Rule Induction for Semantic Query Optimization

Chun-Nan Hsu and **Craig A. Knoblock**
Information Sciences Institute and Department of Computer Science
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
{chunnan,knoblock}@isi.edu

Abstract

Semantic query optimization can dramatically speed up database query answering by knowledge intensive reformulation. But the problem of how to learn required semantic rules has not previously been solved. This paper describes an approach using an inductive learning algorithm to solve the problem. In our approach, learning is triggered by user queries and then the system induces semantic rules from the information in databases. The inductive learning algorithm used in this approach can select an appropriate set of relevant attributes from a potentially huge number of attributes in real-world databases. Experimental results demonstrate that this approach can learn sufficient background knowledge to reformulate queries and provide a 57 percent average performance improvement.

1 INTRODUCTION

Speeding up a system's performance is one of the major goals of machine learning. Explanation-based learning is typically used for speedup learning, while applications of inductive learning are usually limited to data classifiers. In this paper, we present an approach in which inductively learned knowledge is used for semantic query optimization to speed up query answering for data/knowledge-based systems.

The principle of semantic query optimization (King 1981) is to use semantic rules, such as *all Tunisian seaports have railroad access*, to reformulate a query into a less expensive but equivalent query, so as to reduce the query evaluation cost. For example, suppose we have a query to *find all Tunisian seaports with railroad access and 2,000,000 ft^3 of storage space.* From the rule given above, we can reformulate the query so that there is no need to check the railroad access of seaports, which may save some execution time.

Many algorithms for semantic query optimization have been developed (Hsu & Knoblock 1993; King 1981; Shekhar, Srivastava, & Dutta 1988; Shenoy & Ozsoyoglu 1989). Average speedup ratios from 20 to 40 percent using hand-coded knowledge are reported in the literature. This approach to query optimization has gained increasing attention recently because it is applicable to almost all existing data/knowledge-base systems. This feature makes it particularly suitable for intelligent information servers connected to various types of remote information sources.

A critical issue of semantic query optimization is how to encode useful background knowledge for reformulation. Most of the previous work in semantic query optimization in the database community assume that the knowledge is given. (King 1981) proposed using *semantic integrity constraints* for reformulation to address the knowledge acquisition problem. Examples of semantic integrity constraints are *The salary of an employee is always less than his manager's*, and *Only female patients can be pregnant.* However, the integrity rules do not reflect properties of the contents of databases, such as related size of conceptual units, cardinality and distribution of attribute values. These properties determine the execution cost of a query. Moreover, integrity constraints rarely match query usage patterns. It is difficult to manually encode semantic rules that both reflect cost factors and match query usage patterns. The approach presented in this paper uses example queries to trigger the learning in order to match query usage patterns and uses an inductive learning algorithm to derive rules that reflect the actual contents of databases.

An important feature of our learning approach is that the inductive algorithm learns from complex real-world information sources. In these information sources, data objects are clustered into conceptual units. For example, the conceptual unit of a relational databases is a *relation*, or simply a *table*. For object-based databases, it is a *class*. In description-logic knowledge bases (Brachman & Schmolze 1985), data instances are clustered into *concepts*. Each conceptual unit has attributes that describe the relevant features. Most inductive learning systems, such as ID3, assume that relevant attributes are given. Consider a database with three relations: `car`, `person`, and `company`. We might want to characterize a class of persons by the company they work for, or by the cars they drive, or by the manufacturers of their cars. In these cases, we need attributes from different relations to describe a desired class of objects. Previous studies (Almuallim & Dietterich 1991; Russell 1989) have shown that the choice of *instance language bias* (i.e., selecting an appropriate set of attributes) is critical for the performance of an inductive learning system. To address this problem, we propose an inductive learning algorithm that can select attributes from different relations automatically.

The remainder of this paper is organized as follows. The next section illustrates the problem of semantic query optimization for data/knowledge bases. Section 3 presents an overview of the learning approach. Section 4 describes our inductive learning algorithm for structural data/knowledge bases. Section 5 shows the experimental results of using learned knowledge in reformulation. Section 6 surveys related work. Section 7 reviews the contributions of the paper and describes some future work.

2 SEMANTIC QUERY OPTIMIZATION

Semantic query optimization is applicable to various types of database and knowledge base. Nevertheless, we chose the relational model to describe our approach because it is widely used in practice. The approach can be easily extended to other data models. In this paper, a *database* consists of a set of primitive relations. A *relation* is then a set of instances. Each *instance* is a vector of attribute values. The number of attributes is fixed for all instances in a relation. The values of attributes can be either a number or a symbol, but with a fixed type. Below is an example database with two relations and their attributes:

```
geoloc(name,glc_cd,country,latitude,longitude),
seaport(name,glc_cd,storage,silo,crane,rail).
```

where the relation `geoloc` stores data about geographic locations, and the attribute `glc_cd` is a geographic location code.

