Mining Source Coverage Statistics for Data Integration

Zaiqing Nie Subbarao Kambhampati Ullas Nambiar Sreelakshmi Vaddi
Department of Computer Science and Engineering
Arizona State University,Tempe, AZ 85287-5406
{nie, rao, mallu, slakshmi}@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. The key challenge in learning such statistics is keeping the number of needed statistics low enough to have the storage and learning costs manageable. Naive approaches can become infeasible very quickly. In this paper we present a set of connected techniques that estimate the coverage and overlap statistics while keeping the needed statistics tightly under control. Our approach 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 describe the details of our method, and present preliminary experimental results showing the feasibility of the approach.

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 [CGHI94, LRO96, ACPS96, LKG99, PL00] 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, a 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 only contain a partial extension of the relation. Query optimization in data integration [FKL97, NK01] thus requires the ability to figure out what sources are most relevant to the given query, and in what order those sources should be called. For this purpose, the query optimizer needs to access 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.

Example: Consider a simple mediator that integrates information sources exporting information about papers in computer science. Suppose there is one relation in the global schema of this system: **paper**(title, author, conference, year). There may be hundreds of Internet sources, each of which contain only a subset of the papers of the whole global relation. Some sources may only have information about artificial intelligence, some may focus on databases, etc. In order to answer the user's query efficiently, we need to find and query only the relevant subset of the sources.

Suppose, for example, the user asks a selection query:

Q(title,author) :− **paper**(title, author, conference, year), conference="AAAI".

To answer this query efficiently, we need to know the coverage of each source S with respect to the query Q. Such a coverage can be represented by $P(S|Q)$, the probability that an answer tuple for the query belongs to the source S. If we have this information, then we can rank all the sources in descending order of $P(S|Q)$, and access the first source (say S') in the ranking.

Although it would seem that the ranking provides the complete order in which to access the sources, this is unfortunately not true in general. For example, it is 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.

In general, after we access the source S' with the maximum coverage $P(S'|Q)$ w.r.t. Q, we need to access as the second source, the source that has the highest *residual* coverage (i.e., provides the maximum number of those answers that are not provided by the first source). Specifically, we need to access the source S'' that has the maximum value for $P(S'' \wedge \neg S'|Q)$. In order to compute $P(S'' \wedge \neg S'|Q)$, we need to know the overlap between sources S' and S'' w.r.t. to Q. This can be represented by the probability $P(S'' \wedge S'|Q)$. Once we have the probability, it is a simple matter to compute

$$P(S'' \wedge \neg S'|Q) = P(S''|Q) - P(S'' \wedge S'|Q)^1.$$

This example thus demonstrates the need for coverage and overlap statistics in query optimization. □

Given that sources tend to be autonomous in a data integration scenario, gathering the coverage and overlap statistics poses several challenges. It is impractical to assume that the sources will export such statistics. Consequently, data integration systems should be able to learn the coverages of sources. 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. [NK01]), there has not been any work on effectively learning the statistics in

[1]This formula can be generalized to compute $P(S_i \wedge \neg S_1 \wedge ... \wedge \neg S_k|Q)$, where k is the number of selected sources, and Q refers to a query.

.

the first place.

In this paper, we consider the issue of learning the coverage and overlap statistics for sources. The key challenge in this problem is to keep the number of needed statistics low enough to have the storage and learning costs manageable. Naive approaches can become infeasible very quickly. In the example above, we were assuming the availability of coverage statistics with respect to every source-query combination, and overlap information about every subset of sources with respect to a query!

We propose a set of connected techniques that estimate the coverage and overlap statistics while keeping the needed statistics tightly under control. The basic idea of our approach is to learn coverage statistics not with respect to individual queries but with respect to query classes. Specifically, we develop a hierarchical classification of queries starting with a hierarchical classification of the values of certain key attributes of the global relations. The class hierarchy allows us to approximate the coverage of a source with respect to a class C in terms of its coverage with respect to a more general class C'. By selectively deciding the level of generality of the classes with respect to which the coverage statistics are learned, we 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 statistics learning itself is done using threshold-based variants of the well known association rule mining techniques. The thresholds are used to decide whether to learn statistics with respect to a given class C, or one of its generalizations. Specifically, using thresholds on the support counts, we dynamically identify "large" classes, and learn coverage statistics only with respect to these classes. The resolution of the learned statistics is thus controlled in an adaptive manner. An interesting by-product of this adaptive approach is that by identifying classes with high support, it also provides a macro characterization of the focus areas of the mediator.

In the rest of the paper, we describe our approach and provide an empirical evaluation of its feasibility. The paper is organized as follows. In the next section, we give an overview of our approach and define the needed terminology. Next we discuss how the sources are probed to generate training data for mining statistics. We then describe the algorithms for learning coverage and overlap statistics. This is followed by an empirical evaluation of our approach. We end with a discussion of the related work and a summary of our contributions.

2. FRAMEWORK

In this section we give an overview of our approach and define the terminology we use in the paper. 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 Constructing Query Classes

Our approach consists of grouping queries into abstract classes. 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.

