International Journal of Cooperative Information Systems
Vol. 11, Nos. 1 & 2 (2002) 119–144
© World Scientific Publishing Company

SELECTIVELY MATERIALIZING DATA IN MEDIATORS BY ANALYZING USER QUERIES*

NAVEEN ASHISH†

IBM Almaden Research Center
650 Harry Road, San Jose CA 95120-6099
ashish@us.ibm.com

CRAIG KNOBLOCK AND CYRUS SHAHABI

Information Sciences Institute
Integrated Media System Center and Department of Computer Science
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
knoblock@usc.edu
shahabi@usc.edu

There is currently great interest in building information mediators that can integrate information from multiple data sources such as databases or Web sources. The query response time for such mediators is typically quite high, mainly due to the time spent in retrieving data from remote sources. We present an approach for optimizing the performance of information mediators by selectively materializing data. We first present our overall framework for materialization in a mediator environment. The data is materialized *selectively*. We outline the factors that are considered in selecting data to materialize. We present an algorithm for identifying classes of data to materialize by analyzing one of the factors which is the distribution of user queries. We present results with an implemented version of our optimization system for the Ariadne information mediator, which show the effectiveness of our algorithm in extracting patterns of frequently accessed classes from user queries. We also demonstrate the effectiveness of approach in optimizing mediator performance by materializing such classes.

1. Introduction

Several "information mediator" systems have been built to provide integrated and structured query access to multiple information sources. The representative systems include TSIMMIS,[10] Information Manifold,[11] The Internet Softbot,[6] InfoSleuth,[12]

*The research reported here was supported in part by the Integrated Media Systems Center, an NSF Engineering Research Center, in part by the United States Air Force under contract number F49620-98-1-0046, in part by the Jet Propulsion Laboratory (JPL) of the National Aeronautics and Space Administration, NASA, under contract number 961518, and in part by research grants from NCR and General Dynamics Information Systems. The views and conclusions contained in this article are the authors' and should not be interpreted as representing the official opinion or policy of any of the above organizations or any person connected with them.
†Work done when the author was a graduate student at the University of Southern California.

Infomaster,[9] DISCO,[13] HERMES,[14] SIMS[15] and Ariadne.[16] Most of the systems are based on a mediator-wrapper[7] architecture, where database like querying of semi-structured Web sources through *wrappers* around pre-specified Web sources and integrated access to multiple sources is provided by a mediator.

The query response time for information mediators, particularly Web based mediators is often very high, mainly because the speed of the resulting mediator application is heavily dependent on the Web sources being accessed. A large amount of time is spent in retrieving data from the Web source when answering a user query. We presented an approach to optimizing the performance of information mediators by locally materializing data in Ref. 17.

To answer most queries a large number of Web pages must be fetched over the network. For instance, consider a mediator that provides integrated access to Web sources of information about countries in the world. For all mediator applications the set of Web sources from which the mediator will extract and integrate information is prespecified and fixed. For the countries mediator, the set of sources is:

- The CIA World Factbook[a] which provides interesting information about the geography, people, government, economy etc. of each country in the world.
- The NATO homepage[b] from which we can get a list of NATO member countries.
- The InfoNation[c] source which provides statistical data about UN member countries.

Without any kind of optimization i.e. assuming that all data must be fetched from the Web sources in real time, a typical query to this mediator such as *"Find the defense expenditure and spending on education of all countries that have a national product greater than 500 billion dollars"* can take several minutes to return an answer. This is because for this particular query the mediator must retrieve the pages of all countries in the CIA World Factbook to determine which ones have a national product greater than $500 billion, which takes a large amount of time. The query response time can be greatly improved if frequently accessed data is materialized at the mediator side.

We present a performance optimization approach for information mediators based on selectively materializing data. Our work is different from Web caching systems[18] where caching or materialization is done at the level of individual Web pages. Our work is aimed at optimizing mediators that extract and integrate information from pre-specified semi-structured Web sources rather than retrieving entire Web pages from arbitrary sites. In our approach there are two primary issues that must be addressed. First we must design the overall framework for materialization i.e. how do we represent and use the materialized data. Then there is the issue of

[a]http://www.odci.gov/cia/publications/factbook/country.html
[b]http://www.nato.int/family/countries.htm
[c]http://www.un.org/Pubs/CyberSchoolBus/infonation/e_infonation.htm

what data should be materialized. We describe our overall approach in Sec. 2. In Sec. 3, we describe an approach for selecting data to materialize by analyzing the distribution of user queries. In Sec. 4, we present experimental results demonstrating the effectiveness of our approach. Section 5 contains a complexity analysis of this algorithm. In Sec. 6, we describe related work followed by a conclusion and directions for future work in Sec. 7.

2. Overall Approach

We now present a description of our approach to performance optimization by materialization. We first provide a brief overview of the SIMS information mediator, in fact the SIMS architecture is typical of many of the other information mediator systems we mentioned. The Ariadne architecture also borrows heavily from that of SIMS. SIMS is used to integrate information from primarily database systems whereas Ariadne integrates information from semi-structured Web sources. We then provide a description of our framework for materializing data in mediators. Finally, we outline the factors in selecting data to materialize and the portion focused on in this paper.

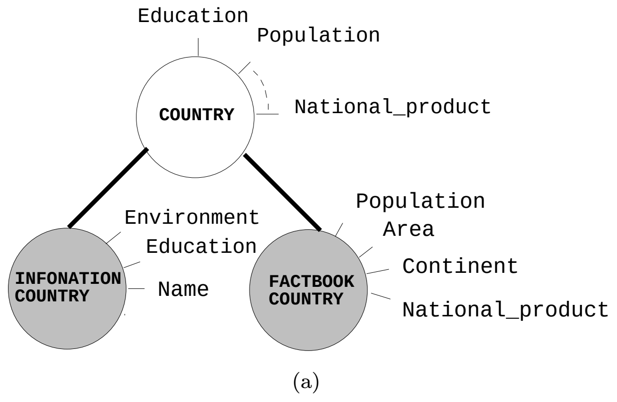
2.1. *SIMS architecture*

In the SIMS system we use the LOOM[1] knowledge representation language (we can also view this as a data modeling language) for modeling data. The user is presented with an integrated view of the information in several different sources, which is known as the *domain model*. We describe the contents of the individual information sources in terms of the domain model. A simple example is shown in Fig. 1(a). The white circle labeled COUNTRY represents a domain *concept* (equivalent of a class in an object-oriented model) and the shaded circles represent sources. The small lines on the circles represent attributes of the concepts. In this example, the domain concept COUNTRY provides an integrated view over two sources of information about countries — FACTBOOK-COUNTRY and INFONATION-COUNTRY. The user queries the integrated view i.e. concepts in the domain model and the query planner in the mediator generates plans to retrieve the requested information from one or more sources. Please refer to Ref. 15 for a more detailed description of SIMS.

2.2. *Materializing data in mediators*

Our approach to optimization is based on an idea[19] where we identify useful classes of information to materialize, materialize the data in these classes in a database local to the mediator and define these classes as auxiliary information sources that the mediator can access. For instance, in the countries application[d] suppose we

[d]The model showing the attributes of the COUNTRY concept in the countries application is given in the appendix. We will be using this model in examples throughout the paper.



