To appear in *Advances of Knowledge Discovery and Data Mining*, AAAI Press, 1995

17 Using Inductive Learning To Generate Rules for Semantic Query Optimization

Chun-Nan Hsu and Craig A. Knoblock
Information Sciences Institute and
Department of Computer Science
University of Southern California

Abstract

Semantic query optimization can dramatically speed up database query answering by knowledge intensive reformulation. But the problem of how to learn the required semantic rules has not been previously solved. This chapter presents a learning approach to solving this problem. In our approach, the learning is triggered by user queries. Then the system uses an inductive learning algorithm to generate semantic rules. This inductive learning algorithm can automatically select useful join paths and attributes to construct rules from a database with many relations. The learned semantic rules are effective for optimization because they will match query patterns and reflect data regularities. Experimental results show that this approach learns sufficient rules for optimization that produces a substantial cost reduction.

17.1 Introduction

This chapter presents an approach to learning semantic knowledge for semantic query optimization (SQO). SQO optimizes a query by using semantic rules, such as *all Maltese seaports have railroad access*, to reformulate a query into a less expensive but equivalent query. For example, suppose we have a query to *find all Maltese 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 SQO algorithms have been developed (Hammer and Zdonik 1980; King 1981; Shekhar et al. 1988; Shenoy and Ozsoyoglu 1989). Average savings from 20 to 40 percent using hand-coded knowledge are reported in the literature.

A learning approach to automatic acquisition of semantic knowledge is crucial to SQO. Most of the previous work in SQO assumes that semantic knowledge is given. King (1981) proposed using *semantic integrity constraints* given by database programmers to address the knowledge acquisition problem. An example of semantic integrity constraints is that *Only female patients can be pregnant.* However, integrity constraints do not reflect properties of data that affect the query execution cost, such as, relation sizes and distributions of attribute values. Moreover, integrity constraints rarely match query patterns. It is difficult to encode semantic knowledge that both reflects database properties and matches query patterns. The approach presented in this chapter uses example queries to trigger the learning so as to match query patterns, and induces effective semantic rules that reflect regularities of data.

Unlike most rule mining systems (Agrawal et al. 1993; Mannila et el. 1994), which are designed to derive rules from a single database table, our inductive learning algorithm can learn semantic rules from a database with many relations. Consider a database with three relations: `person`, `car`, and `company`. An interesting rule about persons might involve the companies they work for, or the cars they drive, or even the manufacturers of their cars. Our inductive learning algorithm can select relevant join paths and attributes automatically instead of requiring users to do this difficult and tedious task. With semantic rules describing regularities of joined relations, the SQO optimizer will be more effective because it is able to delete a redundant join or introduce new joins in a query.

The remainder of this chapter is organized as follows. The next section illustrates the problem of semantic query optimization for databases. Section 17.3 presents an overview of the learning approach. Section 17.4 describes our inductive learning algorithm for databases with many relations. Section 17.5 shows the experimental results of using learned knowledge in optimization. Section 17.6 surveys related work. Section 17.7 reviews the contributions and describes some future work.

17.2 Semantic Query Optimization

Semantic query optimization is applicable to different types of databases. 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 chapter, a *database* consists of a set of 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 string, but with a fixed type. Figure 17.1 shows the schema of an example database with two relations and their attributes. In this database, the relation `geoloc` stores data about geographic locations, and the attribute `glc_cd` is a geographic location code.

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

Semantic Rules:
```
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.
```

Figure 17.1
Schema of a geographic database and semantic Rules

Although the basic SQO approach (King 1981) applies only to conjunctive queries, it can be extended to optimize complex queries with disjunctions, *group-by* or aggregate operators. The idea is that a complex query can be decomposed into conjunctive subqueries. The system can then apply SQO to optimize each subquery and propagate constraints among them for global optimization. We have developed such an SQO algorithm to optimize heterogeneous multidatabase query plans (Hsu and Knoblock 1993a). Rules learned for optimizing conjunctive queries can be used for optimizing complex queries. In this chapter, we will focus on the problem of learning and SQO for conjunctive queries.

The queries considered here are conjunctive Datalog queries, which corresponds to the select-from-where subset of SQL. A query begins with a predicate `answer`. There can be one or more arguments to `answer`. 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`.

Semantic rules for query optimization are expressed in terms of Horn clauses. Semantic rules must be consistent with the data. 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 also shown in Figure 17.1.

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 shown in Figure 17.2. `Q21` is deduced from `Q1` and `R3`. This is an example of *constraint deletion*

```
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.
```

Figure 17.2
Equivalent queries

reformulation. From `R2`, we can delete one more literals 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 the semantic rules. For example, we can add a constraint on `?latitude` to `Q22` from `R1`, and the resulting query `Q23` is still equivalent to `Q1`. Adding a new constraint could be useful when the new constraint is on an indexed attribute. 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 exact execution cost of a query depends on the physical implementation and the contents of the databases. 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 of evaluating `Q21` will be less than that of `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`.