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. If we know the domains (or representative values) of multiple attributes, we can choose as the classificatory attribute the best k attributes whose values differentiate the sources the most[2], 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. For example, suppose a mediator system just has three sources: source S_1 only has papers in conference AAAI, S_2 only has papers in conference IJCAI, and S_3 only has papers in conference SIGMOD. In order to rank access to these sources, we need only choose the "conference" attribute as the classificatory attribute, even if we know the domain of the "year" attribute.

2.1.1 *Attribute Value Hierarchy*

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, while the non-leaf nodes are abstract values that correspond to the union of values below them. We use the term *feature* to describe the nodes of an AV hierarchy. *Classificatory Attributes* refer to the attributes for which we have AV hierarchies in the mediator. Continuing the example in Section 1, we shall assume that the mediator has two AV hierarchies (see Figure 1), one is for the "conference" attribute, and the other for the "year" attribute.

2.1.2 *Query Classes*

A *class* is a description of a subset of queries. The set of *primitive classes* is thus just the set of all queries. The set of *leaf classes* is just the set of all primitive queries with all classificatory attributes and only these classificatory attributes bound. Our interest is to define *abstract classes* that cover multiple queries. Let H_i denote the set of leaf node and non-leaf node features in the i-th AV hierarchy in the mediator. We shall assume that the special feature $ROOT$ corresponds to the root of the hierarchy. The set of abstract classes τ_c is just the cartesian product

$$H_1 \times H_2 \times ... \times Hn.$$

We call τ_c the *classSet* of the mediator.

The AV hierarchies induce subsumption relations among the abstract 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. Finally, a query Q belongs to a class C if the values of the classificatory attributes in Q are equal to or specializations of the features defining C.

Example For a mediator with two very simple feature hierarchies:

$H_1 = \{SIGMOD, DB, ROOT\}$ and

$H_2 = \{1994, 90-99, ROOT\}$,

the $classSet$ of the mediator will be

$$\tau_c = \{(SIGMOD, 1994), (SIGMOD, 90-99), (DB, 1994), (DB, 90-99), (SIGMOD), (DB), (1994), (90-99), ROOT\}$$

Here the class $C_1 = (SIGMOD, 1994)$ refers to all the SIGMOD'94 papers, and the class $C_2 = (DB, 90-99)$ refers to all the database papers published from year 1990 to 1999. As we can see C_1 is subsumed by C_2. C_1 and C_2 are classes with multiple features, each of which comes from a distinct AV hierarchy. The class $(SIGMOD)$=$(SIGMOD, ROOT)$ is a class with a single feature. □

[2]The selection of the attributes may either be done by the mediator designer or using automated techniques (such as decision tree learning techniques to learn their information gains of classifying the sources).

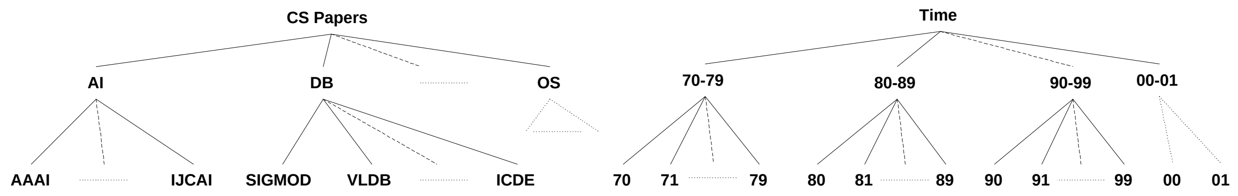

Figure 1: ***The Attribute Value hierarchies***

2.2 An Overview of Our Approach

In this section, we show how to learn the coverage and overlap statistics by using our association rule mining approach, and how to map a user's query to an abstract class with statistics.

2.2.1 Coverage and Overlap

The *coverage* of a data source for a class refers to the degree to which the source covers the class. We use the notation $P(S|C)$ to denote the coverage of source S for class C. We assume that the union of the 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 k sources for a class refers to the degree to which these sources cover the same part of the data in the class. We use the formula $P(S_1 \wedge S_2 \wedge ... \wedge S_k|C)$ to denote the overlap among source S_1, S_2,..., S_k for class C.

2.2.2 Mining association rules

In order to define the term association rule, we first define the term *source set*. Let $\tau_s = \{S_1, S_2, ..., S_m\}$ be a set of all the sources available to a mediator. A subset of τ_s is referred to as a source set. A source set that contains k sources is a k-$sourceSet$. For example, $\{S_1, S_2\}$ is a 2-$sourceSet$.

An *association rule* represents strong associations between a class and a sourceSet. It's an implication of the form $C \rightarrow \widehat{S}$, where $C \in \tau_c$ and $\widehat{S} \subseteq \tau_s$. Notice that a class may be defined by a single leaf node feature, a single non-leaf node feature or a combination of features. For example, $AAAI \rightarrow S_1$, $AI \rightarrow S_1$, $AI\&(1990-1999) \rightarrow S_1$ and $(1980-1989) \rightarrow S_1 \wedge S_2$ are all possible association rules.

Rule support and confidence are two measures of a rule's significance. The support of the class C (denoted by $P(C)$) refers to the percentage of tuples in the global relation that belong to the class C. 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 $sourceSet$ $\widehat{S}$.