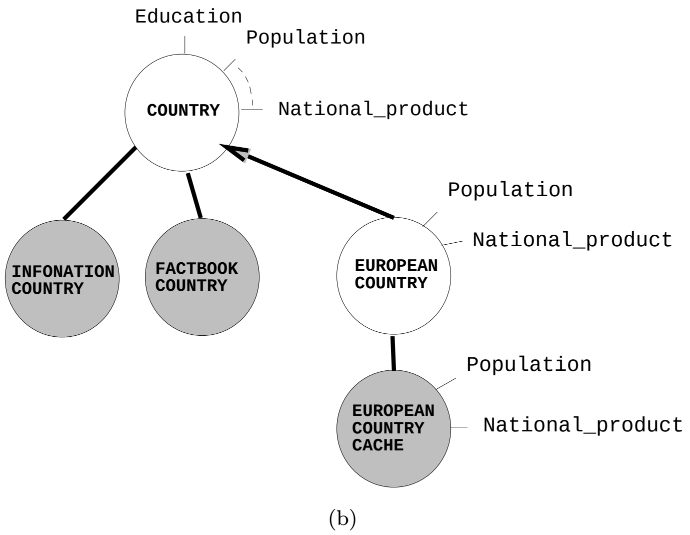


Fig. 1. Information modeling in SIMS.

determined that the class of information — *the population and national product of all European countries* was frequently queried and thus useful to materialize. We materialize this data and define it as an additional information source as shown in Fig. 1(b). Given a query the mediator prefers to use the materialized data instead of the original Web source(s) to answer the query.

Defining the materialized data as another information source has two advantages. First, we can provide a semantic description of the contents of the materialized data (in LOOM). Second, the query planner in the mediator considers the materialized data source when generating plans to retrieve data to answer a user query. The SIMS query planner is designed to generate high quality plans and will generate plans that attempt to use the materialized data sources whenever they reduce the cost of processing a query. We use the materialized data in essentially the same manner as in a *semantic* caching[20] system. Our approach of defining the materialized data as another information source allows us to use the information mediator's knowledge representation system and query planner to address two important problems that arise in any semantic caching system, namely providing a description of the materialized or cached data and doing containment checking i.e. determining if all or some portion of data in a query has already been cached.

2.3. *Selecting data to materialize*

The key problem that remains to be addressed is how to determine what classes of information are useful to materialize. Note that we wish to only selectively materialize data that is useful. The brute force approach of materializing all the data in all the Web sources being integrated is impractical for the following reasons:

- The sheer amount of space needed to store all the data could be very large.
- Data gets updated at the original Web sources and the maintenance cost of keeping the materialized data consistent could be very high.

It is clear that data must be materialized *selectively*. This leaves us with the question of how do we automatically identify the portion of data that is most useful to materialize. We propose an approach where we consider several factors in combination to identify data to materialize. The factors considered are:

- User Query Distribution: By analyzing the user query distribution we can identify the classes of data that are queried most frequently by users and consider materializing such classes. In addition to the materialization framework, an algorithm to identify such frequently accessed classes is the other major contribution of this paper.
- Application and Source Structure: Since we are gathering and integrating data from Web sources that were not designed to support database like querying, certain kinds of queries can be very expensive. For many such queries we can materialize data that will improve response time for those queries. Consider again the country mediator. Assume we have an interface that allows us to ask queries with a selection condition on the GDP. Now such a query is expensive as the source for this information — the CIA World Factbook source does not support selections by GDP and we have to retrieve pages of all the 270 countries in the CIA World Factbook to determine which ones have GDP $>$ \$500 billion. We can prefetch and materialize the primary key (country name) and the selection attribute (GDP) which will improve the response time for selection queries on GDP as the selection can now be done locally. We are developing an approach where we first identify the kinds of queries that can ever be asked of the domain classes in a particular mediator application. In many cases the user can query the mediator application only through a user interface that might further restrict the kinds of queries that can be asked of the domain classes. We have developed a language for providing the specification of the user interface so that it may be used in determining the kinds of queries that can be asked. We then estimate the

costs of the various classes of queries using a cost estimator which is part of the mediator. The purpose is to identify in advance the expensive kinds of queries that could be asked in a particular mediator application. Then using heuristics based on the kind of query, source or sources used to answer the query and also the different data processing operations that are performed to answer the query we prefetch and materialize data that can improve the response time for the expensive queries.

- Update Characteristics and Frequency: We integrate data from Web sources that may get updated. The materialized data must be kept consistent with the original sources and also the maintenance cost for the materialized data must be taken into account. We have developed an approach to automatically estimating the maintenance cost for each proposed materialized class of data from specifications about the update characteristics of the sources and the user's requirements for freshness of data. The maintenance cost is also considered when deciding what classes of data to materialize. There is also the problem of knowing when data in a source has changed. For certain sources we may know exactly at what time and with what frequency the source is updated. For other sources we propose to use techniques for change detection such as those developed in Ref. 21.

In this paper our focus is on the first factor i.e. the analysis of the distribution of user queries to decide what classes of data to materialize. In the following section, we describe how we extract patterns of frequently accessed classes of data from a user query distribution. We present experimental results demonstrating the effectiveness of our approach in extracting patterns from queries. We also present experimental results demonstrating the effectiveness of materializing frequently accessed classes of data in optimizing a mediator application.

3. Analyzing the Distribution of User Queries

One of the hypotheses of our approach is that there will be *patterns* present in user queries i.e. some classes of data would be queried more frequently than others. It would be very useful if we could extract such patterns by analyzing previous user queries as we could consider materializing those. We provide a description of the CM (Cluster and Merge) algorithm which we have developed which identifies useful classes of data to materialize by extracting patterns from user queries. The algorithm takes as input a distribution of user queries and outputs a *compact* description of patterns that it can extract from the query distribution in the form of classes of data. A compact description of frequently accessed classes is necessary from a performance point of view. For each class of data we materialize we define a new information source for the mediator. The general problem of query planning to gather information from many sources is combinatorially hard and having a large number of sources will create performance problems for the query planner.

3.1. *CM algorithm*

The pseudo code for the CM() algorithm is given in Fig. 2. There are three main steps in the algorithm:

- Classifying queries (CLASSIFY_QUERIES()). This is to determine what classes of data the user is interested in.
- Clustering attribute groups (CLUSTER_ATTRIBUTE_GROUPS()). To determine attribute groups of interest for each class.
- Merging classes (MERGE_CLASSES()). This is to try and merge classes of data to make the description more compact.

We now describe the steps in the algorithm in more detail.

3.1.1. *Classifying queries*

We first analyze queries to determine what classes of information users are interested in (procedure CLASSIFY_QUERIES()). For instance queries of the form:

SELECT A
FROM COUNTRY
WHERE region= "Europe"

indicate that the user is interested in the class of European countries. We maintain an ontology in LOOM of classes of information that are queried by users. Initially the ontology contains only the classes in the domain model. We then add sub-classes of these existing classes to the ontology, the sub-classes are generated by analyzing constraints in the user queries. Assuming an SQL syntax for the queries, a query to a domain class has the following general form:

SELECT A
FROM S
WHERE P

where A is the set of attributes queried for the domain class S and $P = P_1$ and P_2 and... P_n are predicates specifying the query constraints (we restrict ourselves to conjunctive queries). We denote as S_P the "query sub-class" which is the sub-class of S satisfying P. We denote as $\{S_P1, S_P2, \ldots, S_Pn\}$ the set of "sub-classes of interest" where the P_i s are the individual predicates comprising P and S_Pi is the sub-class of S satisfying P_i. For instance consider a query such as:

SELECT $population, area$
FROM COUNTRY
WHERE (region = "Europe") AND (government = "Republic");

