critiques.

**Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003**

need for these statistics with an example.

BibFinder Example: We have been developing *BibFinder* (http://rakaposhi.eas.asu.edu/bibfinder, Figure 1), a publicly available computer science bibliography mediator. *BibFinder* integrates several online Computer Science bibliography sources. It currently covers *CSB, DBLP, Network Bibliography, ACM Digital Library, ScienceDirect,* and *CiteSeer.* Plans are underway to add several additional sources including *IEEE Xplore* and *Computational Geometry Bibliography*.

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. However it also brings some interesting optimization challenges. Let us assume that the global schema exported by *BibFinder* includes just the relation:

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

Each of the individual sources only export a subset of the global relation. For example, *Network Bibliography* only contains publications in Networks, *DBLP* gives more emphasis on Database related publications, while *ScienceDirect* only has archival journal publications etc. To efficiently answer users' queries, *BibFinder* needs to find and access the most relevant subset of the sources for the given query. Suppose, the user asks a selection query:

Q(title,author) :– **paper**(title, author, conf/journal, year),
conf="AAAI".

To answer this query efficiently, *BibFinder* needs to know the *coverage* of each source S with respect to the query Q, i.e. $P(S|Q)$, the probability that a random answer tuple for query Q belongs to source S. Given this information, we can rank all the sources in descending order of $P(S|Q)$. The first source in the ranking is the one that *BibFinder* should access first while answering query Q. Although ranking seems to provide the complete order in which to access the sources, this is unfortunately not true in general. It is quite possible that the two sources with the highest coverage with respect to Q happen to mirror each others' contents. Clearly, calling both sources is not going to give any more information than calling just one source. Therefore, after we access the source S' with the maximum coverage $P(S'|Q)$, we need to access as the second source, the source S'' that has the highest *residual coverage* (i.e., provides the maximum number of those answers that are not provided by the first source S'). Specifically we need to pick the source S'' that has next highest coverage w.r.t. Q but has minimal *overlap* (common tuples) with S'.

Given that sources tend to be autonomous in a data integration scenario and that the mediation may or may not be authorized, it is impractical to assume that the sources will automatically export coverage and overlap statistics. Consequently, data integration systems should be able to learn the necessary statistics. Although previous work has addressed the issue of how to model these statistics (c.f. [FKL97]), and how to *use* them as part of query optimization (c.f. [NLF99],[NK01],[DH02]), there has not been any work on effectively learning the statistics in the first place.

In our research, we address the problem of learning the coverage and overlap statistics for sources with respect to user queries. A naive approach may involve learning the coverages and overlaps of all sources with respect to all queries. This will necessitate $N_q * 2^{N_S}$ different statistics, where N_q is the number of different queries that the mediator needs to handle and N_S is the number of data sources that are integrated by the mediator. An important challenge is to keep the number of statistics under control, while still retaining their advantages in query optimization.

We will demonstrate *StatMiner* (see Figure 2), a statistics mining module for web based data integration. *StatMiner* is being applied to *BibFinder* to help users find CS papers efficiently. *StatMiner* comprises of a set of connected techniques that estimate the coverage and overlap statistics while keeping the amount of needed statistics tightly under control. Since the number of potential user queries can be quite high, *StatMiner* aims to learn the required statistics for *query classes* i.e. groups of queries. By selectively deciding the level of generality of the query classes with respect to which the coverage statistics are learned, *StatMiner* can tightly control the number of needed statistics (at the expense of loss of accuracy). The loss of accuracy may not be a critical issue for us as it is the *relative* rather than the *absolute* values of the coverage statistics that are more important in ranking the sources.

The rest of this paper is organized as follows. In Section 2 we describe our approach for modeling coverage and overlap between sources in terms of query classes. In Section 3, we describe *StatMiner*, which uses this model to automatically mine source statistics. In Section 4, we describe what we plan to demonstrate.