As we discussed in Section 1, it may be prohibitively expensive to learn and store the coverage and overlap statistics for every possible class. In order to keep the number of association rules low, we prune classes and source sets in the following way:

- **Discovering large classes:** We use a threshold on the support of the classes to discover large classes (any class with support higher than a given threshold) and prune small classes. In this paper we present an algorithm to efficiently discover the large classes by using the *anti-monotone property*[3]([HK00]).

- **Discovering strongly correlated source sets:** In order to remember small number of overlap statistics, we just store overlap statistics for strongly correlated source sets. For the uncorrelated source sets, we assume that the sources follow the independence assumption(FKL97])[4]. In the paper, we discuss how to use the *Apriori* algorithm([AS94]) to discover strongly correlated source sets for all the large classes.

After discovering the large classes and strongly correlated source sets, we can compute the coverage and overlap statistics in the following way:

- For each large class C and each 1-$sourceSet$ $\widehat{S}$, we generate a rule $C \rightarrow \widehat{S}$. The confidence of the rule, $P(\widehat{S}|C)$, denotes the coverage of the single source in $\widehat{S}$ for class C;

- For each large class C and each strongly correlated k-$sourceSet$ $\widehat{S}$ (where $k > 1$), we generate a rule $C \rightarrow \widehat{S}$. The confidence of the rule, $P(\widehat{S}|C)$, denotes the overlap among the sources in $\widehat{S}$ for class C. For example, $AI \rightarrow S_1 \wedge S_2$ with confidence = 40% means that sources S_1 and S_2 have 40% overlap AI papers.

2.2.3 Mapping users' queries to abstract classes

In order to use the learned coverage and overlap statistics of the large classes, we need to map a user's query to a discovered large class. Then the coverage and overlap statistics for the corresponding class can be used to predict the coverage of the sources and overlap among the sources for the query.

The mapping can be done according to the following algorithm.

1. If the classificatory attributes are bound in the query, then find the lowest ancestor abstraction class with statistics[5] for the features of the query;

2. If no classificatory attribute is bound in the query, then we should do one of the following,

   - Check whether we have learned some association rules between the non-classificatory features in the query with classificatory features[6]. If we did, use these features as features of the query to get statistics, go to step 1;

[3]If a set cannot pass a test, all of its supersets will fail the same test as well.

[4]If source S_1 and S_2 are independent, then the probability that a tuple is present in source S_1 is independent of the probability that the same tuple is present in S_2. Thus $P(S_1 \wedge S_2) = P(S_1) \times P(S_2)$.

[5]If we have multiple ancestor classes, the lowest ancestor class with statistics means the ancestor class with lowest support counts among all the discovered large classes.

[6]In order to simplify the problem, we did not discuss this kind of association rule mining in this paper, but it is just a typical association rule mining problem. A simple example would be to learn the rules like:$J.Ullman \rightarrow Databases$ with high enough confidence and support.

- Present the discovered classes to the user, and take the user's feedback to select a class;
- Use the root of the hierarchy as the class of the query.

2.2.4 *Ranking sources for a class*

In this section we discuss how we rank the sources for the mapped class C using the statistics we learned. At first we select the source with the highest coverage[7] as the first source, then we use the overlap statistics to compute the residual coverages of the rest of the sources to find the second best, and so on, until we get a plan with the utility we want. However as we discussed earlier, we only keep overlap statistics for highly correlated source sets. For source sets without overlap statistics we use the independence assumption to estimate their overlap information.

Example: Suppose S_1,S_2 andS_3 are sources exporting tuples for class C. Let $P(S_1|C)$, $P(S_2|C)$ and $P(S_3|C)$ be the learned coverage statistics, and $P(S_1 \wedge S_2|C)$ and $P(S_2 \wedge S_3|C)$ be the learned overlap statistics. With the independence assumption, it's easy to estimate the overlap of the sources in the 2-$sourceSet$ (S_1, S_3)

$$P(S_1 \wedge S_3|C) = P(S_1|C) \times P(S_3|C)$$

But computing $P(S_1 \wedge S_2 \wedge S_3|C)$ becomes non-trivial, since it contains both independent and highly correlated subsets. In this case we begin by choosing the 2-sourceSet with maximal overlap among (S_1, S_2), (S_2, S_3) and (S_1, S_3). Let (S_2, S_3) be the maximally overlapping 2-$sourceSet$. Then assuming S_1 is independent of $S_2 \wedge S_3$, we compute

$$P(S_1 \wedge S_2 \wedge S_3|C) = P(S_1|C) \times P(S_2 \wedge S_3|C).$$

Similarly, the overlap for k sources can be estimated as the product of the overlap of the maximally overlapping $(k-1)$ sources and the coverage of the remaining source.

3. GENERATING DATASET BY PROBING

In order to use association rule mining approach to learn the coverage and overlap statistics, we have to collect the input dataset for mining. However in a data integration scenario, we can not get all the data from the sources directly because of their autonomous nature. So the only way we can extract features from the autonomous sources is to probe the sources. In this section we discuss how to generate probing queries and store the probing results.