The queries we are considering here are Horn-clause queries. A query always begins with the predicate `answer` and has the desired information as argument variables. For example,

```
Q1: answer(?name):-
       geoloc(?name,?glc_cd,"Malta",_,_),
       seaport(_,?glc_cd,?storage,_,_,_),
       ?storage > 1500000.
```

retrieves all geographical location names in Malta. There are two types of literals. The first type corresponds to a relation stored in a database. The second type consists of built-in predicates, such as `>` and `member`. Sometimes they are referred to as *extensional* and *intentional* relations, respectively (see (Ullman 1988)). We do not consider negative literals and recursion in this paper.

Semantic rules for query optimization are also expressed in terms of Horn-clause rules. Semantic rules must be consistent with the contents of a database. To clearly distinguish a rule from a query, we show queries using the Prolog syntax and semantic rules in a standard logic notation. A set of example rules are shown as follows:

```
R1: geoloc(_,_,"Malta",?latitude,_)
    ⇒ ?latitude ≥ 35.89.

R2: geoloc(_,?glc_cd,"Malta",_,_)
    ⇒ seaport(_,?glc_cd,_,_,_,_).

R3: seaport(_,?glc_cd,?storage,_,_,_) ∧
    geoloc(_,?glc_cd,"Malta",_,_)
    ⇒ ?storage > 2000000.
```

Rule `R1` states that the latitude of a Maltese geographic location is greater than or equal to 35.89. `R2` states that all Maltese geographic locations *in the database* are seaports. `R3` states that all Maltese seaports have storage capacity greater than 2,000,000 ft^3. Based on these rules, we can infer five equivalent queries of `Q1`. Three of them are:

```
Q21: answer(?name):-
       geoloc(?name,?glc_cd,"Malta",_,_),
       seaport(_,?glc_cd,_,_,_,_).

Q22: answer(?name):-
       geoloc(?name,_,"Malta",_,_).

Q23: answer(?name):-
       geoloc(?name,_,"Malta",?latitude,_),
       ?latitude ≤ 35.89.
```

`Q21` is deduced from `Q1` and `R3`. This is an example of *constraint deletion* reformulation. From `R2`, we can delete one more literal on `seaport` and infer that `Q22` is also equivalent to `Q1`. In addition to deleting constraints, we can also add constraints to a query based on rules. For example, we can add a constraint on `?latitude` to `Q22` from `R1`, and the resulting query `Q23` is still equivalent to `Q1`. Sometimes, the system can infer that a query is unsatisfiable because it contradicts a rule (or a chain of rules). It is also possible for the system to infer the answer directly from the rules. In both cases, there is no need to access the database to answer the query, and we can achieve nearly 100 percent savings.

Now that the system can reformulate a query into equivalent queries based on the semantic rules, the next problem is how to select the equivalent query with the lowest cost. The shortest equivalent query is not always the least expensive. The exact execution cost of a query depends on the physical implementation and the contents of the data/knowledge bases. However, we can usually estimate an approximate cost from the database schema and relation sizes. In our example, assume that the relation `geoloc` is very large and is sorted only on `glc_cd`, and assume that the relation `seaport` is small. Executing the shortest query `Q22` requires scanning the entire set of `geoloc` relations and is thus even more expensive than executing the query `Q1`. The cost to evaluate `Q21` will be less expensive than `Q1` and other equivalent queries because a redundant constraint on `?storage` is deleted, and the system can still use the sorted attribute `glc_cd` to locate the answers efficiently. Therefore, the system will select `Q21`.

Although the number of equivalent queries grows combinatorially with the number of applicable rules, semantic query optimization can be computed without explicitly searching this huge space. We have developed an efficient reformulation algorithm that is polynomial in terms of the number of applicable rules. We also extended this algorithm to reformulate multi-database query access plans and showed that the reformulations produce substantial performance improvements (Hsu & Knoblock 1993).

We conclude this section with the following observations on semantic query optimization.

1. Semantic query optimization can reduce query execution cost substantially.
2. Semantic query optimization is not a tautological transformation from the given query; it requires nontrivial, domain-specific background knowledge. Learning useful background knowledge is critical.
3. Since the execution cost of a query is dependent on the properties of the contents of information sources being queried, the utility of a semantic rule is also dependent on these properties.
4. The overhead of reformulation is determined by the number of applicable rules. Therefore, the utility problem (Minton 1988) is likely to arise and the learning must be selective.

3 OVERVIEW OF THE LEARNING APPROACH

This section presents an overview of our learning approach to address the knowledge acquisition problem of semantic query optimization. The key idea of our learning approach is that we view a query as a logical description (conjunction of constraints) of the answer, which is a set of instances satisfying the query. With an appropriate bias, an inductive learner can derive an equivalent query that is less expensive to evaluate than the original. Based on this idea, the learning is triggered by an example query that is expensive to evaluate. The system then inductively constructs a less expensive equivalent query from the data in the databases. Once this equivalent query is learned, the system compares the input query and constructed query, and infers a set of semantic rules for future use.