In the above query the query sub-class is that of European Republic Countries and the sub-classes of interest are European Country and Republic Country. In the

```
CM ($QS$) { /* $QS$ is set of user queries */
      $O$=CLASSIFY_QUERIES($QS$) /* build ontology $O$ of classes
of interest */
      CLUSTER_ATTRIBUTE_GROUPS($O$) /* cluster attribute
groups in each class */
      FOR (all coverings ($S, C$) in $O$) {
          MERGE_CLASSES($S, C$) /* merge classes */
      }
}

CLASSIFY_QUERIES($QS$)
WHILE ( more queries($QS$)) {
   $Q$ = get_next_query($QS$)
   $S_P$ = get_query_subclass($Q$)
   { $S_Pi$ } = get_interest_subclasses($Q$)
/*subclasses based on individual predicates */
   update_ontology($S_P$)
   IF $P = \emptyset$ {
    update_attribute_count($S$,$A$)
   } ELSE {
    n=count({ $S_Pi$ })
    FOR i = 1 to n DO update_attribute_count( $S_Pi, A$ )
    update_ontology( $S_P$ )
}

CLUSTER_ATTRIBUTE_GROUPS($O$) {
/* cluster attribute groups for each class in $O$ with similar frequency */
}

MERGE_CLASSES($S, C$)
$i$=1
$n$=number_of_elements($C$) /* size of set $C$ */
totalsize = 0 /* space occupied by matching groups */
seed = get_seed($Ci$) /* pick an attribute group from Ci */
WHILE (seed $\neq \emptyset$ ) {
   seedsize=get_size(seed) /* space occupied by seed */
   totalseedsize= seedsize*n /* space occupied by (S,seed) */
   FOR i = 1 to n DO {
     temp = find_match(seed, Ci) /* find best matching group for
seed in Ci */
     size = get_size(temp)
     totalsize += size /* total size of matching groups */
   }
   IF (totalsize/totalseedsize) $\geq$ MERGETHRESHOLD {
/* merging criteria satisfied */
    remove(seed,C) /* remove attributes in seed from all classes */
    add_group(S,seed) /* form group in superclass */
   }
   ELSE {
     mark_down(seed,Ci) /* so that the same seed is not chosen again */
   }
i= (i+1)mod(n) /* chose next class in $C$ cyclically */
seed = get_seed($Ci$) /* get seed from next class */
}
```

Fig. 2. The CM algorithm for extracting patterns in queries.

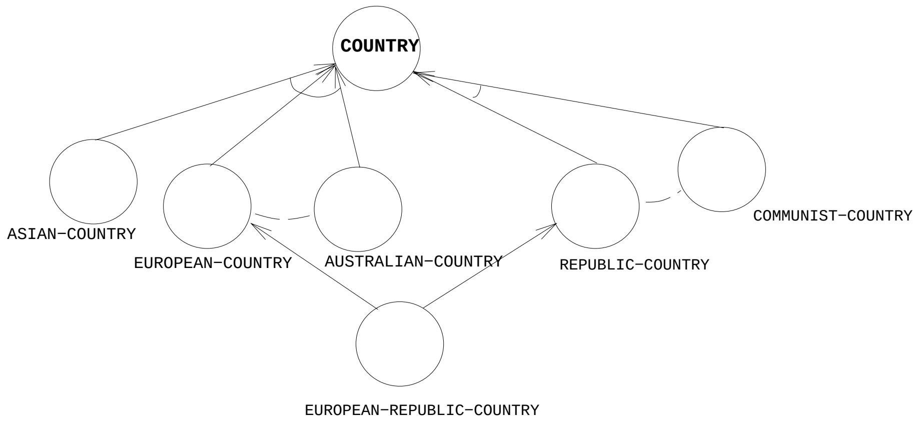


Fig. 3. Ontology of subclasses of COUNTRY.

CLASSIFY_QUERIES() procedure for each query we first determine the query sub-classes and set of sub-classes of interest and insert them into the ontology if they are not already present. For instance for a set of queries on the concept COUNTRY in which the WHERE clauses have constraints on the attributes *region* (bound to a value such as Europe, Asia etc.) or *government* (bound to a value such as Republic, Monarchy, Communist etc.) or both, we would create an ontology such as shown in Fig. 3. (The arcs in the figure represent coverings of groups of subclasses for the superclass COUNTRY).

We also update the query count for the query sub-class S_P for the attribute group A. This is to maintain a record for each sub-class of what attribute groups have been queried and how many times.

3.1.2. *Clustering attribute groups*

After the step of classifying queries we have an ontology of classes of interest and also for each class what attribute groups have been queried and with what frequency. We attempt to merge together attribute groups with similar frequencies in order to reduce the number of groups for each class that we have to consider. This makes the description of classes more compact. Attribute groups are merged together if the relative difference of their frequencies is within a preset limit known as CLUSTER-DIFFERENCE. This is done in the procedure CLUSTER_ATTRIBUTE_GROUPS(). It is a straightforward procedure where we sort the attribute groups by the number of queries and merge together groups with number of queries that differ relatively by no more than CLUSTER-DIFFERENCE.

3.1.3. *Merging classes*

We mentioned earlier that it is important to keep the number of classes of information materialized small from a query processing perspective. Consider the following classes of information, each of which is essentially a group of attributes in a class:

(i) (EUROPEAN-COUNTRY,{population, area})
(ii) (ASIAN-COUNTRY,{population, area})
(iii) (AFRICAN-COUNTRY,{population, area})
(iv) (N.AMERICAN-COUNTRY,{population, area})
(v) (S.AMERICAN-COUNTRY,{population, area}) and
(vi) (AUSTRALIAN-COUNTRY,{population, area}).

We could replace the above six classes by just one class (COUNTRY,{population, area}) which represents exactly the same data. In general thus a group of classes of information of the form $(C_1, A), (C_2, A), \ldots, (C_n, A)$[e] may be replaced by one class i.e. (S, A) if $C_1, C_2, \ldots, C_n$ are direct subclasses of S and form a covering of S. As the ontology of classes is maintained in LOOM, we use LOOM to determine groups of classes we can merge based on class/subclass relationships.

In fact we also allow for a kind of "relaxed" merge where we may merge a set of classes such as $(C_1, A_1), \ldots, (C_n, A_n)$ to (S, A) where the C_is are direct subclasses of S as above. However $A_1, \ldots, A_n$ need not be exactly equal groups rather they just need to overlap, and A is the union of $A_1, \ldots, A_n$. The disadvantage in this case is that the merged class of information will contain some extra data i.e. data not present in any of the classes of information merged to form the merged class. There is a tradeoff between space wasted to store the extra data and the time gained (in query planning) in reducing the number of classes materialized. The amount of space that can be wasted by extra data is limited by a parameter known as the MERGE–THRESHOLD.

The procedure MERGE_CLASSES() shows how we do exact or relaxed merging of classes. The procedure takes as input a superclass S and a set of subclasses C of S that form a covering of S. For each class in C we also have the set of groups of attributes queried. The basic idea is to take an attribute group A in a class C_i in C and see if we can merge with other groups in other classes in C to the group A in the superclass S. We describe the steps in the procedure MERGE_CLASSES() by stepping through the procedure with an example as shown in Table 1. It shows the the various classes in C along with their attribute groups. The asterisk(*) next to the {imports, exports} group of the first class i.e. EUROPEAN–COUNTRY indicates that we will choose that group as a *seed* for merging with other classes. This seed group is chosen randomly. The next step is to find matching groups for the seed in all other classes. This is done by the find_match() procedure which given a seed and a class returns the largest subset of attributes of the seed that it can find in any group in the class. The results of find_match() for each of the classes in C are shown in Table 2. The next step is to find the ratio of the space occupied by the matching groups in the classes of C to the space needed to store the group A for the superclass S. The ratio should be higher than the MERGE–THRESHOLD to allow merging the matching clusters to the cluster A in S . Intuitively this is to ensure

[e] C_is are classes and A is an attribute group.