3.1 Probing queries

Once the design of global schema and AV hierarchies of a data integration system is done, a set of probing queries have to be generated. The probing queries can be generated by just including all the features of the leaf nodes of a single AV hierarchy. Note that even if multiple classificatory attributes' features can be served as probing queries, we still just need one classificatory attribute's leaf node features as the probing queries. This is because querying all the sources by binding all the leaf node features of a classificatory attribute will give you the whole relation or a representative subset of the relation. For example, in our motivating example, if all the sources can be queried by giving a conference name, then the probing queries are just selection queries on the conference names in the leaf nodes of the AV hierarchy.

3.2 Datasets

[7]We can also rank the sources based on the combined utility with other quality parameters such as response time, freshness of the data, etc. For a detailed discussion on ranking sources see [FKL97],[NLF99] and [NK01].

CID	Conference	Year	Count
1	ICDE	2001	79
2	ICDE	2000	67
3	ICDE	1999	70

Table 1: ***Tuples in the table classInfo***

CID	Source	Count
1	(S_2, S_7)	79
2	(S_1, S_2, S_3)	38
2	(S_1, S_2)	20
2	S_3	9
3	(S_2, S_3, S_4)	63
3	(S_1, S_2, S_3, S_4)	7

Table 2: ***Tuples in the table sourceInfo***

After we get the list of probing queries, we can query all the sources using queries from the query list. Once we get all the answers back from the sources, we union the results by deleting overlap tuples, and keep the results in the result dataset. This dataset will be used as input for our association rule mining algorithm. Specifically the result dataset consists of two tables, **classInfo**(CID, A_{c_1},...,A_{c_n}, Count) and **sourceInfo(CID, Source, Count)**, where A_{c_j} refers to the j^{th} classificatory attribute. The leaf classes with at least one tuple in the sources are given a class identifier, CID. The total number of distinct tuples for each leaf class are entered into **classInfo**, and a separate table **sourceInfo** keeps track of which tuples come from which sources. If multiple sources have the same tuples in a leaf class then we just need to remember the total number of common tuples for that overlapped source set. In the worst case, we have to keep the counts for all the possible subsets for each class(2^n of them, where n is the number of sources)[8].

Example: Continuing the example in Section 1, we shall assume the following query is the first probing query:

Q(title, author, conference, year) :−
  paper(title, author, conference, year),
  conference="ICDE".

Then we can update these tuples into the dataset: **classInfo**(see Table 1) and **sourceInfo** (see Table 2). In the table **classInfo**, we use attribute CID to keep the id of the class, attributes Conference and Year to keep the classificatory attribute values, and attribute Count to keep the total number of distinct tuples of the class. In the table **sourceInfo**, we use attribute CID to keep the id of the class, attribute Source to keep the overlap sources in the class, and attribute Count to keep the number of overlapped tuples of the sources. For example, in the leaf class with class CID=2, we have three subsets of overlapped sources which disjointly export the total 67 tuples. As we can see, all the sources in the set (S_1, S_2, S_3) export 38 tuples in common, all the sources in the set (S_1, S_2) export another 20 tuples in common, and the single source S_3 itself export another 9 tuples.

[8]Although in practice the worst case is not likely to happen, if the results are too many to remember, we can do one of the following: use a single scan mining algorithm(see Section 4.1.2), then we can count query by query during probing, in this way we just need to remember the results for the current query; just remember the counts for the higher level abstract classes; or just remember overlap counts for upto k-$sourceSet$s, where k is a predefined value($k < n$).

Algorithm 1 LCS algorithm

```
input: the AV hierarchies; dataset: classInfo,sourceInfo; min_sup: minimum support threshold;
output: ruleSet: the learned rules, classSet: discovered large classes;
begin
$classSet = \{\}, ruleSet = \{\}$;
for ($k = 1; k <= n; k++$) do
  $classSet_k = \{\}$;
  for (each leaf class $lc \in dataset$) do
    $C_{lc} = genClassSet(k, lc, ...)$;
    for (each class $c \in C_{lc}$) do
      IF($c \notin classSet_k$)
      THEN $classSet_k = classSet_k \cup \{c\}$;
      $c.count = c.count + lc.Count$;
      for (each source $S \in t.sources$ ) do
        IF(rule $r_{c \to s} \notin ruleSet$)
        THEN $ruleSet = ruleSet \cup \{r_{c \to s}\}$;
        $lc.Count_S \leftarrow$ total number of tuples in Source $S$ and Class $lc$;
        $r_{c \to s}.count = r_{c \to s}.count + lc.Count_S$;
      end for
    end for
  end for
  $classSet_k = \{c \in classSet_k | c.count >= min\_sup\}$;
  remove rules of corresponding low support classes from $ruleSet$;
  $classSet = classSet \cup classSet_k$;
end for
for (each rule $r_{c \to s} \in ruleSet$) do
  $r_{c \to s}.confidence = \frac{r_{c \to s}.count}{c.count}$
end for
return $ruleSet$;
end

Procedure genClassSet(k: number of features; lc: the leaf class; the AV hierarchies; classSet; classSet_k)
for (each feature $f_i \in lc$) do
  $ftSet_i = \{f_i\}$;
  $ftSet_i = ftSet_i \cup \{ancestor(f_i)\}$; {//root of the hierarchy is not included in ancestor of $f_i$}
end for
$candidateSet = \bigcup_j ftSet_{j1} \times ftSet_{j2} \times ... \times ftSet_{jk}$;{//Using cartesian product to generate all $k$ feature classes.}
{//This pruning procedure can be implemented inside the cartesian product procedure}
for (each class $c \in candidateSet$ but $c \notin classSet_k$ ) do
  if (any class c' (with $k - 1$ subset features of class c) $\notin classSet_{k-1}$) then
    delete c from $candidateSet$;
  end if
end for
return $candidateSet$;
```