Figure 1 illustrates a simple scenario of this learning approach. An example query is given to a small database table with 3 instances. Evaluating this query will return an instance, which is marked with a `‘‘+’’` sign. Conceptually, instances in this table are labeled by the query as positive (answers) or negative (non-answers). We can use the inductive learning algorithm to generate an equivalent *alternative query* with appropriate biases so that the generated query is less expensive to evaluate. The generated alternative query should be satisfied by all answer instances and none of the others. This guarantees the equivalence of the two queries with regard to the current status of the data/knowledge base. Suppose that in this simple database, a short query is always less expensive to execute. The system will bias the learning in favor of the shortest description and inductively learn an alternative query `(A1 = ‘Z’)`. The inductive learning algorithm will be discussed in the next section.

The equivalence of the alternative query and the example query provides a *training example* of reformulation. In other words, this training example shows which equivalent query an input query should be reformulated into. The operationalization component will deduce a set of rules from the training example. This process consists of two stages. In the first stage, the system uses a logical inference procedure to transform the training example into the required syntax (Horn clauses). This syntax is designed so that the query reformulation can be computed efficiently. The equivalence between the two queries is converted to two implication rules:

$$(1)(A2 \leq 0) \wedge (A3 = 2) \Longrightarrow (A1 = `Z')$$

$$(2)(A1 = `Z') \Longrightarrow (A2 \leq 0) \wedge (A3 = 2)$$

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

$$(3)(A1 = `Z') \Longrightarrow (A2 \leq 0)$$

$$(4)(A1 = `Z') \Longrightarrow (A3 = 2)$$

After the transformation, we have proposed rules (1), (3), and (4) that satisfy our syntax criterion. In the second stage, the system tries to compress the antecedents of rules to reduce their match costs. In our example, rules (3) and (4) contain only one literal as antecedent, so no further compression is necessary. These rules are then returned immediately and learned by the system.

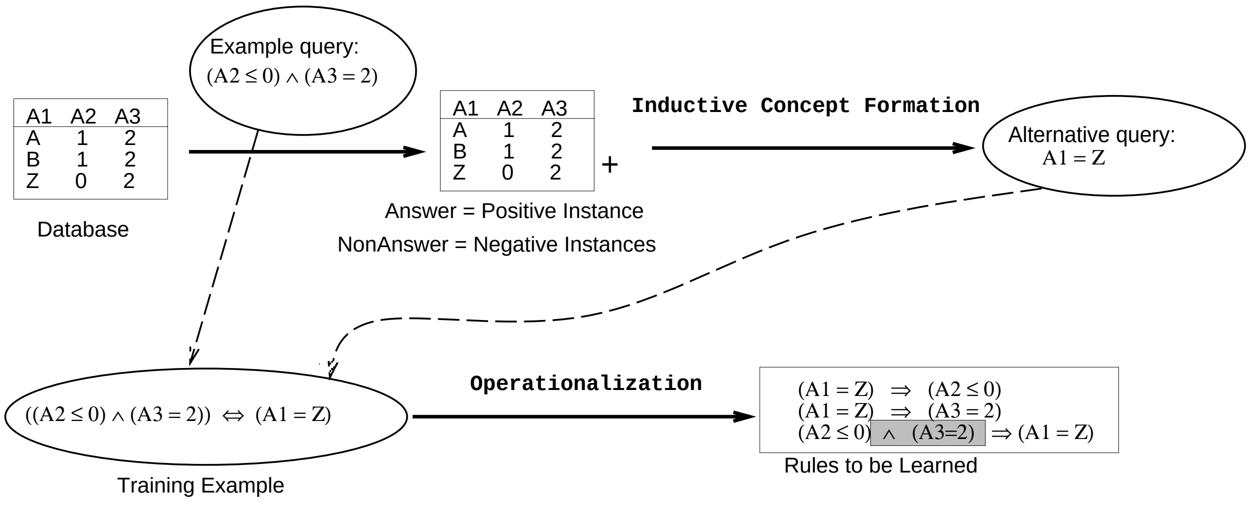


Figure 1: An Simplified Example Learning Scenario

If the proposed rule has more than one antecedent literal, such as rule (1), then the system can use the *greedy minimum set cover* algorithm (Cormen, Leiserson, & Rivest 1989) to eliminate unnecessary constraints. The problem of minimum set cover is to find a subset from a given collection of sets such that the union of the sets in the subset is equal to the union of all sets. We rewrite rule (1) as

$$(5)\neg(A1 = `Z') \Longrightarrow \neg(A2 \leq 0) \vee \neg(A3 = 2).$$

The problem of compressing rule (1) is thus reduced to the following: given a collection of sets of data instances that satisfy $\neg$(`A2` $\leq$ `0`) $\vee$ $\neg$(`A3 = 2`), find the minimum number of sets that cover the set of data instances that satisfy $\neg$(`A1 = 'Z'`). Since the resulting minimum set that covers $\neg$(`A1 = 'Z'`) is $\neg$(`A2` $\leq$ `0`), we can eliminate $\neg$(`A3 = 2`) from rule (5) and negate both sides to form the rule

$$(A2 \leq 0) \Longrightarrow (A1 = `Z').$$

4 LEARNING ALTERNATIVE QUERIES

The scenario shown in Figure 1 is a simplified example where the database consists of only one table. However, real-world databases and knowledge bases usually decompose their application domain into multiple conceptual units. One could try to combine every conceptual unit that could be relevant into a large table, then apply the learning system for tabular databases directly. However, learning from a large table is too expensive computationally. Such an approach will not work unless a small number of relevant attributes are correctly identified before learning.