Table 1. Merging across classes.

set of subclasses C with attribute groups
(EUROPEAN-COUNTRY,{*imports, exports},{area, gdp, economy})
(ASIAN-COUNTRY,{imports, exports, climate},{debt, economy})
(AFRICAN-COUNTRY,{imports}, {population, languages})
(N.AMERICAN-COUNTRY,{climate, terrain},{government},{literacy})
(S.AMERICAN-COUNTRY,{area, coastline},{imports, exports}
(AUSTRALIAN-COUNTRY,{imports, exports, debt},{gdp, defense})

Table 2. Merging across classes (matching classes).

matching groups	size
{imports, exports}	2
{imports, exports}	2
{imports}	1
{}	0
{imports, exports}	2
{imports, exports}	2

Table 3. Classes after one merging step.

set of subclasses C with attribute groups
(EUROPEAN-COUNTRY,{area, gdp, economy})
(ASIAN-COUNTRY,{*climate},{debt, economy})
(AFRICAN-COUNTRY,{population, languages})
(N.AMERICAN-COUNTRY,{climate, terrain},{government},{literacy})
(S.AMERICAN-COUNTRY,{area, coastline}
(AUSTRALIAN-COUNTRY,{debt},{gdp, defense})

that the attributes in A occur sufficiently through the classes in C to justify merging the matching groups to the group A in S. In this example *totalsize* i.e. the space occupied by the matching groups is $(2+2+1+0+2+2) = 9$ units. The *totalseedsize* i.e. the space that should be occupied in case of an exact merge is 2*6= 12. Thus the ratio is $9/12 = 0.75$ and we do merge to the group {imports, exports} for the superclass COUNTRY (assume that MERGE–THRESHOLD is 0.7). Table 3 shows the same set of classes after the merging step when the attributes in A= {imports, exports} have been removed from the classes in C. In case we do not merge the groups we do not remove the attributes from the classes. However we mark the seed group as "down" so that it is not picked again as a seed. We then pick a seed from the next class ASIAN–COUNTRY and repeat the above procedure.

The main motivation behind the steps of merging attribute groups for a single class and also merging classes based on class/subclass relationships is to keep the description of the classes of data extracted as patterns compact. Finally we also keep count of how many queries each class of data supports i.e. how many queries can be answered using that class. From this we can calculate the ratio of supported queries by each class to the total number of queries in the distribution. A class is finally output as a pattern by the CM algorithm only if this ratio is greater than a threshold known as the *query ratio threshold* (q).

3.2. *Language learned by CM*

What CM extracts from queries are essentially classes of data. Such classes if materialized get mapped to (possibly new) concepts in the domain model. We must thus see what exactly is the language of such concepts learned by CM. In CM we analyze the queries that are asked of each individual domain concept. To review what we described above, for each domain concept we may further create subclasses (in the ontology of classes of interest) based on constraints in the query. The subclasses are created based on either equality constraints or numeric constraints. We further merge attribute groups in each class and also merge across class coverings. It is easy to see that CM extracts patterns of the form:
SELECT $A_1, A_2, \ldots A_n$
FROM C

where $A_i s$ are attributes and C is either a domain concept or some subclass of a domain concept (say) where C is the subclass of D such that D satisfies the constraint P. P is a conjunction of predicates that may be equality predicates (numeric or string) or a numeric range predicate.

If required, we could extend CM further to learn more general descriptions than at present. For instance one extension would to be able to have more types of constraints in the queries (and the language learned) such as including negation constraints. Also CM currently analyzes queries on single domain classes (or individual subclasses of domain classes). In case of having a large number of join queries across various domain classes it would be interesting to extract patterns that were joins across 2 or more classes (if present in the distribution). One interesting extension to CM would be to also analyze join queries in the distribution and extract join patterns by generalizing the join queries present.

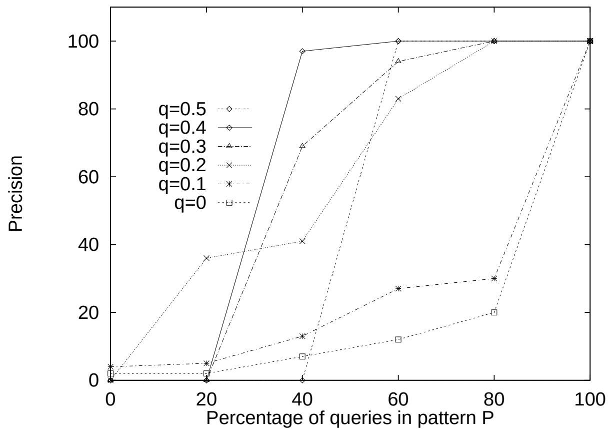
4. Experimental Evaluation

The experimental evaluation consists of two parts. First we evaluate how effective the CM algorithm is in extracting patterns from a query distribution. Next we evaluate the effectiveness of a performance optimization system that we built for Ariadne which materializes data based on just the user query distribution.

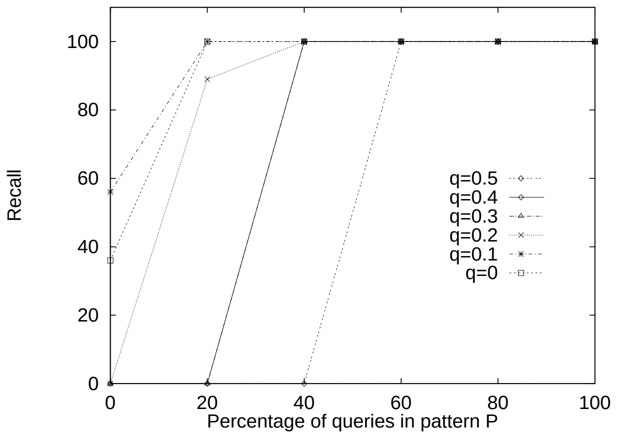
4.1. *Evaluating the CM algorithm*

We set up an experiment to evaluate the effectiveness of the CM algorithm in extracting patterns from a query distribution. The experiment is based on standard precision and recall measurements for evaluating information retrieval systems. This is because we are trying to estimate how effective CM is in extracting patterns that are present and also to what extent it extracts extraneous data as patterns.

We first defined a schema for an imaginary mediator application against which we can pose queries. The schema consists of a class S with 50 attributes $A1, A2, \ldots, A50$. The class S is further partitioned into 5 disjoint subclasses $S1, S2, S3, S4$ and $S5$. Each subclass has 10 instances, $S1$ has instances $E1, E2, \ldots, E10, S2$ has instances $E11, \ldots, E20$ etc. We then defined a "pattern"



(a) Precision



(b) Recall

Fig. 4. Effectiveness of CM Algorithm.

P which is the class $S3$ with attributes $A25, \ldots A30$. We then generated different query distributions against this schema varying the percentage of queries that fall within the pattern. We input each distribution to the CM algorithm to see what patterns it would extract from the distribution. We use standard precision and recall measurements from information retrieval to measure the effectiveness of CM in extracting the predefined pattern P. The precision is the percentage of data extracted that is relevant whereas recall is the percentage of relevant data extracted. In our experiment the predefined pattern P is the *relevant* data, while for each time we run the CM algorithm over a query distribution the patterns extracted from the distribution is the data *retrieved*. Finally since the query ratio threshold (q) affects what patterns are ultimately output by the CM algorithm we present precision and recall measures for varying threshold values.