4. ALGORITHMS FOR LEARNING COVERAGE AND OVERLAP

As we discussed earlier, we use association rules to learn the coverage and overlap statistics. In this section, we introduce an algorithm, LCS, to efficiently discover large classes, generate association rules between these classes and sources, and compute the confidence of the rules using the input dataset. We also show how the Apriori algorithm can be applied to learn the overlap statistics.

4.1 The LCS Algorithm

The LCS algorithm (see Algorithm 1) requires the dataset: $classInfo$ and $sourceInfo$, the AV hierarchies, and the minimum support as inputs, and dynamically discovers the large classes inside a mediator system. As mentioned earlier, in order to avoid too many small classes, we can set support count thresholds to prune the classes with support count below the threshold. We use a uniform minimum support for all the classes. We use the anti-monotone property (which means that if a set cannot pass a test, all of its supersets will fail the same test as well) to improve the efficiency of the algorithm.

As we can see, the LCS algorithm makes multiple passes over the data. Specifically, we first find all the large classes with just one feature, then we find all the large classes with two features using the previous results and the anti-monotone property to efficiently prune classes before we start counting, and so on. We continue until we get all the large classes with all the n features. For each tuple in the k-th pass, we find the set of k feature classes it falls in, increase the count $support(C)$ for each class C in the set, and increase the count $support(C \cap S)$ for each source S with this tuple. We prune the classes with total support count less than the minimum support count. After identifying the large classes, we can easily compute the coverage of each source S for every large class C as follows:

$$confidence(C \to S) = \frac{support(C \cap S)}{support(C)}$$

4.1.1 The "genClassSet" function

In the algorithm, we find all the candidate classes with k features for a leaf class $lc = \{CID, A_{c_1}, ..., A_{c_n}, Count\}$ by a procedure **genClassSet**. The procedure prunes small classes using the large class set $classSet_{k-1}$ found in the $(k-1)$th pass. We explain the procedure using the following example.

Example: Assume we have a leaf class lc={1, ICDE, 2000, 67} and k=2. We first extract the feature values $\{A_{c_1} = ICDE, A_{c_2} = 2000\}$ from the leaf class. Then for each feature, we generate a feature set which includes all the ancestors of the feature. Then we will get two feature sets: $ftSet_1 = \{ICDE, DB\}$ and $ftSet_2 = \{2000, (00-01)\}$. Suppose the class with the single feature "ICDE" is not a large class in the previous results, then any class with the feature "ICDE" can not be a large class according to the anti-monotone property[9]. We can prune the feature "ICDE" from $ftSet_1$, then we get the candidate class set for the leaf class c,

$$\begin{aligned} candidateSet &= ftSet_1 \times ftSet_2 \\ &= \{DB\&(00-01), DB\&2000\}. \end{aligned}$$

4.1.2 Single versus Multiple Scans of Data

In the LCS algorithm, we assume that the number of classes will be high. In order to avoid considering a large number of classes, we prune classes during counting. By doing so, we have to scan the dataset multiple times. However if the number of classes are small and the cost of scanning the whole dataset is very expensive, then we have to use a one pass algorithm. For each leaf class lc of every probing query's results, the algorithm has to generate an candidate class set of lc, increase the counts of each class in the set. By doing so, we have to remember the counts for all the possible classes during the counting, but we don't need to remember all the probing query results.

4.2 Learning Overlap among Sources

Once we discover large classes in the mediator, we can learn the overlap between sources for each large class. Here we also use the dataset: **classInfo** and **sourceInfo**. In this section we discuss how to learn the overlap information between sources for a given class.

[9]In order to improve the efficiency of the algorithm, we can prune the small classes during the cartesian product procedure.

From the table **classInfo** we can classify the leaf classes into the large classes we learned in the previous section. Here a leaf class can be classified into multiple classes. For example, a leaf class about a paper in Conference:"AAAI", and Year:"1999", can be classified into the following classes: AAAI, AI, 1999, 90-99, AAAI&1999, AAAI&90-99, AI&1999, AI&90-99, if all of these classes are large classes in the mediator.

After we classify the leaf classes in **classInfo**, for each discovered large class C, we can get its descendent leaf classes, which can be used to generate a new table $sourceInfo_c$ by selecting relative tuples for its descendent leaf classes from **sourceInfo**.

Next we apply the Apriori algorithm to find strongly correlated source sets. The candidate source sets will include all the combinations of the sources, with 1-$sourceSets$, 2-$sourceSets$,...,n-$sourceSets$, where n is the total number of sources. In order to use Apriori, we have to decide a minimum support threshold, which will be used to prune uncorrelated source sets.

