To appear in the *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases, Washinton, D.C., 1993*

Learning Database Abstractions For Query Reformulation*

Chun-Nan Hsu
Department of Computer Science
University of Southern California
Los Angeles, CA 90089-0782
(213) 740-9328
chunnan@cs.usc.edu

Craig A. Knoblock
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
(310) 822-1511
knoblock@isi.edu

Abstract

The query reformulation approach (also called *semantic query optimization*) takes advantage of the semantic knowledge about the contents of databases for optimization. The basic idea is to use the knowledge to reformulate a query into a less expensive yet equivalent query. Previous work on semantic query optimization has shown the cost reduction that can be achieved by reformulation, we further point out that when applied to distributed multidatabase queries, the reformulation approach can reduce the cost of moving intermediate data from one site to another. However, a robust and efficient method to discover the required knowledge has not yet been developed. This paper presents an example-guided, data-driven learning approach to acquire the knowledge needed in reformulation. We use example queries to guide the learning to capture the database usage pattern. In contrast to the heuristic-driven approach proposed by Siegel, the data-driven approach is more likely to learn the required knowledge for the various reformulation needs of the example queries. Since this learning approach minimizes the dependency on the database structure and implementation, it is applicable to heterogeneous multidatabase systems. With the learning capability, the query reformulation will be more effective and feasible in real-world database applications.

*The research reported here was supported by Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency under contract no. F30602-91-C-0081. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, RL, the U.S. Government, or any person or agency connected with them.

1 Introduction

Query optimization methods have been studied since the introduction of declarative data models and languages [Ullman 88]. This is because it is often difficult to efficiently implement declarative queries. The *query reformulation* approach, also known as *semantic query optimization* approach in previous work [Chakravarthy et al. 90, Hammer and Zdonik 80, King 81, Siegel 88], addresses the problem differently from the conventional syntactical approaches [Apers et al. 83,Jarke and Koch 84] in that it brings to bear a richer set of knowledge about the contents of databases to optimize queries. The use of semantic knowledge offers more potential for cost reduction than that can be derived from syntactic and physical aspects of queries alone.

Figure 1 illustrates a simple example of how query reformulation works in a query processing system. Suppose we have a database containing 3 instances, and each has 3 attributes, $A1$, $A2$ and $A3$, where $A1$ is indexed. We also have a set of *database abstractions*, which are rules that describe the contents of a database. Suppose a query that contains two constraints is addressed to the database system. The query reformulation uses the database abstractions to reformulate the given query into a less expensive yet equivalent one to reduce the cost of query execution. In this example, the query reformulation unit reformulates the given query into a query that contains only one constraint. This remaining constraint is on an indexed attribute so it is less expensive, while the retrieved data is still the same as that retrieved by the original query. In addition to adding a constraint on indexed attribute, there are many other ways to reduce the cost of the query by reformulation [King 81]. For example, we can predict that the query will return `NIL` (empty set). The reformulation system can use the database schema to estimate the access cost of the queries and guide the search for the least expensive equivalent query.

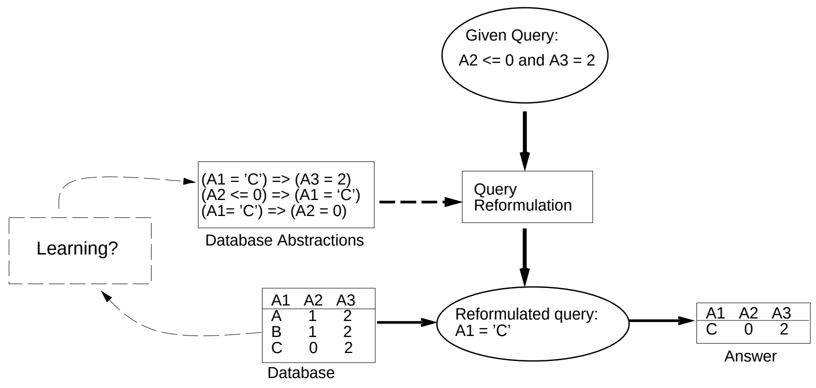

Figure 1: Learning for Query Reformulation

To make this approach feasible, we need a very efficient algorithm for the reformulation, so that the overhead will not exceed the saving. We have developed an efficient prototype reformulation algorithm to address this issue [Hsu and Knoblock 93]. The next issue is that we need a robust and efficient methodology to acquire sufficient knowledge for reformulation. Most of the previous work relies on the semantic integrity constraints that are encoded by domain experts. However, it is not always the case that the domains of databases are well understood so that sufficient semantic integrity constraints can be easily encoded. As indicated in dash lines in Figure 1, we would like a system that can automatically learn the database abstractions. This paper describes an example-guided, data-driven learning approach to address this problem.