In this section, we discuss inductive learning for Horn-clause queries from a database with multiple relations. Our learning problem is to find an alternative query to characterize a class of instances defined in a relation. In standard machine learning terms, this subset of instances are labeled as positive examples, and the others are negative examples.

Before we discuss the algorithm, we need to clarify two forms of constraints implicitly expressed in a query. One form is an *internal disjunction*, a set of disjunctions on the values of an attribute. For example, an instance of `geoloc` satisfies:

```
C1:geoloc(?name,_,?cty,_,_),
   member(?cty,["Tunisia","Italy","Libya"]).
```

iff its `?cty` value is `"Tunisia"`, `'"Italy"`, or `"Libya"`. The other form is a *join constraint*, which combines instances from two relations. For example, a pair of instances of `geoloc` and `seaport` satisfy a join constraint:

```
C2:geoloc(?name1,?glc_cd,_,_,_),
   seaport(?name2,?glc_cd,_,_,_,_).
```

iff they share common values on the attribute `glc_cd` (geographic location code).

Our inductive learning algorithm is extended from the greedy algorithm that learns internal disjunctions proposed by (Haussler 1988). Of the many inductive learning algorithms, Haussler's was chosen because its hypothesis description language is the most similar to ours. His algorithm starts from an empty hypothesis of the target concept description to be learned. The algorithm proceeds by constructing a set of candidate constraints that are consistent with all positive examples, and then using a *gain/cost* ratio as the heuristic function to select and add candidates to the hypothesis. This process of candidate construction and selection is repeated until no negative instance satisfies the hypothesis.

We extended Haussler's algorithm to allow join constraints in the target description. To achieve this, we extended the candidate construction step to allow join constraints to be considered, and we extended the heuristic function to evaluate both internal disjunctions and join constraints. Also, we adopted an approach to searching the space of candidate constraints that restricts the size of the space.

4.1 CONSTRUCTING AND EVALUATING CANDIDATE CONSTRAINTS

In this subsection, we describe how to construct a candidate constraint, which can be either an internal disjunction or a join constraint. Then we describe a method for evaluating both internal disjunctions and join constraints. Given a relation partitioned into positive and negative instances, we can construct an internal disjunctive constraint for each attribute by generalizing attribute values of positive instances. The constructed constraint is consistent with positive instances because it is satisfied by all positive instances. Similarly, we can construct a join constraint consistent with positive instances by testing whether all positive instances satisfy the join constraint. The constructed constraints are candidates to be selected by the system to form an alternative query.

For example, suppose we have a database that contains the instances as shown in Figure 2. In this database, instances labeled with `''+''` are positive instances. Suppose the system is testing whether join constraint `C2` is consistent with the positive instances. Since for all positive instances, there is a corresponding instance in `seaport` with a common `glc_cd` value, the join constraint `C2` is consistent and is considered as a candidate constraint.

Once we have constructed a set of candidate internal disjunctive constraints and join constraints, we need to measure which one is the most promising and add it to the hypothesis. In Haussler's algorithm, the measuring function is a *gain/cost* ratio, where *gain* is defined as the number of negative instances excluded and *cost* is defined as the syntactic length of a constraint. This heuristic is based on the generalized problem of minimum set cover where each set is assigned a constant cost. Haussler used this heuristic to bias the learning for short hypotheses. In our problem, we want the system to learn a query expression with the least cost. In real databases, sometimes additional constraints can reduce query evaluation cost. So we keep the *gain* part of the heuristic, while defining the *cost* of the function as the estimated evaluation cost of the constraint by a database system.

The motivation of this formula is also from the generalized minimum set covering problem. The gain/cost heuristic has been proved to generate a set cover within a small ratio bound ($\ln|n|+1$) of the optimal set covering cost (Chvatal 1979), where n is the number of input sets. However, in this problem, the cost of a set is a constant and the total cost of the entire set covers is the sum of the cost of each set. This is not always the case for database query execution, where the cost of each constraint is dependent on the execution ordering. To estimate the actual cost of a constraint is very expensive. We therefore use an approximation heuristic here.

The evaluation cost of individual constraints can be estimated using standard database query optimization techniques (Ullman 1988) as follows. Let $\mathcal{D}_1$ denote the constraining relation, and $|\mathcal{D}_1|$ denote the size of a relation, then the evaluation cost for an internal disjunctive constraint is proportional to

$$|\mathcal{D}_1|$$

because for an internal disjunction on an attribute that is not indexed, a query evaluator has to scan the entire database to find all satisfying instances. If the internal disjunction is on an indexed attribute, then the cost should be proportional to the number of instances satisfying the constraint. In both cases, the system can always sample the database query evaluator to obtain accurate execution costs.