2 Modeling Coverage and Overlap w.r.t. Query Classes

Our approach consists of grouping queries into abstract classes. In order to better illustrate the novel aspects of our association rule mining approach, we purposely limit the queries to just projection and selection queries.

2.1 Classifying Mediator Queries

Since we are considering selection queries, we can classify the queries in terms of the selected attributes and their values. To abstract the classes further we assume that the mediator has access to the so-called "attribute value hierarchies" for a subset of the attributes of each mediated relation.

Attribute Value Hierarchies: An *AV hierarchy* (or attribute value hierarchy) over an attribute A is a hierarchical classification of the values of the attribute A. The leaf nodes of the hierarchy correspond to specific concrete values of A. For numerical attributes, we can take value ranges as leaf nodes. While the non-leaf nodes are abstract values that correspond to the union of values below them. Figure 3 shows the AV hierarchies for the "conference" and "year" attributes of the "paper" relation.

Note that hierarchies do not have to exist for every attribute, but rather only for those attributes over which queries are classified. We call these attributes the **classificatory attributes**. We can choose as the classificatory attributes the best k attributes whose values differentiate the

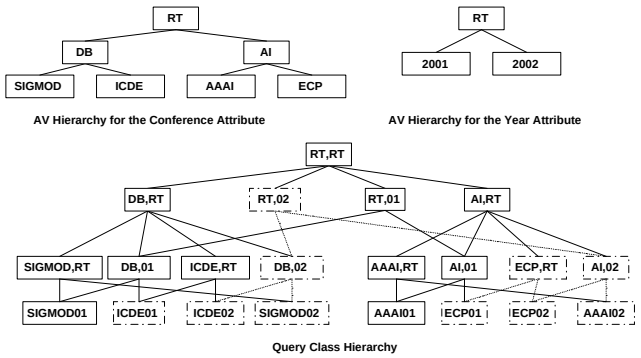

Figure 3: *AV Hierarchies and the Corresponding Query Class Hierarchy*

sources the most, where the number k is decided based on a tradeoff between prediction performance versus computational complexity of learning the statistics by using these k attributes. The selection of the classificatory attributes may either be done by the mediator designer or using automated techniques. The AV hierarchies themselves can either be hand-coded by the designer, or can be learned automatically (see Section 3).

Query Classes: Since we focus on selection queries, a typical query will have values of some set of attributes bound. We group such queries into query classes using the AV hierarchies of the classificatory attributes. A query **feature** is defined as the assignment of a classificatory attribute to a specific value from its AV hierarchy. A feature is "abstract" if the attribute is assigned an abstract (non-leaf) 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 all the classificatory attributes. Specifically, let H_i be the set of features derived from the AV hierarchy of the i^{th} classificatory attribute. Then the set of all query classes (called $classSet$) is simply $H_1 \times H_2 \times ... \times Hn$. The AV hierarchies induce subsumption relations among the query classes. A class C_i is subsumed by class C_j if every feature in C_i is equal to, or a specialization of, the same dimension feature in C_j. A query Q belongs to a class C if the values of the classificatory attributes in Q are equal to or are specializations of the features defining C. Figure 3 shows an example class hierarchy for a very simple mediator with the two example AV hierarchies. The query classes are shown at the bottom, along with the subsumption relations between the classes.

Query List: We assume the mediator maintains a query list *QList*, which keeps track of the user queries and their access frequency. We use FR_Q to denote the access frequency of a query Q, and FR to denote the total frequency of the all the queries in QList. The *query probability* of a query Q, denoted by $P(Q)$, is the probability that a random query posed to the mediator is the query Q. It can be computed using the formula: $P(Q) = \frac{FR_Q}{FR}$. The *class probability* of a class C, denoted by $P(C)$, is the probability that a random query posed to the mediator is subsumed by the class C. It can be computed as: $P(C) = \sum_{Q \in C} P(Q)$.