The idea to automatically derive rules for reformulation is proposed originally by [Siegel 88]. In his approach, although example queries are used, the learning is mainly driven by a fixed set of heuristics, which are designed based on the database structure and implementation. Our approach differs from theirs in that we do not rely on explicit heuristics. Instead, our approach focuses more on the example queries and the data they retrieve to identify the rules needed for reformulation. The advantage of our approach is that it minimizes the dependency on the database structure and implementation, and it is more likely to capture the semantic aspects of the queries, and the various reformulation needs of the example queries.

The reminder of this paper is organized as follows. The next section briefly describes more about the query reformulation approach and what kind of knowledge is used. Section 3 describes our learning approach. Section 4 discusses the issues of maintaining the learned database abstractions. Section 5 compares this learning approach with other related work in knowledge discovery and machine learning. Section 6 reviews the contributions of the paper, and discusses the general issues that arise from this learning approach.

2 Query Reformulation

The goal of the query reformulation is to search for the least expensive query from the space of semantically equivalent queries to the original one. Two queries are defined to be *semantically equivalent* [Siegel 88] if they return identical answer given the same contents of the database[1]. The reformulation from one query to another is by logical inference using *database abstractions*, the abstracted knowledge of the contents of relevant databases. The database abstractions describe the databases in terms of the set of closed formulas of first-order logic. These formulas describe the database in the sense that they are satisfied by all instances in the database. They are of the form of implication rules with an arbitrary number of range propositions on the antecedent side and one range proposition on the consequent side. Figure 2 shows the schema and a small set of database abstractions for the database `geoloc`, which stores data of geographical locations. In all formulas the variables are implicitly universally quantified.

The first two rules in Figure 2 state that for all instances in the database, the value of its attribute country name is `"Germany"` if and only if the value of its attribute country code is `"FRG"`. The range propositions can be an interval for attributes of numeric type(see Rule 3) or a set of possible values for attributes of string type(see Rule 5). Figure 3 shows three queries addressed to the database `geoloc`. The query `Q1` retrieves the `geocode` of the

[1]There are other definitions of semantic equivalence [King 81] that require the queries to return identical answer given **any** contents of the database. However, this more restrictive definition would require using semantic integrity constraints, which are not usually available.

Schema:

```
      (Database      geoloc
       :Attributes   (latitude       real_number :indexed)
                     (longitude      real_number :indexed)
                     (geocode        string      :length 4)
                     (country_name   string      :length 20)
                     (country_code   string      :length 3))
```

Rules:

1:(geloc.country_name = "Germany") $\Longrightarrow$ (geoloc.country_code = "FRG")

2:(geoloc.country_code = "FRG") $\Longrightarrow$ (geoloc.country_name = "Germany")

3:(geoloc.country_code = "FRG") $\Longrightarrow$ (47.15 $\leq$ geoloc.latitude $\leq$ 54.74)

4:(geoloc.country_code = "FRG") $\Longrightarrow$ (6.42 $\leq$ geoloc.longitude $\leq$ 15.00)

5:(geoloc.country_name = "Taiwan")
 $\Longrightarrow$ (geoloc.geocode $\in$ ("gdpp" "wcsp" "bccc" "gtsa"))

6:(51.15 $\leq$ geoloc.latitude $\leq$ 53.74)$\wedge$(7.55 $\leq$ geoloc.longitude $\leq$ 14.87)
 $\Longrightarrow$ (geoloc.country_name = "Germany")

Figure 2: Example Database Schema and Database Abstractions

geographical locations in Germany, while query `Q2` retrieves the `geocode` for the instances with `country_code "FRG"`. The first clause `(geoloc ?geoloc)` in both queries binds the variable `?geoloc` to the instances of database `geoloc`. The second and third clauses bind the variable and constant to the values of attributes of `?geoloc` respectively. With rules 1 and 2, we can reformulate the query `Q1` to `Q2` by replacing the constraint on `country_name` with the constraint on `country_code`. We can inversely reformulate `Q2` to `Q1` with the same rules. Basically, there are three types of reformulation. Given a query Q, let $C_1, \ldots, C_k$ be the set of constraints in Q, the following *reformulation operators* return a semantically equivalent query:

- **Constraint Addition:** Given a rule $A \rightarrow B$, if a subset of constraints in Q implies A, then we can add constraint B to Q.
- **Constraint Deletion:** Given a rule $A \rightarrow B$, if a subset of constraints in Q implies A, and B implies C_i, then we can delete C_i from Q.
- **Query Refutation:** Given a rule $A \rightarrow B$, if a subset of constraints in Q implies A, and B implies $\neg C_i$, then we can assert that Q will return `NIL`.