For join constraints, let $\mathcal{D}_2$ denote the new relations introduced by a join constraint, and $\mathcal{I}_1, \mathcal{I}_2$ denote the *cardinality* of join attributes of two relations, that is, the number of distinct values of attributes over which $\mathcal{D}_1$ and $\mathcal{D}_2$ join. Then the evaluation cost for the join over $\mathcal{D}_1$ and $\mathcal{D}_2$ is proportional to

$$|\mathcal{D}_1| * |\mathcal{D}_2|$$

when the join is over attributes that are not indexed, because the query evaluator must compute a cross product to locate pairs of satisfying instances. If the join is over indexed attributes, the evaluation cost is proportional to the number of instance pairs returned from the join, that is,

$$\frac{|\mathcal{D}_1| * |\mathcal{D}_2|}{max(\mathcal{I}_1, \mathcal{I}_2)}.$$

This estimate assumes that distinct attribute values distribute uniformly in instances of joined relations. Again, if possible, the system can sample the database for more accurate execution costs. For the above example problem, we have two candidate constraints that are the most promising:

```
C3:geoloc(?name,_,"Malta",_,_).

C4:geoloc(?name,?glc_cd,_,_,_),
   seaport(_,?glc_cd,_,_,_,_).
```

Suppose `|geoloc|` is 300, and `|seaport|` is 8. Cardinality of `glc_cd` for `geoloc` is 300 again, and for `seaport` is 8. Suppose both relations have indices on `glc_cd`. Then the evaluation cost of `C3` is 300, and `C4` is $300 * 8/300 = 8$. The gain of `C3` is $300 - 4 = 296$,

```
geoloc("Safaqis",    8001, Tunisia, ...)     seaport("Marsaxlokk"   8003 ...)
geoloc("Valletta",   8002, Malta,   ...)+    seaport("Grand Harbor" 8002 ...)
geoloc("Marsaxlokk", 8003, Malta,   ...)+    seaport("Marsa"        8005 ...)
geoloc("San Pawl",   8004, Malta,   ...)+    seaport("St Pauls Bay" 8004 ...)
geoloc("Marsalforn", 8005, Malta,   ...)+    seaport("Catania"      8016 ...)
geoloc("Abano",      8006, Italy,   ...)     seaport("Palermo"      8012 ...)
geoloc("Torino",     8007, Italy,   ...)     seaport("Traparri"     8015 ...)
geoloc("Venezia",    8008, Italy,   ...)     seaport("AbuKamash"    8017 ...)
           ⋮              ⋮
```

Figure 2: The Database Fragment

and the gain of C4 is $300 - 8 = 292$, because only 4 instances satisfy C3 while 8 instances satisfy C4. (There are 8 seaports, and all have a corresponding geoloc instance.) So the gain/cost ratio of C3 is $296/300 = 0.98$, and the gain/cost ratio of C4 is $292/8 = 36.50$. The system will select C4 and add it to the hypothesis.

4.2 SEARCHING THE SPACE OF CANDIDATE CONSTRAINTS

When a join constraint is selected; a new relation and its attributes are introduced to the search space of candidate constraints. The system can consider adding constraints on attributes of the newly introduced relation to the partially constructed hypothesis. In our example, a new relation seaport is introduced to describe the positive instances in geoloc. The search space is now expanded into two layers, as illustrated in Figure 3. The expanded constraints include a set of internal disjunctions on attributes of seaport, as well as join constraints from seaport to another relation. If a new join constraint has the maximum gain/cost ratio and is selected later, the search space will be expanded further. Figure 3 shows the situation when a new relation, say channel, is selected, the search space will be expanded one layer deeper. At this moment, candidate constraints will include all unselected internal disjunctions on attributes of geoloc, seaport, and channel, as well as all possible joins with new relations from geoloc, seaport and channel. Exhaustively evaluating the gain/cost of all candidate constraints is impractical when learning from a large and complex database.

We adopt a search method that favors candidate constraints on attributes of newly introduced relations. That is, when a join constraint is selected, the system will estimate only those candidate constraints in the newly expanded layer, until the system constructs a hypothesis that excludes all negative instances (i.e., reaches the goal) or no more consistent constraints in the layer with positive gain are found. In the later case, the system will backtrack to search the remaining constraints on previous layers. This search control bias takes advantage of underlying domain knowledge in the schema design of databases. A join constraint is unlikely to be selected on average, because an internal disjunction is usually much less expensive than a join. Once a join constraint (and thus a new relation) is selected, this is strong evidence that all useful internal disjunctions in the current layer have been selected, and it is more likely that useful candidate constraints are on attributes of newly joined relations. This bias works well in our experiments. But certainly there are cases when this search heuristic prunes out useful candidate constraints. Another way to bias the search is by including prior knowledge for learning. In fact, it is quite natural to include prior knowledge in our algorithm, and we will discuss this later.