Figures 4(a) and (b) show the precision and recall values respectively against varying percentages of queries that fall within the predefined pattern P in a query distribution. For each we present precision and recall values for different query ratio thresholds (q). The CM algorithm does indeed prove to be efficient in extracting the predefined pattern P as the recall values are very high (100%) for most of the threshold range for moderate or high percentages of queries in the pattern P. For extraneous data extracted along with P we must analyze the precision values graph. For high threshold values ($q = 0.4$–0.5), the precision is very high when a high percentage of queries are in P ($> 50\%$) but very low for lower percentages. This is because even if queries in the pattern P are present, they need to be in a very high proportion for the CM algorithm to extract them at all. For lower threshold ($q = 0$–0.1), the precision is quite low even when a high percentage of queries is in P. This is because of a low threshold the CM algorithm extracts a lot of random classes as patterns in addition to the pattern P. It is best to keep the threshold at an intermediate value (0.2–0.3) where the precision is high for moderate or high percentage of queries in P. The recall remains quite high (100%) for most of the threshold range for moderate or high percentages of queries in the pattern P.

4.2. *Evaluating the performance improvement*

We have implemented a materialization system for the Ariadne mediator based on the overall approach to materialization described at the beginning of this paper. The focus of this paper is on the analysis of the distribution of user queries. In the previous subsection we experimentally demonstrated the effectiveness of our algorithm in extracting patterns of frequently accessed data. We now present results demonstrating the effectiveness of materializing patterns of frequently accessed data in optimizing the performance of an application.

The experimental hypotheses states that the materialization system is able to successfully extract patterns of frequently accessed data from a user query distribution. Such frequently accessed patterns when materialized will improve the performance of an application. In this set of experiments we present experimental results to validate the second part of this hypothesis. Note that the materialization system also uses the other factors of source structure and update analysis to materialize data. For these experiments, however, we do not do any kind of source structure analysis and also assume that the data does not change at the original sources.

4.2.1. *Admission and replacement*

While we do not describe the details of the design of the materialization system, it is important to briefly describe the admission and replacement policy for classes of materialized data. In the general case, classes of data to materialize are proposed by analyzing both the structure of sources and the distribution of user queries. Indeed it would be beneficial to materialize all such classes for optimizing performance. However, there may be constraints on the total space available for storing the materialized data locally, the total maintenance cost for keeping it consistent or both. To be able to optimally choose a subset of classes (in case all classes cannot be materialized) we rank the classes in order of a *profit* similar to that used in data warehouse cache management.[22] While the general approach uses query cost, hits, space occupied by a class and the maintenance cost for a class to compute this profit, for this set of experiments we simply ranked classes in order of the ratio $Q * H/S$ where Q is the query cost, H is the number of hits to a class and S is the space occupied. Classes of data are materialized locally in this order till all available local space for materialized data is used up.

4.2.2. *Experimental setup*

The materialization system is essentially a complete and separate software system that can be augmented to a mediator (Ariadne in our case) to optimize its performance. It has various modules for tasks such as source structure analysis, query distribution analysis, analyzing updates etc. and also includes a database system as the materialized data store. We will describe the various Ariadne applications used for the experiments when we present the actual results. The modules of the materialization system were implemented primarily in C. For the implementation of the CM algorithm we used Powerloom[3] as the knowledge representation system. The Informix Universal Server[2] was used as the database for storing the materialized data. The materialization system and Ariadne were run on a Sun Ultra 1 running Solaris for obtaining the results. We now describe the applications and present the results.

4.2.3. *Results*

We tested the effectiveness of materializing patterns extracted from user queries using three different Ariadne applications that we have developed. These applications are in different domains and integrate information from a variety of different Web sources. We describe the applications and results below.

Table 4. Effectiveness of extracting patterns in countries mediator.

Query set	Response Time (No optimization)	Response Time (With Materialization)	%improvement
$Q1$	9671 sec	1549 sec	84%
$Q2$	3739 sec	2174 sec	40%

(i) Information about Countries

As described in the Introduction, the information about countries mediator is an Ariadne application integrating information from the following Web sources:

- The CIA World Factbook (http://www.odci.gov/cia/publications/factbook/country.html) which provides interesting information about the geography, people, government, economy etc. of each country in the world.
- The NATO homepage (http://www.nato.int/family/countries.htm) from which we can get a list of NATO member countries.
- The InfoNation (http://www.un.org/Pubs/CyberSchoolBus/infonation/e_infonation.htm) source which provides statistical data about UN member countries.

For each of the applications, we demonstrate effectiveness of materializing patterns by measuring response time against a query set, both with and without any materialization. We use generated query sets for all the applications. We also introduce some distinct query patterns in these query sets. This is because the goal here is to demonstrate that materializing patterns of frequently accessed data (if any) does improve performance. For the countries mediator, we measured total query response time both with and without materializing frequently accessed classes We used 2 query sets $Q1$ (of 200 generated queries) and $Q2$ (of 185 actual user queries). In the query set $Q1$, we introduced some distinct patterns in the query distribution. It is described in the appendix. Table 4 shows the query response times for query sets $Q1$ and $Q2$ both with and without materialization. There is a significant overall improvement in performance compared to the case where we do not materialize any data for both query sets $Q1$ and $Q2$. The performance improvement is greater in the case of $Q1$ as we introduced some distinct patterns in the distribution that the system was able to successfully extract and materialize. Thus, a majority of the queries for query set $Q1$ could be answered using the materialized data store and not having to access the remote Web sources at all. This accounts for the impressive improvement in performance over no materialization.

(ii) TheaterLoc

TheaterLoc[23] is another Ariadne mediator application which provides integrated access to Web sources about movies and theatres, an interactive map server depicting their various locations and a video server from which users can see video trailers

Table 5. Effectiveness of extracting patterns in theaterloc.

Query set	Response Time (No optimization)	Response Time (With Materialization)	%improvement
$Q1$	22013 sec	1644 sec	93%

of movies playing at the selected theatres. This application is available online at http://www.isi.edu/ariadne.

Integrated access is provided to the following Web sources:

- Cuisinet (http://www.cuisinenet.com). Web source providing information about restaurants in various US cities.
- Yahoo Movies (http://movies.yahoo.com/movies/). Provides theater and movie showtime information.
- Hollywood.com (http://www.hollywood.com). Movie previews source.
- E-TAK Geocoder (http://www.geocode.com). Geocodes street addresses.
- US Census Map Server (http://tiger.census.gov/cgi-bin/mapbrowse-tbl). Online interactive map server.

The results for the effectiveness of materializing frequently accessed classes of data for TheaterLoc are shown in Table 5. As for the countries application, in this case too we measured response times with and without materialization against of generated query set $Q1$ of 200 queries. We introduced some distinct patterns of frequently accessed classes in the TheaterLoc domain in the query set $Q1$. In this application as well, materializing data provides a significant performance improvement over the case when no data is materialized.

(iii) Flight Delay Predictor

Finally we have the Flight Delay Predictor application for performing flight delay predictions given information about a particular flight's departure and arrival times and airports, airline name, weather predictions from the Yahoo weather service (http://weather.yahoo.com) and historical flight and weather data. A demo of this application is available at http://www.isi.edu/ariadne/demo/tw/.

Table 6 shows the improvement in performance due to materializing frequently accessed classes in the Flight Delay Predictor. Query set $Q1$ is a set of 200 generated queries having patterns of frequently accessed classes in the Flight Delay Predictor domain. The performance gain of 34% (over no materialization) is substantial. However it is significantly less than the performance improvement achieved for the countries and TheaterLoc applications. This is because all queries for the Flight Delay Predictor require fetching just 2 pages from one Web source (retrieving the weather predictions from the Yahoo! weather service). As a result the queries to the Delay Predictor even without any materialization do not take a very long time to execute.