Replacing constraints is treated as a combination of addition and deletion. Note that these reformulation operators do not always lead to more efficient versions of the query. Knowledge about the access cost of attributes is required to guide the search. Usually, we can estimate the cost from the database schema. For example, because the string length of `country_name` is long and expensive to evaluate, reformulating `Q1` to `Q2` from rules 1 and 2 will reduce this cost. From rule 3, we can reformulate `Q2` to `Q3` by adding a new constraint on an indexed attribute `latitude`. The DBMSs will take advantage of the indexed attribute

Q1:
```
(retrieve (?geocode)
  (:and (geoloc ?geoloc)
        (geoloc.geocode ?geoloc ?geocode)
        (geoloc.country_name ?geoloc "Germany")))
```
Q2:
```
(retrieve (?geocode)
  (:and (geoloc ?geoloc)
        (geoloc.geocode ?geoloc ?geocode)
        (geoloc.country_code ?geoloc "FRG")))
```
Q3:
```
(retrieve (?geocode)
  (:and (geoloc ?geoloc)
        (geoloc.geocode ?geoloc ?geocode)
        (geoloc.country_code ?geoloc "FRG")
        (geoloc.latitude ?geoloc ?latitude)
        (geoloc.longitude ?geoloc ?longitude)
        (?latitude ≥ 47.15) (?latitude ≤ 54.74)))
```

Figure 3: Equivalent Queries

to speed up the retrieval. The reformulation process does not need to be step-by-step, from one equivalent query to another, as described here. We have developed an efficient reformulation algorithm [Arens et al. 93, Hsu and Knoblock 93], which fires all applicable database abstractions simultaneously, and then formulates the least expensive equivalent query from the partial result of the rule applications.

We also extended the algorithm to reformulate queries to distributed databases [Hsu and Knoblock 93, Arens et al. 93]. We found that the reformulation approach can reduce the intermediate data that is transferred from remote database sites to the local retrieval system. This is often the most costly aspect of the multidatabase queries. Consider the following hypothetical example. Suppose that there are two databases in the multidatabase system, one for the data of ports, and another for ships. A query is given to retrieve the data of ports that can accommodate ships of the type tanker. This query may be very expensive because all data of ports must be retrieved and compared with the data of tankers. Suppose that the system learned from the ship database that if the ship type is tanker, then its draft is at least 10m. With this knowledge, the original query can be reformulated such that the system only retrieves data of ports whose depth are greater than 10m. This additional constraint may reduce a significant amount of data needed to be retrieved from the ship database. The cost is thus reduced substantially.

Table 1 provides statistical data concerning the preliminary experimental results of the multidatabase query reformulation. In this experiment, the SIMS system is connected with two remote Oracle databases. One of the databases consists of 16 tables, 56,078 instances, the other database consists of 14 tables, 5,728 instances. The queries used were selected from the set of SQL queries constructed by the original users of the databases. The first three queries are single database queries, while the others are multidatabase queries. This initial results indicate that our algorithm can reduce the total cost of the retrieval substantially. In most multidatabase queries, the amount of intermediate data is reduced significantly. The overheads of reformulation is included in the total execution time and is relatively small compared to the cost of executing most queries.

The system used 267 database abstraction rules in this experiment. These rules were prepared by compiling the databases. The compiling process is a semi-automatic method that requires the external guidance from a programmer to search for the useful rules for reformulation.

query	1	2	3	4	5	6	7	8	9	10
planning time (sec)	0.5	0.3	0.6	2.1	1.1	0.7	0.7	0.5	0.5	0.8
reformulation time	0.1	0.1	0.0	0.5	0.1	0.0	0.0	0.1	0.1	0.3
rules fired (times)	37	18	11	126	63	8	17	15	19	71
query exec. time w/oR[a]	0.3	8.2	0.6	12.3	11.3	2.0	251.0	401.8	255.8	258.8
query exec. time w/R[b]	0.3	1.5	0.0	11.3	11.1	0.0	0.3	207.5	102.9	195.2
total elapsed time w/oR	0.8	8.5	1.2	14.4	12.4	2.7	251.7	402.3	256.3	259.6
total elapsed time w/R	0.9	1.9	0.6	13.9	12.3	0.7	1.0	208.1	103.5	196.3
intermediate data w/oR	-	-	-	145	41	1	810	956	808	810
intermediate data w/R	-	-	-	145	35	0	28	233	320	607

[a]w/oR = Without reformulation.
[b]w/R = With reformulation.