Returning to the example, since C4 was selected, the system will expand the search space by constructing consistent internal disjunctions and join constraints on seaport. Assuming that the system cannot find any candidate on seaport with positive gain, it will backtrack to consider constraints on geoloc again. Next, the constraint on country is selected (see Figure 3) and all negative instances are excluded. The system thus learns the query:

```
Q3: answer(?name):-
       geoloc(?name,?glc_cd,"Malta",_,_),
       seaport(_,?glc_cd,_,_,_,_).
```

The operationalization component will then take Q1 and this learned query Q3 as a training example for reformulation,

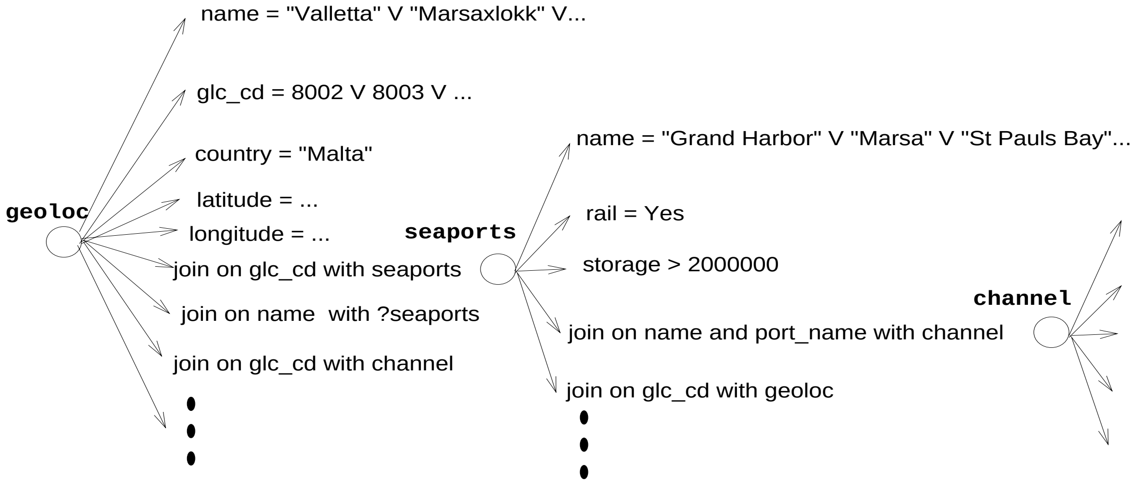
```
  geoloc(?name,_,"Malta",_,_)
  ⇔ geoloc(?name,?glc_cd,"Malta",_,_) ∧
     seaport(_,?glc_cd,_,_,_,_).
```

and deduce a new rule to reformulate Q1 to Q3:

```
  geoloc(_,?glc_cd,"Malta",_,_)
  ⇒ seaport(_,?glc_cd,_,_,_,_).
```

This is the rule R2 we have seen in Section 2. Since the size of geoloc is considerably larger than that of seaport, next time when a query asks about geographic locations in Malta, the system can reformulate the query to access the seaport relation instead and speed up the query answering process.

The algorithm can be further enhanced by including prior knowledge to reduce the search space. The idea is to use prior knowledge, such as *determinations* pro-



geoloc
name = "Valletta" V "Marsaxlokk" V...
glc_cd = 8002 V 8003 V ...
country = "Malta"
latitude = ...
longitude = ...
join on glc_cd with seaports
join on name with ?seaports
join on glc_cd with channel
seaports
name = "Grand Harbor" V "Marsa" V "St Pauls Bay"...
rail = Yes
storage > 2000000
join on name and port_name with channel
join on glc_cd with geoloc
channel

Figure 3: Candidate Constraints to be Selected

posed by (Russell 1989), to sort candidate constraints by their comparative relevance, and then test their gain/cost ratio in this sorted order. For example, assuming that from its prior knowledge the system knows that the constraints on attributes `latitude` and `longitude` of `geoloc` are unlikely to be relevant, then the system can ignore them and evaluate candidate constraints on the other attributes first. If the prior knowledge is correct, the system will construct a consistent hypothesis with irrelevant constraints being pruned from the search space. However, if the system cannot find a constraint that has a positive gain, then the prior knowledge may be wrong, and the system can backtrack to consider "irrelevant" constraints and try to construct a hypothesis from them. In this way, the system can tolerate incorrect and incomplete prior knowledge. This usage of prior knowledge follows the general spirit of FOCL (Pazzani & Kibler 1992).

5 Experimental Results

Our experiments are performed on the SIMS knowledge-based information server (Arens *et al.* 1993; Hsu & Knoblock 1993). SIMS allows users to access different kinds of remote databases and knowledge bases as if they were using a single system. For the purpose of our experiments, SIMS is connected with three remotely distributed Oracle databases via the Internet. Table 1 shows the domain of the contents and the sizes of these databases. We had 34 sample queries written by users of the databases for the experiments. We classified these queries into 8 categories according to the relations and constraints used in the queries. We then chose 8 queries randomly from each category as input to the learning system and generated 32 rules. These rules were used to reformulate the remaining 26 queries. In addition to learned rules, the system also used 163 attribute range facts (e.g., the range of the `storage` attribute of `seaport` is between 0 and 100,000) compiled from the databases. Range facts are useful for numerically typed attributes in the rule matching.