Table 6. Effectiveness of extracting patterns in flight delay predictor.

Query set	Response Time (No optimization)	Response Time (With Materialization)	%improvement
$Q1$	2399 sec	1603 sec	34%

To summarize, the above results do indeed validate our claim that materializing data extracted as frequently accessed classes in user queries is effective in optimizing the performance of a mediator application. The degree to which performance is improved varies across applications depending on the patterns in the query distribution and also the time spent in answering queries using only the original Web sources in the different applications.

5. Complexity

As CM has been designed to take as input a large number of previous user queries (a typical number might be say 1000 queries) for extracting patterns, the complexity of running CM is also a matter of concern. We present below a complexity analysis of the algorithm with some reasonable assumptions about the query distribution. For a particular domain class:
Number of queries = N.
Number of attributes of domain class = M.
Number of attributes on which constraints may be specified = K.

We analyze the complexity by analyzing each step of the CM algorithm:

(i) Creating the ontology of subclasses of interest. For each query
SELECT $A_1, A_2, \ldots A_n$
FROM C
WHERE P
Suppose P consists of a single predicate. In case of string equality constraints (of the form $A_i = V$) we form subclasses of P based on the value of A_i. We assume that the number of such subclasses is a small constant t. The assumption is valid as we will not form subclasses of C based on an attribute A_i that may be the primary key of C (thus resulting in several subclasses). Also for numeric constraints, both equality and range we partition into subclasses based on ranges for the value of A_i and we can assume that the number of such subclasses is again a small constant t.

Since we can specify constraints on at most K attributes and the number of subclasses for a single attribute value can be at most t, it is easy to see that in an ontology (see Fig. 5) the number of subclasses is at most $(1+t)^K$. This is because for each of the K attributes for which we can specify a constraint we may either not specify a constraint or the constraint will be one of the t possible values for the attribute in case of string constraints. In case of numeric constraints it will be

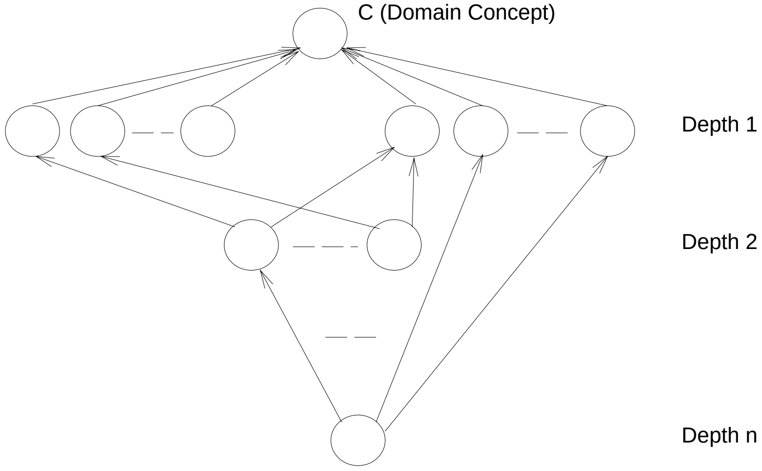


Fig. 5. Ontology of subclasses of interest.

in one of the t possible ranges. However for N queries the number of subclasses that can be created will be much less than what $(1+t)^K$ could be. In the ontology at depth 1, we have at most tK subclasses. Now for each query we will create subclasses at depth 1 in the ontology (unless they already exist) and 1 subclass at depth n if the query constraint has n predicates and $N \succ 1$. Thus for N queries the number of subclasses created can be at most $(tk + N)$. For each query we may create up to $(nt + 1)$ new subclasses where n is the number of predicates in the query constraint. This is because we could have up to nt new subclasses at depth 1 and 1 new subclass at depth n. For each of these new subclasses we need to ensure that either the subclasses are already present in the ontology or place them at appropriate places in the ontology. A linear search on the ontology is required which takes time at most $(Kt + 1)(tK + N)$ for each query as at most K predicates can be present in the query constraint. As the number of queries is N, the entire first step of creating the ontology takes time $N(Kt + 1)(Kt + N)$ which is $O(N^2K + K^2N)$.

(ii) The second step is for each subclass, to cluster together the groups of attributes based on similarity of cluster and similarity of frequency with which they have been queried. As mentioned earlier we use an approximate 2D clustering algorithm and it has a running time that is quadratic in the number of queries. Assuming an even distribution of the N queries in the $(tk + N)$ subclasses, we have on an average of $N/(tk + N)$ queries in each subclass. Also to measure “similarity” of attribute clusters in each subclass, we compare clusters pairwise and each comparison takes time at most M^2 as M is the total number of attributes for the domain concept. The clustering of attribute groups in each subclass thus takes time $= c\, M^2(N/(tk+N))^2$ where c is a constant. As we have at most $(tK + N)$ subclasses the entire clustering step takes time $= c\ (tK + N)M^2(N/(tk + N))^2 = c\ M^2N^2/(tK + N)$ which is $O(M^2N)$ (assuming $M < N$).

(iii) The final step is to cluster based on coverings. The number of coverings is obviously less than $(tK + N)$, the total number of subclasses in the ontology. Now when we consider a covering we merge across at most t subclasses. This is because we merge across the direct subclasses of a particular class and this can be at most t. Recall that we start with a seed cluster and compare with other clusters to see if there is a good overlap. Each comparison (of the seed cluster with another) takes time M^2.Again assuming an even distribution of queries among the subclasses in the ontology we have an average of $N/(tK + N)$ queries in any subclass. As there are at most t subclasses in a covering, the total number of attribute clusters is at most $tN/(tK + N)$. If each cluster is compared with every other cluster in the covering (in the course of searching for overlapping clusters), the number of such comparisons is $(tN/(tK+N))^2$. The actual number of comparisons while searching for overlapping clusters to merge will be less than $(tN/(tK+N))^2$ as we will go on removing clusters on merging them. Merging across each covering thus takes time at most $M^2(tn/(tK + N))^2$. As there are at most $(tK + N)$ coverings, the entire third step takes time $M^2(tN)^2/(tK + N)$ which is $O(M^2N/K)$. As the 3 different steps have complexity $O(N^2K+K^2N), O(M^2N)$ and $O(M^2N/K)$ respectively, the complexity for the overall algorithm is $O(N^2K^2+M^2N/K)$. As K can range from 1 to M (number of attributes in the domain concept) this can be further simplified to $O(M^2N+N^2M)$.

$O(M^2N + N^2M)$ seems satisfactory as the complexity is polynomial in the number of attributes in a domain concept and the number of queries analyzed. The time for running CM will thus be acceptable even for large values of M and N. The purpose of analyzing the complexity of CM was to ensure that there are no sources of complexity that would make the running time of CM a matter of concern. For instance if it were say exponential in the number of queries or attributes. However with reasonable assumptions about the query distribution, we have shown that CM is a polynomial time algorithm and thus the running time will not be an issue.

6. Related Work

There is a whole body of work on optimizing performance by caching or materializing data in database, operating system or Web server environments and we now describe how our work relates to these approaches. We materialize data in a manner similar to a semantic caching[20] or *predicate based*[24] caching system. A problem with the semantic caching approach is that the containment checking problem is hard and having a large number of semantic regions creates performance problems. A solution proposed in Ref. 24 is to reduce the number of semantic regions by merging them whenever possible. This is in fact an idea we have built on. In the CM algorithm we have presented an approach for systematically creating new semantic regions to consider for materializing and merging them when possible. We have also proposed a relaxed merging of semantic regions in addition to exact merging.