Table 1: Experimental Results

3 Learning Database Abstractions

The effectiveness of the query reformulation is determined by the existence of a useful set of database abstractions. Because the number of rules that can be derived from the database is combinatorially large, only a subset of the possibly derivable rules can be used in reformulation. Therefore, a methodology to *selectively* acquire a set of useful database abstractions is crucial in the query reformulation. There are many algorithms available to derive inference rules from databases [Cai et al. 91, Piatetsky-Shapiro 91] selectively. These algorithms usually require the data to be pre-classified. If the system applies these algorithms directly in our domain, the users must possess knowledge about database structure, usage pattern, and more, to properly trigger the system to learn.

In general, we want to learn the rules that: (1) will be applied frequently, (2) have low match cost, and (3) achieve high cost reduction. We can estimate the match cost of a rule by its syntactic length. It is difficult to predict the application frequency and the resulting cost reduction. This requires the information about the semantics of the database, and its usage pattern. A good learning approach should be able to capture this information to guide its

Figure 4: Database Abstraction Learning Approach: An Overview Diagram

The database abstraction learning is a two-phase process. In the first phase, the inductive concept formation algorithm [Cai et al. 91, Haussler 88, Michalski 83] will generate from the database an *alternative query* q', which is equivalent to the given query, but with lower cost. In the second phase, the system operationalizes the *preliminary database abstraction*, $q \Longleftrightarrow q'$, to formulate the rules that can be used to reformulate q into q'. The *operationalization* process is to transform and refine the preliminary database abstraction so that the resulting rules will satisfy our operational criteria. There are two operational criteria. The first criterion is that the rule must be expressed in our required syntax, that is, inference rules with one range proposition on the consequent side. The *transformation* stage of phase 2 is to transform the preliminary database abstraction to meet this criterion. The second criterion is that the antecedent conjunctions of the rule should be as short as possible. The *refinement* stage of phase 2 takes the output rules of the transformation stage as input, and simplifies them to meet the second criterion. This process will reduce the match cost and increase the utility of the learned rules.

Figure 5 illustrates an example scenario of the database abstraction learning. We use the same database table as the one in Figure 1. The instances in the relevant database table are partitioned by the example query into positive instances ($answer$) and negative instances ($database - answer$, where "$-$" is the set difference). This partition, or classification,

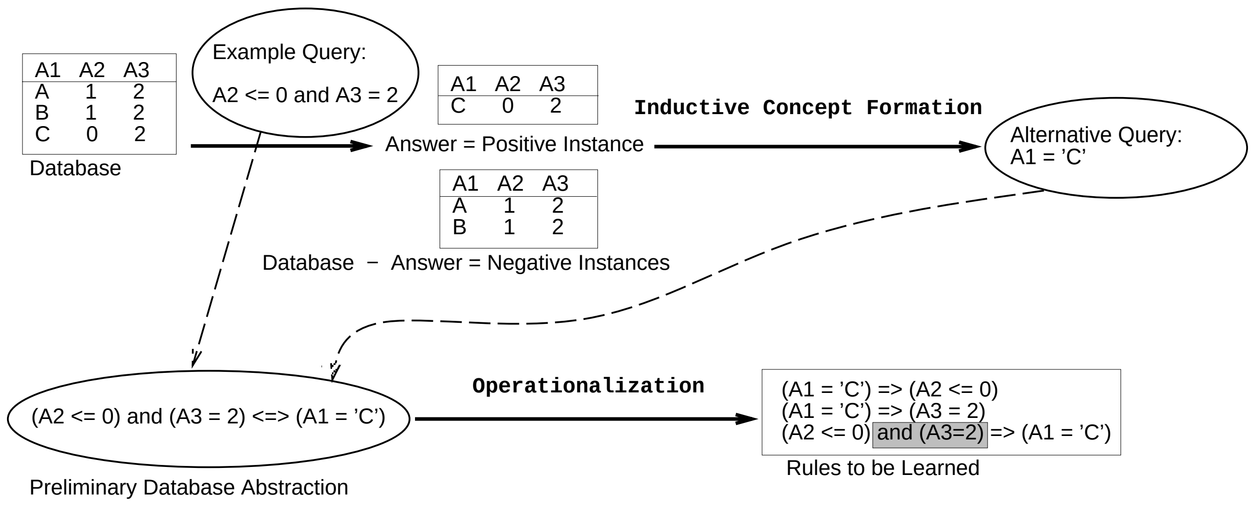

Figure 5: Database Abstraction Learning Approach: An Example Scenario

represents the class of queries that are equivalent to the example query. In this example, the positive set consists of one instance and the negative set consists of the remaining two instances. The inductive concept formation algorithm then generates an alternative query under the guidance of the inductive bias. The generated alternative query should be satisfied by all answer instances and none of the others. This is necessary and sufficient to guarantee the equivalence of two queries. In the example scenario, the alternative query `(A1 = 'C')` is formed. The preliminary database abstraction is a statement that asserts the equivalence of the alternative query and the example query explicitly. The operationalization process then takes this statement as input, and derives the rules we need to reformulate the given query to the alternative one.