Table 1: Database Features

DBs	Contents	Relations	Instances	Size (MB)
Geo	Geographical locations	16	56708	10.48
Assets	Air and sea assets	14	5728	0.51
Fmlib	Force module library	8	3528	1.05

The performance statistics for query reformulation are shown in Table 2. In the first column, we show the average performance of all tested queries. We divide the queries into 3 groups. The number of queries in each group is shown in the first row. The first group contains those unsatisfiable queries refuted by the learned knowledge. In these cases, the reformulation takes full advantage of the learned knowledge and the system does not need to access the databases at all, so we separate them from the other cases. The second group contains those low-cost queries that take less than one minute to evaluate without reformulation. The last group contains the high-cost queries.

The second row lists the average elapsed time of query execution without reformulation. The third row shows the average elapsed time of reformulation and execution. Elapsed time is the total query processing time, from receiving a query to displaying all answers. To reduce inaccuracy due to the random latency time in network transmission, all elapsed time data are obtained by executing each query 10 times and then computing the average. The reformulation yields significant cost reduction for high-cost queries. The overall average gain is 57.10 percent, which is better than systems using hand-coded rules for semantic optimization (Hsu & Knoblock 1993; Shekhar, Srivastava, & Dutta 1988; Shenoy & Ozsoyoglu 1989). The gains are not so high for the low-cost group. This is not unexpected, because the queries in this group are already very cheap and the cost cannot be reduced much further. The average overheads listed in the table show the time in seconds used in reformulation. This overhead is very small compared to the total query processing time. On average, the system fires rules 5 times for reformulation. Note that the same rule may be fired more than once during the reformulation procedure (see (Hsu & Knoblock 1993) for more detailed descriptions).

Table 2: Performance Statistics

	All	Answer inferred	$\leq$ 60s.	$>$ 60s.
# of queries	26	4	17	5
No reformulation	54.27	44.58	10.11	212.21
Reformulation	23.28	5.45	8.79	86.78
Time saved	30.99	39.14	1.31	125.46
% Gain of total elapsed time	57.1%	87.8%	12.9%	59.1%
Average overhead	0.08	0.07	0.07	0.11
Times rule fired	5.00	6.00	4.18	7.00

6 RELATED WORK

Previously, two systems that learn background knowledge for semantic query optimization were proposed by (Siegel 1988) and by (Shekhar *et al.* 1993). Siegel's system uses predefined heuristics to drive learning by an example query. This approach is limited because the heuristics are unlikely to be comprehensive enough to detect missing rules for various queries and databases. Shekhar's system is a data-driven approach which assumes that a set of relevant attributes is given. Focusing on these relevant attributes, their system explores the contents of the database and generates a set of rules in the hope that all useful rules are learned. Siegel's system goes to one extreme by neglecting the importance of guiding the learning according to the contents of databases, while Shekhar's system goes to another extreme by neglecting dynamic query usage patterns. Our approach is more flexible because it addresses both aspects by using example queries to trigger the learning and using inductive learning over the contents of databases for semantic rules.

The problem of inductive learning from a database with multiple relations shares many issues with research work in inductive logic programming (ILP) (Muggleton et al. 1994), especially the issue of when to introduce new relations. The main difference between our approach and ILP is that we also consider the cost of the learned concept description. Our system currently learns only single-clause, non-recursive queries, while ILP approaches can learn multi-clause and recursive rules. However, due to the complexity of the problem, most of the existing ILP approaches do not scale up well to learn from large, real-world data/knowledge-bases containing more than ten relations with thousands of instances. Our approach can learn from large databases because it also uses the knowledge underlying the database design.

Tan's cost-sensitive learning (Tan 1993) is an inductive learning algorithm that also takes the cost of the learned description into account. His algorithm tries to learn minimum-cost decision trees from examples in a robot object-recognition domain. The algorithm selects a minimum number of attributes to construct a decision tree for recognition. The attributes are selected in the order of their evaluation cost. When constructing a decision tree, it uses a heuristic attribute selection function I^2/C, where I is the information gain defined as in ID3, and C is the cost to evaluate a given attribute. This function is similar to our function $gain/evaluation\_cost$. While there is no theoretic analysis about the general performance of the heuristic I^2/C for decision-tree learning, our function is derived from approximation heuristics for minimum set cover problems. (Nunez 1991) defined another similar heuristic $(2^I - 1)/C$ for cost-sensitive decision-tree learning. His paper provides an information-theoretic motivation of the heuristic.

(Cai, Cercone, & Han 1991) present an attribute-oriented learning approach designed to learn from relational databases. The approach learns conjunctive rules by generalizing instances of a single relation. The generalization operations include replacing attribute values with the least common ancestors in a value hierarchy, removing inconsistent attributes, and removing duplicate instances. In contrast to our inductive learning algorithm, this attribute-oriented approach requires users to select relevant attributes before learning can be performed.