Once the frequent source sets from the table $sourceInfo_c$ have been found, it is straightforward to calculate the overlap statistics for these combination of strongly correlated sources[10]. We can compute the overlap probability of these correlated sources $S_1, S_2, ..., S_k$ in class C by using the following formula:

$P((S_1 \wedge S_2 \wedge ... \wedge S_k)|C) = \frac{support\_count(S_1 \cap S_2 \cap ... \cap S_k)}{support\_count(C)}$

Here the $support\_count(C)$ is just the total number of tuples in the table $sourceInfo_c$.

5. EMPIRICAL EVALUATION

In this section we present results of experiments conducted to study the variation in pruning power and accuracy of our algorithms for different class size thresholds. In particular, given a set of sources and probing queries, our aim is to show that with increase in threshold value of large classes, the ***time*** (to identify large classes) and ***space*** (number of large classes remembered) usage decreases but with a reduction in ***accuracy*** of the learnt estimates.

We have implemented a prototype statistics learning system using the algorithms described in the paper. Currently our system only implements LCS algorithm to learn the coverage information of sources. We also designed a data integration system that mimics the simple mediator example provided earlier (see Section 1) and uses the source coverage statistics learnt by our statistics learner. Both the systems are written in Java 2. All the experiments presented here were conducted on a Linux system with 256MB main memory running under Red Hat Linux 6.1. Accordingly we generate a set of 15 data sources exporting the mediator relation **paper**(title,author,conference,year). The sources contain publications of 15 leading research groups in Artificial Intelligence and Database research. We assume no binding restrictions for the sources and hence implement them as tables in a Cloudscape database([C36]). We perform our experiments using sources with sizes ranging from 500 to 1000 tuples. Conference and Year are the *classificatory* attributes in our mediator relation.

Since our learning approach is highly dependent on the AV hierarchy, we assume most system designers will want to start with a simple hierarchy that reflects all the levels of abstraction but contains small number of features at each level. Based on the efficiency of the statistics learnt, they may add additional nodes to the hierarchy. To see how good our algorithm would work in such a scenario, we designed two sets of attribute value hierarchies for our classificatory attributes, called **Large Hierarchies** and **Small Hierarchies** based on Figure 1. Both hierarchies contain three levels of abstraction from leaf to the root but differ in the number of nodes at each level as shown in Figure 5 and Figure 6.

[10]There is only a small variation: we need to add the actual number of counts for each tuple in stead of just add one.

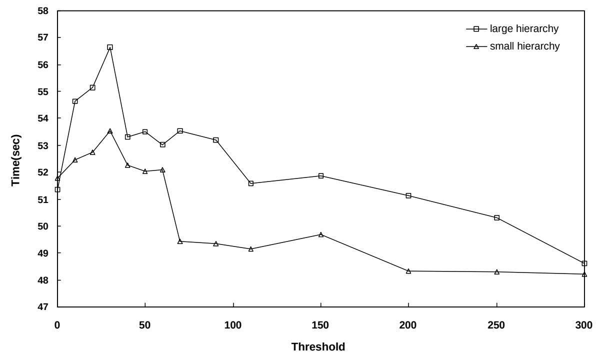

Figure 2: ***LCS learning time for various thresholds***

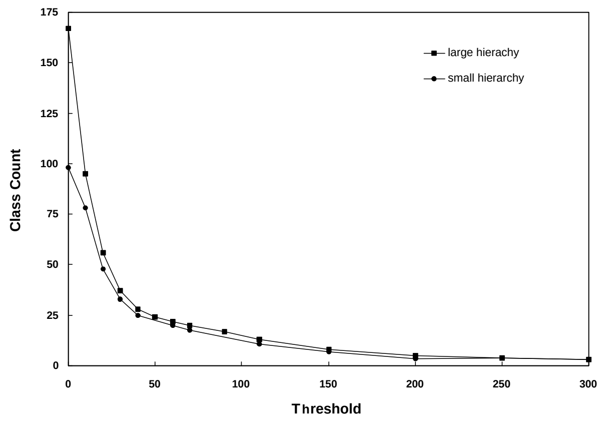

Figure 3: ***Pruning of classes by LCS***

5.1 Time and Space Usage

To learn the coverage statistics for the data sources to be used by the mediator, we used the leaf node features from the AV hierarchies of the classificatory attributes as probing queries. To evaluate the performance of our statistics learner, we varied the threshold value for a large class and measured the number of large classes and the amount of time used for learning source coverage statistics for