The operationalization process consists of two stages. In the first stage, we use a logical inference procedure to transform the preliminary database abstraction into our required syntax. The equivalence in the preliminary database abstraction is converted to two implication rules:

$(1)(\texttt{A2} \leq \texttt{0}) \wedge (\texttt{A3} = \texttt{2}) \Longrightarrow (\texttt{A1} = \texttt{'C'})$

$(2)(\texttt{A1} = \texttt{'C'}) \Longrightarrow (\texttt{A2} \leq \texttt{0}) \wedge (\texttt{A3} = \texttt{2})$

Rule (2) can be further expanded to satisfy our syntax criterion:

$(3)(\texttt{A1} = \texttt{'C'}) \Longrightarrow (\texttt{A2} \leq \texttt{0})$

$(4)(\texttt{A1} = \texttt{'C'}) \Longrightarrow (\texttt{A3} = \texttt{2})$

After the transformation, we have proposed rules (1), (3), and (4) that satisfy our syntax criterion. Among them, rules (3) and (4) are short enough to satisfy our second operational criterion. No further refinement is necessary for them. These rules are then returned and learned by the system.

If the proposed rule has more than one antecedent, such as rule (1), then we use the *greedy minimum set cover* algorithm [Chivatal 79, Cormen et al. 89] to eliminate unnecessary constraints. In this example, we want to reduce the number of antecedents of rule (1). This problem can be reduced to the problem that given a collection of sets of data instances that satisfy $\neg$`(A2 ≤ 0)` $\vee$ $\neg$`(A3 = 2)`, find the minimum number of sets that cover the set of data instances that satisfy $\neg$`(A1 = 'C')`. Since the resulting minimum sets that cover $\neg$`(A1 = 'C')` is $\neg$`(A2 ≤ 0)`, we can eliminate `(A3 = 2)` (as in shaded box in Figure 5) and form the rule `(A2 ≤ 0)` $\Rightarrow$ `(A1 = 'C')`. These learned rules can be used to reformulate the given query into a more efficient one.

```
Learning_Alternative_Query(P,N,S)
  P is positive data;
  N is negative data;
  S is the schema of database table;
BEGIN
  1. q' = NIL;
  2. For each attribute $A$, find the range of their values $R$ in P,
        and construct candidate constraint from $R$.
     let A = {x | $\forall$ $A$, x = candidate constraint on $A$};
  3. For each x in A , compute gain(x) and cost(x);
  4. If none of x has a gain > 0 then return FAIL;
  5. Select x in A such that x has the highest gain(x)/cost(x);
  6. q' = q' $\cup$ {x}; A = A - {x};
     N = N - {n|$\forall$n, n $\in$ N $\wedge$ n is eliminated by x};
  7. If N = NIL, return q'; else go to 3;
END.

gain(x) = the size of {n|$\forall$n, n $\in$ N $\wedge$ n is eliminated by x};

cost(x) = E-cost * C-cost =
  evaluation_cost(x) * (P + N - gain(x)), if x is on an indexed attribute
  evaluation_cost(x) * (P + N)          , otherwise.
```

Figure 6: Inductive Learning Algorithm For Alternative Query

One of the polynomial time inductive learning algorithms that generate internal disjunctive concepts as the syntax of our queries is proposed by [Haussler 88]. This algorithm reduces the learning problem to the set coverage problem. The inductive bias is specified as a gain/cost ratio to bias the search for the shortest concepts. We can include more information such as the access cost of the attributes, to bias the search for both shorter and more efficient alternative queries. A well defined bias should lead the learner to tailor the database abstractions to the type of queries that are frequently asked. The algorithm that learns the alternative query is shown in Figure 6.

Suppose we want to derive an alternative query of `Q1` in Figure 3, from the example database table in Table 2. The additional column "`P or N`" in the table indicates whether an instance is positive or negative for `Q1`. In Step 1, the alternative query is initialized to empty. The system then extracts the candidate range constraints for each attribute from the positive instances and computes the gain and the cost for each candidate constraint.