The operationalization component in our learning approach can be enhanced with an EBL-like explainer to filter out low utility rules and generalize rules. A similar "induction-first then EBL" approach can be found in (Shen 1992). Shen's system uses general heuristics to guide the inductive learning for regularities expressed in a rule template $P(x,y) \wedge R(y,z) \Rightarrow Q(x,z)$. Our system has a definite goal, so we use example queries to guide the learning and do not restrict the format of learned rules to a specific template.

7 Conclusions and Future Work

This paper demonstrates that the knowledge required for semantic query optimization can be learned inductively under the guidance of example queries. We have described a general approach in which inductive learning is triggered by example queries, and an algorithm to learn from a database with multiple relations. Experimental results show that query reformulation using learned background knowledge produces substantial cost reductions for a real-world intelligent information server.

In future work, we plan to experiment with different ways of selecting example queries for training, and to develop an effective approach to using prior knowledge for constraining searches in the inductive learning algorithm. We also plan to enhance the operationalization component so that the system can be more selective and thus avoid the utility problem.

A limitation to our approach is that there is no mechanism to deal with changes to data/knowledge bases. There are three possible alternatives to address this problem. First, the system can simply remove the invalid rules due to the update and let the system learn from future queries after the update. Second, the system can predict the expected utility of each rule, and choose to update or re-learn a subset of invalid rules. Third, the system can update or re-learn all rules after the update. We plan to experiment with all of these alternatives and propose an approach to let the system decide which update alternative is the most appropriate for an expected model of database change.

Acknowledgements

We wish to thank Yigal Arens and Chin Y. Chee for their help on this work. Thanks also to the anonymous reviewers for their valuable comments on earlier drafts of this paper. The research reported here was supported in part by the National Science Foundation under grant No. IRI-9313993 and in part by Rome Laboratory of the Air Force Systems Command and the Advanced Research Projects Agency under Contract No. F30602-91-C-0081.

References

Almuallim, H., and Dietterich, T. G. (1991). Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 547–552.

Arens, Y.; Chee, C. Y.; Hsu, C.-N.; and Knoblock, C. A. (1993). Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems* 2(2):127–159.

Brachman, R., and Schmolze, J. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science* 9(2):171–216.

Cai, Y.; Cercone, N.; and Han, J. (1991). Learning in relational databases: An attribute-oriented approach. *Computational Intelligence* 7(3):119–132.

Chvatal, V. (1979). A greedy heuristic for the set covering problem. *Mathematics of Operations Research* 4.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. (1989). *Introduction To Algorithms*. Cambridge, MA: The MIT Press/McGraw-Hill Book Co.

Haussler, D. (1988). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence* 36:177–221.

Hsu, C.-N., and Knoblock, C. A. (1993). Reformulating query plans for multidatabase systems. In *Proceedings of the Second International Conference on Information and Knowledge Management*.

King, J. J. (1981). *Query Optimization by Semantic Reasoning*. Ph.D. Dissertation, Stanford University, Department of Computer Science.

Minton, S. (1988). *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Ph.D. Dissertation, Carnegie Mellon University, School of Computer Science.

Muggleton, S. editor. (1994). Special section on inductive logic programming. *SIGART Bulletin* 5(1):5–49.

Nunez, M. (1991). The use of background knowledge in decision tree induction. *Machine Learning* 6:231–250.

Pazzani, M. J., and Kibler, D. (1992). The utility of knowledge in inductive learning. *Machine Learning* 9:57–94.

Russell, S. J. (1989). *The Use of Knowledge in Analogy and Induction*. San Mateo, CA: Morgan Kaufmann.

Shekhar, S.; Hamidzadeh, B.; Kohli, A.; and Coyle, M. (1993). Learning transformation rules for semantic query optimization: A data-driven approach. *IEEE Transactions on Knowledge and Data Engineering* 5(6):950–964.

Shekhar, S.; Srivastava, J.; and Dutta, S. (1988). A formal model of trade-off between optimization and execution costs in semantic query optimization. In *Proceedings of the 14th VLDB Conference*.

Shen, W.-M. (1992). Discovering regularities from knowledge bases. *International Journal of Intelligent Systems* 7:623–635.

Shenoy, S. T., and Ozsoyoglu, Z. M. (1989). Design and implementation of a semantic query optimizer. *IEEE Trans. Knowledge and Data Engineering* I(3):344–361.

Siegel, M. D. (1988). Automatic rule derivation for semantic query optimization. In Kerschberg, L., ed., *Proceedings of the Second International Conference on Expert Database Systems*. Fairfax, VA: George Mason Foundation. 371–385.

Tan, M. (1993). Cost-sensitive learning of classification knowledge and its application in robotics. *Machine Learning* 13:7–33.

Ullman, J. D. (1988). *Principles of Database and Knowledge-base Systems*, volume I,II. Palo Alto, CA: Computer Science Press.