For the mediator environment an approach to caching is described in Ref. 14. The focus of this work however is caching for a mediator environment where information is integrated from sources that may not be traditional database systems. The contribution is a caching approach based on first estimating the cost of accessing various sources based on statistics of costs of actually fetching data from the sources. In this approach reasoning about cache contents is done through the notion of *invariants* which are expressions that show possible substitutions or rewritings of queries. This approach provides very limited semantic reasoning capabilities about the contents of the cached data as compared to our approach in which we are able to perform more powerful reasoning of the materialized data contents through LOOM.

Another approach to caching for federated databases is described in Ref. 25. This is also a semantic caching approach where the data cached is described by queries. They also define some criteria for choosing an optimal set of queries to cache. Finding the optimal set is an NP-complete problem and an A^* algorithm is used to obtain a near optimal solution. A limitation of this approach is that the cached classes can only be in terms of classes in a predefined hierarchy of classes of information for a particular application. Our approach is much more flexible in that we dynamically construct classes of information to materialize.

There is some research on materializing the Web described in Ref. 8. The focus of this work however is to build a fully materialized view over Web data for data in a user specified generic domain of interest. Our goal is different in that we wish to just partially materialize data to improve performance for mediators that would otherwise access data from remote sources.

Our work is also related to recent work on view selection in a data warehousing environment. One of the most important decisions in designing a data warehouse is selecting what views to materialize so that the total query response time is minimized with a constraint such as limited storage space and/or cost of maintaining the views. Yang *et al.*[26,27] show that this is an intractable problem and present heuristic algorithms for near optimal solutions. However the warehousing problem differs from our problem in several aspects. In warehousing there is a fixed set of views and a decision is to be made for each view whether to materialize it or not, in the mediator environment, however, we dynamically propose new "views" (classes of data to materialize) with the additional constraint that the number of such classes be small. The warehouse design approach described in Ref. 4 does not start with a fixed set of views. Instead they start with the set of all queries (of interest) materialized as views and then produce alternative view selections for materialization by modifying views, decomposing views, eliminating views or generating auxiliary views. In this case, however, a rewriting of any query over the set of views is already available. In our system the "rewriting" involves generating a query plan over the materialized and remote sources which is combinatorially hard. Also in warehousing, the cost that we are trying to minimize is that of reading large relation tables from disk into main memory, and main memory operations on these tables. In the mediator environment the dominant cost is that of retrieving data from the remote Web sources which is what we attempt to minimize. Work on mediators that support views that integrate data from multiple sources is described in Ref. 28. The views could be materialized (i.e. the data is materialized locally), virtual (i.e. the data is obtained from the remote sources at run time) or hybrids of both. However, specifying exactly which views are materialized, virtual or hybrid is done using annotations. They do not address the problem of how to optimally annotate views in a mediator. In our approach, we automate the process of choosing what to materialize based on many factors.

The problem of Web sources having limitations in terms of query processing has been looked at by work in the area of wrappers.[29,30] Query planning in these systems is done using *capabilities based rewriting* (CBR) where the limitations of query processing of wrappers are taken into account. If a wrapper cannot support a certain type of query, the query (plan) is rewritten including additional operations above the operations that the wrapper can support. Our approach is different from CBR in that first, any additional operations (not supported by the source) are performed by the wrapper itself. Next, as performing such additional operations causes processing the queries to be expensive, we identify and materialize locally information that will speed up the processing of the expensive queries.

Finally, the problem of extracting patterns from queries is somewhat similar to the problem of data mining, particularly that of mining *association rules*.[31] The problem of mining association rules is that of extracting implications of the form $X \Rightarrow Y$ from a database where $X \subset I$ and $Y \subset I$ and $X \cap Y = \phi$, where $I = \{i_1, i_2, \ldots i_m\}$ is a set of data items. The patterns that we extract from queries can also be looked upon as implications. For instance a pattern such as "economy and population of European Countries" is an implication European Countries $\Rightarrow$ (economy, population). Extracting patterns however differs from mining association rules in several respects. First, when mining association rules, the antecedent X in an implication $X \Rightarrow Y$ could be any $X \subset I$ where at least a certain minimum percentage of transactions in the database contain $X \cup Y$. In extracting patterns the antecedents are just the individual classes and subclasses of data queried. Second, we do not cluster data items in the consequents in the implication rules whereas in extracting patterns we do try and group together data items in consequents (for the same antecedent) that overlap well. Third, when extracting patterns we can extract patterns when the antecedents (classes and subclasses) are organized in a class/subclass hierarchy. This hierarchy is dynamically generated. In mining association rules however the hierarchy in which antecedents may be organized is predefined and fixed.[5] Finally mining association rules is often done for databases of very large size and the optimizations focus on issues like minimizing the disk scans of the relations for data mining. In extracting patterns the entire query distribution being analyzed can fit into main memory and the optimizations for minimizing disk scans are not required. For all the above reasons the algorithms for mining association rules cannot be directly applied to the problem of extracting patterns which is why we developed the CM algorithm for this task.

7. Future Work and Conclusion

We have described an approach for optimizing the performance of information mediators. The major contributions of this paper can be summarized as follows:

- We described an approach for selecting data to materialize based on a combination of user query distribution, source structure analysis and update cost.
- We presented an algorithm for extracting patterns from a distribution of user queries in a compact manner.
- We demonstrated the effectiveness of our algorithm in extracting patterns and also the effectiveness of materializing frequently accessed classes in optimizing mediator performance.

The extraction of patterns from a query distribution can be viewed as "query mining" on a query distribution. Extracting such patterns from user queries could be of interest in areas besides materialization. One example is E-commerce where sellers may be interested in analyzing online user queries and request to find classes of items (data) that users are most interested in. The user profile may also be analyzed in addition to the queries. An example of a useful pattern extracted for an online book store may be "college student customers interested in humor/fiction mostly buy books by PG Wodehouse or Evelyn Waugh". It would thus be useful to investigate how an algorithm like CM may be enhanced or modified to satisfy the needs of query mining in applications like E-commerce. For instance being able to learn a more sophisticated description and augmenting the analysis of the user query distribution with customer profiles are some of the enhancements that may be needed.

The described work is part of a completed effort to optimizing mediator performance by materializing data, that also takes other factors of source structure analysis and update aspects into account. A description of these other components is however beyond the scope of this paper.

A. COUNTRY Relation

Attributes of COUNTRY relation.
Relation: COUNTRY
Attributes: (geography location map_references region area total_area land_area comparative_area land_boundaries coastline maritime_claims international_disputes climate terrain natural_resources land_use irrigated_land environment note people population age_structure population_growth_rate birth_rate death_rate net_migration_rate infant_mortality_rate life_expectancy_at_birth total_fertility_rate nationality ethnic_divisions religions languages literacy labor_force government

names digraph type capital administrative_divisions independence national_holiday constitution legal_system suffrage executive_branch legislative_branch judicial_branch political_parties_and_leaders other_political_or_pressure_groups diplomatic_representation_in_US us_diplomatic_representation organization flag economy overview national_product national_product_real_growth_rate national_product_per_capita inflation_rate_consumer_prices unemployment_rate budget exports imports external_debt industrial_production electricity industries agriculture illicit_drugs economic_aid currency exchange_rates fiscal_year transportation railroads highways inland_waterways pipelines ports merchant_marine airports communications telephone_system radio television defense_Forces branches manpower_availability defense_expenditures)

B. Query Set Q1

Details of distribution:
1. 30% of the queries are of the form:
SELECT *A*
FROM COUNTRY
WHERE name= *C*;
where *A* is a set of attributes and *C* is a country name. *A* and *C* are generated for each query of this form. In 80% of the instances of *A* all the attributes in set *A* lie within a set of 3 predefined "favourite" attributes. The remainder 20% contain randomly selected attributes. We chose the favourite set to be {imports, exports, economy}. For *C*, we randomly selected a country name from the set of all country names. The names were picked with approximately equal probabilities.