geocode	country_name	country_code	latitude	longitude	P or N
atbr	France	FRN	43.1046	000.3200	N
bnsg	France	FRN	45.4600	002.1300	N
chbr	France	FRN	48.5706	007.0449	N
emkv	France	FRN	44.3100	005.2900	N
gdpp	Taiwan	TWN	24.2952	121.0338	N
wcsp	Taiwan	TWN	24.5230	121.1220	N
bccc	Taiwan	TWN	24.5124	121.1424	N
gtsa	Taiwan	TWN	24.5200	121.1214	N
lyre	Japan	JPN	35.5000	138.4000	N
qcmu	Japan	JPN	35.4620	139.3646	N
grty	Italy	ITL	41.4840	012.1509	N
jtln	Italy	ITL	41.5900	012.4400	N
tshe	Italy	ITL	41.3915	012.2640	N
agcc	Germany	FRG	50.4100	007.4000	P
atnt	Germany	FRG	52.1900	008.2000	P
bvdn	Germany	FRG	54.7445	009.4830	P
bhgl	Germany	FRG	51.1000	006.4200	P
csdm	Germany	FRG	50.3200	008.3200	P
fjhd	Germany	FRG	53.3900	007.2600	P
guye	Germany	FRG	49.5419	010.5553	P
girl	Germany	FRG	50.0540	008.3806	P
kmtb	Germany	FRG	49.1000	010.2200	P
mlna	Germany	FRG	49.1554	007.2147	P
nhkb	Germany	FRG	50.0300	008.2000	P
pdpf	Germany	FRG	48.5100	015.0000	P
ucla	Germany	FRG	49.3055	009.0520	P
uscw	Germany	FRG	49.5957	007.4700	P
vayq	Germany	FRG	47.1500	011.0700	P
wpcn	Germany	FRG	52.5400	008.5200	P
zwab	Germany	FRG	49.1519	007.2008	P

Table 2: Example Database Table

Five candidate constraints and their gain/cost ratio are listed in Table 3. The gain of a constraint is the number of the instances eliminated by (or that does not satisfy) the constraint. The cost of a constraint is the evaluation cost **E-cost** times the number of instances to be evaluated **C-cost**. The evaluation cost E-cost is the cost of evaluating whether an instance satisfies the constraint. This cost is proportional to the number of terms to compare in the given constraint times the cost of each comparison. For strings, the comparison cost of each term is their length defined in the schema, and for real numbers, 2. The other factor C-cost is dependent on whether the constrained attribute is indexed. For constraints on indexed attributes, C-cost is the number of the instances that satisfy the constraint. Otherwise, C-cost is the total number of instances, because to evaluate these constraints, the system must scan every instance in the database table. Therefore, for the first candidate constraint in Table 3, there are two terms to be compared and each has the comparison cost 2, so its E-cost is 4. Because the constrained attribute `latitude` is indexed, its C-cost is 18. The total cost is thus 4*18=72.

The system selects the first constraint on the attribute `latitude`, because it has the highest gain/cost ratio, and adds it to q' as a conjunction. The negative instances eliminated by the selected proposition are removed from the set of the negative instances. The system iterates this process until all of the negative instances are removed. Now there remains only one instance with country name `France` in the set of negative instances. In the next iteration, the gains and costs are updated for the remaining 4 constraints. The constraint

Candidate Constraint	gain	cost (E*C)		gain/cost
		E-cost	C-cost	
`(:and (?latitude ≥ 47.15)(?latitude ≤ 54.74))`	12	4	18	0.167
`(= ?country_code "FRG")`	13	3	30	0.144
`(:and (?longitude ≥ 6.42)(?longitude ≤ 15.00))`	9	4	21	0.107
`(= ?country_name "Germany")`	13	20	30	0.022
`(member ?geocode ("agcc" "atet" "babv" ...))`	13	68	30	0.006

Table 3: Candidate Constraints and Gain/Cost Ratio

on `country_code` is selected and 2 negative instances are removed. The resulting alternative query is `Q3` in Figure 3. This query is equivalent to and more efficient than `Q1`, as explained in Section 2.

The worst case complexity of this algorithm is $O(MP + NMmin(N, M))$, where N is the number of negative instances, P is the number of positive instances, and M is the number of attributes of the database table. This is a loose upper bound of the algorithm. In the average case, the complexity should be much better. In this complexity analysis, we assume that instances in the database need to be scanned linearly in the algorithm. In fact, there is no need to represent the positive and negative instances explicitly to implement our learning algorithm. We can retrieve the necessary information from the database through its DBMS rather than make a copy of the entire database table in the system's memory. We use the positive instances to extract the candidate atomic propositions, and the negative instances to count the gain of each atomic propositions. Both can be computed by sending queries to the DBMS. We can also take advantage of other facilities provided by the state-of-the-art DBMSs to enhance the efficiency of learning, such as, the delayed-evaluation for "views" (temporary databases) [Ullman 88]. The work by [Cai et al. 91] is a good example of a system that utilizes relational DBMS facilities in inductive learning.

The efficiency of the inductive learning algorithm can be further improved by pruning irrelevant attributes. We can use the semantic knowledge of the databases provided by the SIMS system [Arens and Knoblock 92, Arens et al. 93], on which we built the reformulation component, to identify which attribute is irrelevant.