2.2 Source Coverage and Overlap w.r.t. Query Classes

The *coverage* of a data source S with respect to a class C, denoted by $P(S|C)$, is the probability that a random tuple belonging to class C is present in source S. We assume that the union of contents of the available sources within the system covers 100% of the class. In other words, coverage is measured relative to the available sources. The *overlap* among a set $\widehat{S}$ of sources with respect to a class C, denoted by $P(\widehat{S}|C)$, is the probability that a random tuple belonging to the class C is present in each source $S \in \widehat{S}$.

The coverage and overlap can be conveniently computed using an association rule mining approach. Specifically, we are interested in the class-source association rules of the form $C \rightarrow \widehat{S}$, where C is a query class, and $\widehat{S}$ is a (possibly singleton) set of data sources. The overlap (or coverage when $\widehat{S}$ is a singleton) statistic $P(\widehat{S}|C)$ is simply the "confidence" of such an association rule[1]. Examples of such association rules include: $AAAI \rightarrow S_1$, $AI \rightarrow S_1$, $AI\&2001 \rightarrow S_1$ and $2001 \rightarrow S_1 \wedge S_2$.

3 The StatMiner Architecture

In *StatMiner* (see Figure 2), the frequent query classes are discovered by using the DFC, an algorithm we developed to efficiently identify the query classes with sufficiently large support (for more information see [NK02]), and the coverage and overlap statistics using a variant of the Apriori algorithm [AS94]. The resolution of the learned statistics is controlled in an adaptive manner with the help of two thresholds. A threshold τ_c is used to decide whether a query class has large enough support to be remembered. When a particular query class doesn't satisfy the minimum support threshold, *StatMiner*, in effect, stores statistics only with respect to some abstraction (generalization) of that class. Another threshold τ_o is used to decide whether or not the overlap statistics between a set of sources and a remembered query class should be stored.

Specifically, *StatMiner* automatically builds AV hierarchies using an agglomerative hierarchical clustering algorithm (for more information see [NK02]) from the query list maintained by the mediator. The basic idea of generating an AV hierarchy is to cluster similar attribute values into classes in terms of the coverage and overlap statistics of their corresponding selection queries binding these values. Then the problem of finding similar attribute values becomes the problem of clustering similar selection queries. In order to cluster similar queries, we define a distance function to measure the distance between a pair of selection queries (Q_1, Q_2).

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

[1] Support and confidence are two measures of a rule's significance. The support of the rule $C \rightarrow \widehat{S}$ (denoted by $P(C \cap \widehat{S})$) refers to the percentage of the tuples in the global relation that are common to all the sources in set $\widehat{S}$ and belong to class C. The confidence of the rule (denoted by $P(\widehat{S}|C) = \frac{P(C \cap \widehat{S})}{P(C)}$) refers to the percentage of the tuples in the class C that are common to all the sources in $sourceSet$ $\widehat{S}$.

where $\hat{S}_i$ denotes the i^{th} source set. The interpretation of the distance function is that we consider two queries similar if their source coverage and overlap statistics are similar.

Using DFC we then classify user queries based on the learned AV Hierarchies and dynamically identify frequent classes for which the class probability is above the specified threshold τ_c. We learn and store statistics in the mediator's main memory only with respect to these identified frequent classes. When the mediator, in our case *BibFinder*, encounters a new user query, it maps the query to one of the query classes for which statistics are available. Since we use thresholds to control the set of query classes for which statistics are maintained, it is possible that there is no query class that exactly matches the user query. In this case, we map the query to the nearest abstract query class for which statistics are available. The loss of accuracy in statistics entailed by this step should be seen as the cost we pay for keeping the amount of stored statistics low. Once the query class corresponding to the user query is determined, the mediator uses the learned coverage and overlap statistics to rank the data sources that are most relevant to answering the query.

Details on how we use DFC to efficiently discover the frequent query classes by using the *anti-monotone property*, and how we use an item set mining algorithm, *Apriori*, to discover strongly correlated source sets for all the large classes can be found in [NK02]. That paper also reports empirical studies we conducted to evaluate the effectiveness of the *StatMiner* approach.