2. 20% of the queries are of the form:
SELECT *A*
FROM COUNTRY
WHERE region = "ASIA";
80% of the instances of *A* are within a set of 4 favourite attributes that we chose to be {location, map_references, area, climate}.

3. Another 20% of the queries are of the form:
SELECT *A*
FROM COUNTRY
WHERE region = "ASIA" and organization = "NATO";
As above 80% of the instances of *A* are within a set of 2 favourite attributes that in this case we chose to be {defense_expenditure, external_debt}.

4. Finally the remaining 30% queries are of the form
SELECT *A*
FROM COUNTRY

WHERE region = X;
where X is a region such as ASIA, EUROPE etc. The queries are uniformly distributed amongst the (six) possible values of region. Again, 80% of the instances of A are within a set of 4 favourite attributes that in this case we chose to be {economy, currency, religions, literacy} .

References

1. R. MacGregor, A deductive pattern matcher, *Proc. AAAI-88, Nat. Conf. Artif. Intell.*, St. Paul, MN, 1988.
2. Informix universal server, version 9.1. Technical Report, Informix Press, Menlo Park, CA, March, 1997.
3. R. MacGregor, H. Chalupsky and E. Melz, Powerloom manual, Available online at http://www.isi.edu/isd/LOOM/PowerLoon/documentation/manual.html
4. D. Theodoratos and T. K. Sellis, Designing data warehouses, *Data Knowledge Eng. (DKE)* **31**, 3 (1999) 279–301.
5. R. Srikant and R. Agrawal, Mining generalized association rules, *Proc. 21st Int. Conf. Very Large Databases*, Zurich, Switzerland, 1995.
6. O. Etzioni and D. S. Weld, A softbot-based interface to the Internet, *Comm. ACM*, **37**, 7 1994.
7. G. Wiederhold, Mediators in the architecture of future information systems, *IEEE Comput.*, March 1992.
8. M. D. Rosa, T. Catarci, L. Iocchi, D. Nardi and G. Santucci, Materializing the web, *Proc. Third IFCIS Int. Conf. Coop. Inf. Syst. (CoopIS '98)*, New York, 1998.
9. M. Genesereth, A. Keller and O. Duschka, Infomaster: An information integration system, *Proc. ACM SIGMOD Int. Conf. Management Data*, Tucson, AZ, 1997.
10. J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman and J. Widom, Information translation, mediation, and mosaic-based browsing in the tsimmis system, *Proc. ACM SIGMOD Int. Conf. Management Data*, San Jose, CA, 1995.
11. Z. Ives, D. Florescu, M. Friedman, A. Levy and D. Weld, An adaptive query execution engine for data integration, *Proc. ACM SIGMOD Int. Conf. Management of Data*, Philadelphia, PA, June 1999.
12. R. Bayardo, W. Bohrer, R. Brice, A. Cichocki, G. Fowler, A. Helal. V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh and D. Woelk, Semantic integration of information in open and dynamic environments, Technical Report MCC-INSL-088-96, MCC, Austin, Texas, 1996.
13. A. Tomasic, L. Raschid and P. Valduriez, A data model and query processing techniques for scaling access to distributed heterogeneous databases in disco, *Invited paper in the IEEE Transaction Computers, Special Issue on Distributed Computing System* 1997.
14. A. Adali, K. S. Candon, Y. Papakonstantinou and V. S. Subrahmanian, Query caching and optimization in distributed mediator systems, *Proc. ACM SIGMOD Intelligent Conf. Management Data*, Tucson, AZ, 1997.
15. Y. Arens, C. A. Knoblock and W.-M. Shen, Query reformulation for dynamic information integration, *J. Intell. Inf. Syst., Special Issue on Intelligent Information Integration* **6**, 3 (1996) 99–130.

16. C. A. Knoblock, S. Minton, J.-L. Ambite, N. Ashish, P. J. Modi, I. Muslea, A. Philpot and S. Tejada, Modeling web sources for information integration, *Proc. Fifteenth Natl. Conf. Artif. Intell. (AAAI)*, Madison, WI, 1998.
17. N. Ashish, C. A. Knoblock and C. Shahabi, Selectively materializing data in mediators by analyzing user queries, *Fourth Int. Conf. Coop. Inf. Syst. (CoopIS)*, Edinburgh, Scotland, September 1999.
18. A. Chankunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz and K. J. Worrell, A hierarchical internet object cache, Technical Report 95–611, Computer Science Department, University of Southern California, Los Angeles, CA, 1995.
19. Y. Arens and C. A. Knoblock, Intelligent caching: Selecting, representing and reusing data in an information server, *Proc. Third Int. Conf. Inf. Knowledge Management*, Gaithersburg, MD, 1994.
20. S. Dar, M. J. Franklin, B. T. Jonsson, D. Srivastava and M. Tan, Semantic data caching and replacement, *Proc. 22nd VLDB Conf.* Mumbai (Bombay), India, 1996.
21. S. Chawathe, A. Rajaraman, H. Garcia-Molina and J. Widom, Change detection in Hierarchically structured information, *Proc. ACM Int. Conf. Management Data (SIGMOD)*, Montreal, Quebec, Canada, 1996.
22. P. Scheuermann, J. Shim and R. Vingralek, Watchman: A data warehouse intelligent cache manager, *Proc. 22nd VLDB Conf.*, Mumbai (Bombay), India, 1996.
23. G. Barish, C. A. Knoblock, Y.-S. Chen, S. Minton, A. Philpot and C. Shahabi, Theaterloc: A case study in information integration, *IJCAI Workshop Intell. Inf. Integration*, Stockholm, Sweden, 1999.
24. A. M. Keller and J. Basu, A predicate-based caching scheme for client-server database architectures, *VLDB J.* **5**, 2 (1996) 35–47.
25. A. Goni, A. Illarramendi, E. Mena and J. M. Blanco, An optimal cache for a federated database system, *J. Intell. Inf. Syst.* **1**, 34, 1997.
26. J. Yang, K. Karlapalem and Q. Li, Algorithms for materialized view design in data warehousing environment, *Proc. 23 Int. Conf. Very large Databases*, Athens, Greece, 1997.
27. H. Gupta and I. S. Mumick, Selection of views to materialize under a maintenance cost constraint, submitted to VLDB-98, 1998.
28. R. Hull and G. Zhou, A framework for supporting data integration using the materialized and virtual approaches, *Proc. ACM Int. Coof. Management Data (SIGMOD)*, Montreal, Quebec, Canada, 1996.
29. Y. Papakonstantinou, A. Gupta and L. Haas, Capabilities based query rewriting in mediator systems, *Distr. Par. Databases* **6**, 1 (1998) 73–110.
30. L. Bright, L. Raschid and M. E. Vidal, Optimization of wrappers and mediators for web accessible data sources (websources), *Workshop Web Inf. Data Management (WIDM)*, Washington DC, 1998.
31. R. Agrawal, T. Imielinski and A. Swami, Mining associations between sets of items in massive database, *Proc. ACM SIGMOD Int. Conf. Management Data*, Washington D. C., 1993.