4 Maintaining Database Abstractions

Although our learning approach is selective, after learning from a large number of queries, the number of the database abstractions could become so large that they degrade the reformulation algorithm's efficiency. This problem is referred to as the *utility problem* [Minton 88]. The utility problem might be alleviated by adopting fast rule match algorithms [Doorenbos et al. 92], such as RETE [Forgy 82] and its more efficient variations. However, if we take the space cost into account, it is still prohibitive to keep a set of the database abstractions that is about the same size or larger than the database just for efficient retrieval.

The utility of a rule is defined as follows by [Minton 88]:

$$Utility = AverageSaving \times ApplicationFrequency - AverageMatchCost$$

We can measure the utility of learned rules as follows. The $AverageMatchCost$ is proportional to the syntactic length of the rule. The application frequency and average saving can both be computed from statistical information. When the number of the database abstractions exceeds some threshold, the system will measure the utility of database abstractions and delete those with low utility.

Another task of maintenance is to update the rules in the presence of updates of the database. This includes identifying the invalid rules and then modifying them. Given an invalid rule $A \rightarrow B$, we can compile the data instances that satisfy A to modify B. We can also keep the update frequency of each rules, and use this statistical information as a criterion in measuring utility of the rules. In this way, the set of surviving rules will gradually evolve to capture the semantics of the database. A more interesting approach to update rules is using an incremental learning algorithm. An inductive learning algorithm is considered incremental if it forms new hypothetical rules from inserted or modified data instances without reading the entire database again. This approach is particularly useful for very large-scale databases.

5 Related Work

Siegel [Siegel 88] proposed the first automatic rule derivation approach for the semantic query optimization. Their approach learns from the failure of the rule match in reformulating the example query. The rule match is then guided by a pre-defined fixed set of heuristics. The major difference between our approach and theirs is that our approach is data-driven, while theirs is heuristic-driven. We view the example query as a representation of the set of the data it retrieves, rather than a procedure or plan of retrieval. The semantics of the example queries is easier to be captured with the data it retrieves. From this point of view, we seek the missing rules in the databases. In Siegel's work, on the other hand, the example queries merely provide the templates for the instantiation of the heuristics. The heuristics may not reflect how an example query should be reformulated with regard to the database contents. Consequently, their approach is conservative and the learning may converge only to what the heuristics specified.

Compared to other work in knowledge discovery in databases [Piatetsky-Shapiro 91], our work is to improve the performance of the database retrieval, while most of the discovery task is for database applications, such as discovering classification trees. An interesting prospect of our learning problem is to investigate the usage of the learned database abstraction other than for reformulation. Our learning approach is unsupervised, but it does not simply rely on the *surface values* to acquire or cluster knowledge from databases. The example queries are required to guide the search. Because queries contain the information of users' interests, they may help in providing semantic and functional properties of the data. Using example queries as the background knowledge to guide the knowledge discovery in databases appears to be a promising research direction.

Our learning problem distinguishes itself with other learning paradigms for problem solving in an important aspect. That is, the problem solving is performed in a *reformulate-then-solve* manner, and the learning is to supplement the knowledge required for reformulation. The reformulation is then used to speed up the problem solving (query execution). This manner of "speed-up" is quite different from the explanation-based approaches, such as, chunking in SOAR [Laird et al. 86] and Prodigy/EBL [Minton et al. 89]. In both approaches, the problem is not reformulated or changed, what these systems change is the problem solving strategy. The reformulation-then-solve approach changes the problem statement instead of the problem solving strategy. In our approach, we even try to reformulate the problem (query) to fit the problem solver (query execution unit). ALPINE [Knoblock 90], the abstraction learner of Prodigy, is another example of learning for reformulation to improve performance of problem solving. ALPINE learns to construct the abstraction levels of the problem search space. When given a problem, ALPINE reformulates the problem into subproblems of abstraction levels to reduce the cost. Although our approach also learns the abstraction knowledge from databases, it does not decompose the query into abstraction levels.

6 Conclusion

This paper presents an automatic learning approach to discover useful database abstractions for query reformulation. This approach uses the queries that are actually used in applications to classify the instances in the database. The system then derives an alternative query from the database inductively. This alternative query and the given query form the preliminary database abstraction. The system deduces and learns the rules from this preliminary database abstraction. The significance of this approach is that the query represents the data it retrieves, and the data represents a class of equivalent queries. This point of view leads to the learning approach that does not involve heuristics of specific database structure and implementation. Because the knowledge required for learning is available for almost any database (the data, and database schema), the dependence of the specific database structure and the domain expert's knowledge is minimized. This feature is particularly useful for the application in heterogeneous multidatabase systems. We believe this will make the query reformulation a feasible and effective approach in real-world database applications.