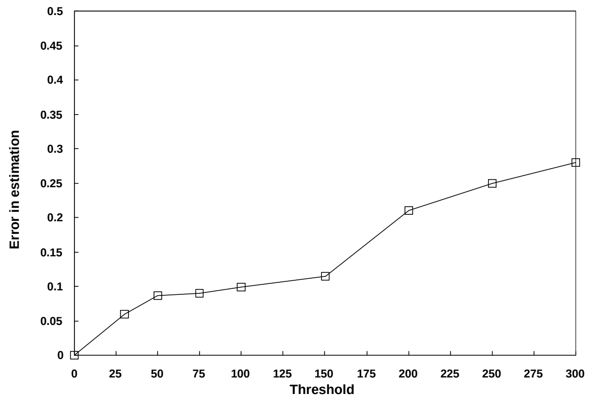

Figure 4: ***Error induced in Coverage Estimation***

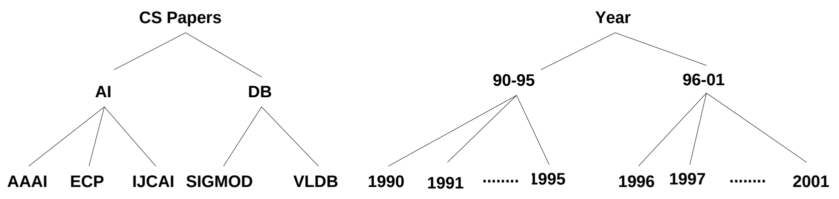

Figure 5: Small Hierarchies

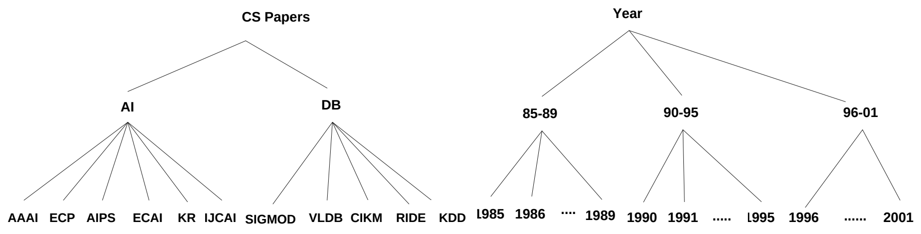

Figure 6: Large Hierarchies

the large classes. We assume that the results of the probing queries are materialized before LCS starts learning rules. We do acknowledge the fact that actual learning time should involve the time to probe and generate results, and given the high latency involved with Internet sources, the probing time will vary based on the number of probing queries, number of sources queried and time of querying. Since variations in threshold values do not necessitate re-probing of the sources, we chose to ignore the probing time. Figure 2 compares the time taken by LCS to learn rules for different threshold values. For a given threshold we note the average time taken by LCS to generate the rules for different sets of probing queries. Figure 3 compares the number of pruned classes with increase in threshold values. As can be seen from Figure 2, for lower thresholds LCS takes more time to learn the rules. As expected, for lower values of support threshold, LCS will prune less number of classes and hence will end up learning more number of rules for the classSet of the mediator. This in turn explains the increase in learning time for lower threshold values. But a contradiction is seen for support threshold value of 0. Here LCS takes less time but learns more rules than say for threshold value 75. This can be explained by noting that for threshold value of 0, no calls to a pruning routine are necessary, while for higher than zero we do have to test and prune each abstract class generated by cartesian product of the features of the AV hierarchies. For threshold values above 75, some of the leaf node features get pruned, which leads to lesser number of abstract classes being generated by way of cartesian products and hence reduces the time for pruning. This leads to a reduction in the overall learning time of LCS for thresholds above 75. Compared to Figure 2, Figure 3 holds no surprises. As is intuitive with increase in threshold value, the number of small classes pruned increases and hence we see a reduction in the number of large classes. For any threshold value greater than the support of the largest abstract class in the classSet, LCS returns only the root as the class to remember. In Figure 3, we get one large class for threshold value 300 and 500 for Small and Large hierarchy respectively. Figures 2 and 3 show LCS performing uniformly for both Small and Large hierarchy. For both hierarchies, LCS generates large number of classes for small threshold values and requires more learning time. For the case of zero threshold value, LCS shows the contradiction explained earlier.

5.2 Accuracy of Estimated Statistics

To calculate the error in our coverage estimates, we make use of the prototype data integration system and a subset of our probing queries as testing queries. Given a query, the integration engine maps it to the lowest abstract class for which coverage statistics have been learnt. The engine then issues the query to the sources in descending order of their coverages for the mapped class. Suppose the testing query is $Conference = SIGMOD$, and the statistics are available for class DB while class $SIGMOD$ was pruned by LCS. From Figures 5 and 6 one can see that DB is the next lowest abstract class for the query and hence the coverage statistics for DB would be used to identify the sources to call to answer the given testing query. The tuples returned by sources are then materialized and coverage statistics are again learnt for each source for the mapped class. We call the newly learnt statistics as the *real coverage* of the sources and use the same to calculate the accuracy of the estimated coverages for the class. Suppose the real coverage of sources S_1 to S_n for a query Q_i is $cov'_{i_1}, cov'_{i_2}, ..., cov'_{i_n}$ (n is the number of sources) for the query. Considering $cov_{i_1}, cov_{i_2}, ..., cov_{i_n}$ as the learned coverages for the class to which Q_i is mapped, we compute the mean absolute error as $EC_i = \frac{\sum_{j=1}^{n}(|cov'_{i_j} - cov_{i_j}|)}{n}$. Given a threshold, the error in estimation of the learner is the average value of the mean absolute error for the test queries. Since we are really interested in relative ordering of sources given a query, this method of calculating the accuracy of estimates imposes a tighter control[11] than is required.