4 Description of the Demo

In this demo we will showcase *StatMiner*'s mining algorithm and show the use of the learned statistics in *BibFinder*, which makes use of *StatMiner* to learn coverage and overlap statistics to integrate its bibliography sources. These sources are partially overlapping both in tuples and attributes. For example, some sources may have the *bibtex* entry for the paper, some may have the pdf file for the paper, while others may have the abstract. By combining them, we can present a unified and more complete view to the user. Since the same paper's information may be stored partially and overlappingly over multiple sources, the testbed will provide an opportunity to exercise and evaluate all the StatMiner techniques. In *BibFinder*, users can ask selection queries on the following attributes: title, author, year, and conference/journal. *BibFinder* maintains a query list which keeps all the queries asked by the users. Figure 2 shows how *BibFinder* interacts with *StatMiner*. Our demonstration will focus on the following aspects:

Building AV hierarchies:. *StatMiner* assumes AV Hierarchies to classify the queries, and these hierarchies are difficult to hand-generate in many scenarios. We will show how StatMiner can automatically build complex hierarchies starting from the query list maintained by *BibFinder*. We will show the challenges and solutions of learning AV hierarchies for different types of attributes. In the context of *BibFinder*, we will show how we learn AV hierarchies for the title attribute, author attribute, year attribute and conference attributes.

Handling the Space and Accuracy Tradeoff: We will show that StatMiner can systematically trade time and space consumption of the statistics computation for accuracy by varying the coverage and overlap thresholds. We first show how significantly StatMiner can reduce the number of classes and association rules remembered by slightly increasing the thresholds. Then we will show that the learned statistics provide tangible improvements in the source ranking, and the improvement is proportional to the type (coverage alone vs. coverage and overlap) and granularity of the learned statistics.

Using the Learned Statistics in *BibFinder*: We will show that *BibFinder* can significantly reduce both the cost of generating all the answers and the cost of generating only partial results for most of user queries by using our coverage and overlap learned statistics. We will also show how the learned statistics can be used in *BibFinder* to avoid irrelevant or redundant source calls.

5 Conclusion

In this paper, we used *BibFinder* to motivate the need for automatically learning the coverage and overlap statistics of sources for efficient query processing in a data integration scenario. We then presented the *StatMiner* architecture for estimating the coverage and overlap statistics while keeping the needed statistics tightly under control. The contributions of *StatMiner* include: (1) developing a model for supporting a hierarchical classification of a set of queries, (2) an approach for estimating the coverage and overlap statistics using association rule mining techniques, and (3) a threshold-based modification of the mining techniques for dynamically controlling the resolution of the learned statistics. We explained how *StatMiner* is used as a backend to improve the query processing in *BibFinder*. Finally, we described the specific capabilities of *StatMiner* and *BibFinder* that we plan to demonstrate.

References

[AS94] Rakesh Agrawal, Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In *VLDB*, Santiago, Chile, 1994.

[DH02] A. Doan and A. Halevy. Efficiently Ordering Plans for Data Integration. In *Proceedings of ICDE-2002*, 2002.

[FKL97] D. Florescu, D. Koller, and A. Levy. Using probabilistic information in data integration. In *Proceeding of the International Conference on Very Large Data Bases* (VLDB), 1997.

[NK01] Z. Nie and S. Kambhampati. Joint optimization of cost and coverage of query plans in data integration. In ACM CIKM, Atlanta, Georgia, November 2001.

[NK02] Z. Nie and S. Kambhampati. Frequency-Based Coverage Statistics Mining for Data Integration. ASU CSE TR 02-004. Dept. of Computer Science & Engg. Arizona State University. $http://www.public.asu.edu/\sim zaiqingn/tech\_freqc.pdf$

[NLF99] F. Naumann, U. Leser, J. Freytag. Quality-driven Integration of Heterogeneous Information Systems. In *VLDB Conference* 1999.