We have briefly shown how query reformulation approach can reduce the cost of multidatabase queries in Section 2. An important issue for future work is to develop an algorithm to include the multidatabase usage patterns to guide the learning. We plan to interleave the query planning, execution and reformulation to support learning. Another issue for future work is to develop algorithms for maintaining the validity of the database abstractions when the database is modified. We plan to establish dependency links between the database and the database abstractions so that the invalid rules will be identified quickly. We will also consider using an incremental algorithm to update rules.

Acknowledgements

We would like to thank the SIMS project leader Yigal Arens and the anonymous reviewers for their comments and suggestions on earlier drafts of this paper. Chin Y. Chee, another member of the SIMS project, is responsible for much of the programming that has gone into SIMS. Thanks also to the LIM project for providing us with the databases and the queries used for our work, as well as in the examples.

References

[Apers et al. 83] P.M.G. Apers, A.R. Hevner and S.B. Yao. Optimizing algorithms for distributed queries. *IEEE Trans. on Software Engineering*, 9:57–68, 1983.

[Arens et al. 93] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 1993. In press.

[Arens and Knoblock 92] Yigal Arens and Craig A. Knoblock. Planning and reformulating queries for semantically-modeled multidatabase systems. In *Proceedings of the First International Conference on Information and Knowledge Management*, Baltimore, MD, 1992.

[Cai et al. 91] Yandong Cai, Nick Cercone, and Jiawei Han. Learning in relational databases: An attribute-oriented approach. *Computational Intelligence*, 7(3):119–132, 1991.

[Chakravarthy et al. 90] Upen S. Chakravarthy, John Grant, and Jack Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.

[Chivatal 79] V. Chivatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4, 1979.

[Cormen et al. 89] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction To Algorithms*. The MIT Press/McGraw-Hill Book Co., Cambridge, MA, 1989.

[Doorenbos et al. 92] Bob Doorenbos, Milind Tambe, and Allen Newell. Learning 10,000 chunks: What's it like out there? In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 830–836, San Jose, CA, 1992.

[Forgy 82] C.L. Forgy. RETE: A fast algorithm for the many pattern/many object pattern matching problem. *Artificial Intelligence*, pages 17–37, 1982.

[Haussler 88] David Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[Hammer and Chan 76] M. Hammer and A. Chan. Index selection in a self-adaptive database management system. In *ACM-SIGMOD International Conference on Management of Data: Proceedings*, pages 1–8, Washington, DC, 1976.

[Hsu and Knoblock 93] Chun-Nan Hsu and Craig A. Knoblock. Reformulating query plans for multidatabase systems. Submitted to the Second International Conference of Information and Knowledge Management, 1993.

[Hammer and Niamir 79] M. Hammer and B. Niamir. A heuristic approach to attribute partitioning. In *ACM-SIGMOD International Conference on Management of Data: Proceedings*, pages 93–101, Boston, MA, 1979.

[Hammer and Zdonik 80] M. Hammer and S. B. Zdonik. Knowledge-based query processing. In *Proceedings of the Sixth VLDB Conference*, pages 137–146, Washington, DC, 1980.

[Jarke and Koch 84] M. Jarke and J. Koch. Query optimization in database systems. *ACM Computer Surveys*, 16:111–152, 1984.

[King 81] Jonathan Jay King. *Query Optimization by Semantic Reasoning.* PhD thesis, Stanford University, Department of Computer Science, 1981.

[Knoblock 90] Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 923–928, Boston, MA, 1990.

[Laird et al. 86] John E. Laird, Paul S. Rosenbloom, and Allen Newell. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, pages 11–46, 1986.

[Minton et al. 89] Steven Minton, Jaime G. Carbonell, Craig A. Knoblock, Daniel R. Kuokka, Oren Etzioni, and Yolanda Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40(1-3):63–118, 1989.

[Michalski 83] Ryszard S. Michalski. A theory and methodology of inductive learning. In *Machine Learning: An Artificial Intelligence Approach*, volume I, pages 83–134. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1983.

[Minton 88] Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* PhD thesis, Computer Science Department, Carnegie Mellon University, 1988.

[Piatetsky-Shapiro 84] Gregory Piatetsky-Shapiro. *A Self-Organizing Database System – A Different Approach To Query Optimization.* PhD thesis, Department of Computer Science, New York University, 1984.

[Piatetsky-Shapiro 91] G. Piatetsky-Shapiro. *Knowledge Discovery in Databases.* MIT Press, Cambridge, MA, 1991.

[Siegel 88] Michael D. Siegel. Automatic rule derivation for semantic query optimization. In Larry Kerschberg, editor, *Proceedings of the Second International Conference on Expert Database Systems*, pages 371–385. George Mason Foundation, Fairfax, VA, 1988.

[Ullman 88] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems*, volume I. Computer Science Press, Palo Alto, CA, 1988.