From Figure 4, we can see that the error in estimation increases with increase in support threshold. This is intuitive, given that with increase in threshold values, the number of classes pruned increase and so a query will be mapped to a high level abstract class instead of the leaf node class to which it actually belongs. The estimation error will have a maximum value of one, and as can be seen from

[11] In fact even though the ranking of sources with real and estimated coverage may be same, the absolute error might be high. On the contrary, for low absolute error one may see a huge difference in relative ranking.

the graph in Figure 4, LCS degrades gracefully.

Altogether the experiments show that our LCS algorithm uses the association mining based approach effectively to control the number of statistics required for data integration. For our sample mediator, an ideal threshold for LCS would be around 75, where LCS effectively prunes a large number of small classes and yet does not have high estimation errors.

6. RELATED WORK

The utility of quantitative coverage statistics in ranking the sources is first explored by Florescu et. al. [FKL97]. The primary aim of their work however was on the "use" of coverage statistics, and they do not discuss how such coverage statistics could be learned. In contrast, our main aim in this paper is to provide a framework for learning the required statistics. We do share their goal of keeping the set of statistics compact. Florescu et. al. achieve the compactness by assuming that each source is identified with a single primary class of queries that it exports. They "factorize" the coverage of a source with respect to an arbitrary class in terms of (a) the coverage of that source with respect to its primary class and (b) the statistics about inter-class overlap. In contrast, we consider and learn statistics about a source's coverage with respect to any arbitrary query class. We achieve compactness by dynamically identifying "big" query classes, and keeping coverage statistics only with respect to these classes. From a learning point of view, we believe that our approach makes better sense since inter-class overlap statistics cannot be learned directly[12].

The work by Gruser et. al. [GRZ$^+$00] considers mining response time statistics for sources in an information gathering scenario. Given that both coverage and response time statistics are important for query optimization, our work can be seen as complementary to theirs. Indeed, in [NK01], we describe a framework that uses both coverage and response time statistics to jointly optimize the cost and coverage of query plans in data integration.

There has been some work on ranking text databases in the context of key word queries submitted to meta-search engines. Recent work ([WMY00], [IGS01]) considers the problem of classifying text databases into a topic hierarchy. While their approach involves estimating the relevance of a database for a given topic, the textual nature of the databases precludes any sophisticated estimation of coverage and overlap.

7. CONCLUSION

In this paper we motivated the need for automatically learning the coverage and overlap statistics of sources for efficient query processing in a data integration scenario. We then presented a set of connected techniques that estimate the coverage and overlap statistics while keeping the needed statistics tightly under control. Our specific contributions include:

- A model for supporting a hierarchical classification of the set of queries.
- An approach for estimating the coverage and overlap statistics using association rule mining techniques.
- A threshold-based modification of the mining techniques for dynamically controlling the resolution of the learned statistics.

We described the details of our approach, and presented preliminary experimental results showing the feasibility of the approach. The mining algorithms presented in this paper are being integrated into a prototype system called HAVASU [13]that we are developing for supporting query processing in data integration. *Havasu* system is intended to support multi-objective query optimization, flexible execution strategies for parallel plans, as well as mining strategies for learning source statistics.

Acknowledgements

We would like to thank Huan Liu, as well as the anonymous referees of the previous version of this paper for their helpful comments and suggestions. This research is supported in part by the NSF young investigator award (NYI) IRI-9457634, and NSF grant IRI-9801676.

References

[ACPS96] S. Adali, K. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proceedings of SIGMOD-96*, 1996.

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

[CGHI94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantino, J.Ullman, J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *IPSJ*, Japan, 1994.

[C36] Cloudscape 3.6, available at http://www.cloudscape.com.

[DL99] A. Doan and A. Levy. Efficiently Ordering Plans for Data Integration. The *IJCAI-99 Workshop on Intelligent Information Integration*, Stockholm, Sweden, 1999.

[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.

[GRZ$^+$00] Jean-Robert Gruser, Louiqa Raschid, Vladimir Zadorozhny, Tao Zhan: Learning Response Time for WebSources Using Query Feedback and Application in Query Optimization. VLDB Journal 9(1): 18-37 (2000)

[HK00] Jiawei Han and Micheline Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmman Publishers, 2000.

[IGS01] P. Ipeirotis, L. Gravano, M. Sahami. Probe, Count, and Classify: Categorizing Hidden Web Dababases. In *Proceedings of SIGMOD-01*, 2001.

[LKG99] E. Lambrecht, S. Kambhampati and S. Gnanaprakasam. Optimizing recursive information gathering plans. In *Proceeding of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[LRO96] A. Levy, A. Rajaraman, J. Ordille. Query Heterogeneous Information Sources Using Source Descriptions. In *VLDB Conference*, 1996.

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

[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.

[PL00] Rachel Pottinger , Alon Y. Levy , A Scalable Algorithm for Answering Queries Using Views Proc. of the Int. Conf. on Very Large Data Bases(VLDB) 2000.

[WMY00] W. Wang, W. Meng, and C. Yu. Concept Hierarchy based text database categorization in a metasearch engine environment. In WISE2000, June 2000.

[12]They would have to be estimated in terms of statistics about the coverages of the corresponding query classes by various sources, as well as the inter-source overlap statistics. In this sense, the statistics in [FKL97] can be thought of as a post-processing factorization of the statistics learned in our framework.

[13]http://rakaposhi.eas.asu.edu/havasu.html