The difference between conventional query optimization (Jarke and Koch 1984; Ullman 1988) and SQO is that the latter uses semantic knowledge to extend the search space. Conventional syntactic query optimization searches for low-cost queries logically equivalent to input queries. Optimization by reordering literals/constraints in a query belongs in this category. SQO, in addition, searches for low-cost queries equivalent to an input queries given some semantic knowledge. Therefore, its search space is much larger and the potential savings that can be achieved are also much larger than those from syntactic optimization alone.

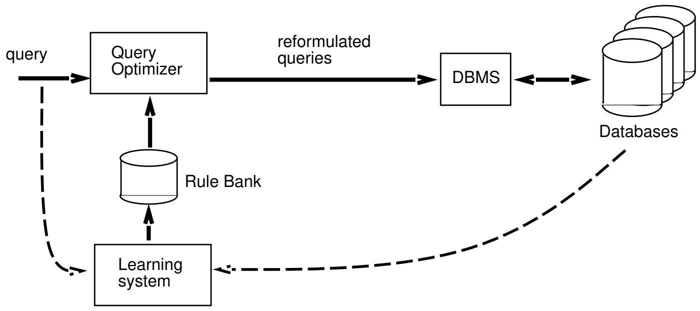

Figure 17.3
Structure of the database system with SQO optimizer and learner

17.3 General Learning Framework

This section presents a general learning framework for the learning problem of SQO. Figure 17.3 illustrates the organization of a database system with an SQO optimizer and a learning system. The optimizer uses semantic rules in a rule bank to optimize input queries, and then sends optimized queries to the DBMS to retrieve data. When the DBMS encounters an expensive input query, it triggers the learning system to learn a set of rules from the data, and then saves them in the rule bank. These rules will be used to optimize future queries. The system will gradually learn a set of effective rules for optimization.

Figure 17.4 illustrates a simplified scenario of our learning framework. This learning framework consists of two components, an inductive learning component, and an operationalization component. A query is given to trigger the learning. The system applies an inductive learning algorithm to induce an *alternative query* equivalent to the input query with a lower cost. The operationalization component then takes the input query and the learned alternative query to derive a set of semantic rules. Previously, Yu and Sun (1989) have shown that semantic rules for SQO can be derived from two equivalent queries. However, they did not show how to automatically generate equivalent queries. Our approach can automatically induce a low-cost alternative query of an expensive input query. The derived rules will thus match query patterns and be effective for SQO in reformulating expensive queries into low-cost equivalent queries.

In Figure 17.4, instances (or tuples) in the database are labeled as positive (+) if they satisfy the input query and negative (−) otherwise. The learned alternative query must cover all positive instances but no negative instances so that it retrieves the same data as the input query and is equivalent to the input query. Given a set of data instances classified as positive or negative, the problem of inducing a description that covers all

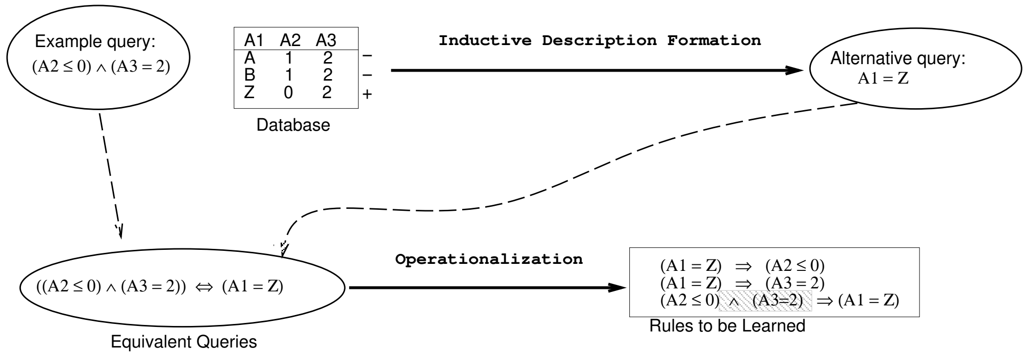

Figure 17.4
A simplified learning scenario

positive data instances but no negatives is known as *supervised inductive learning* in machine learning (Shavlik and Dietterich 1990). Since a query is a description of the data to be retrieved, inductive learning algorithms that learn descriptions expressed in the query language can be used in our framework.

Most supervised inductive learning algorithms are designed for accurate classification of unseen data instances. In our framework, however, the algorithm is also required to induce a low-cost description, that is, a low-cost alternative query that can be evaluated by the DBMS efficiently. Previously, we have developed an inductive learning algorithm that learns low-cost queries from single-table databases (Hsu and Knoblock 1993a). Section 17.4 describes in detail a more advanced algorithm that learns conjunctive Datalog queries from relational databases. This algorithm can be extended to databases with more advanced data models, such as object-oriented and deductive databases.

The operationalization component derives semantic rules from two equivalent queries. It consists of two stages. In the first stage, the system transforms the equivalence of two equivalent queries into the required syntax (Horn clauses) so that the optimizer can use semantic rules efficiently. For the example in Figure 17.4, the equivalence of the two queries is transformed into 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 the Horn-clause syntax requirement:

(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 requirement. 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. If the proposed rule has many antecedent literals, then the system can use the *greedy minimum set cover* algorithm (Coremen et al. 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. Negating both sides of (1) yields:

(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 that satisfy $\neg$(A2 $\leq$ 0) $\vee$ $\neg$(A3 = 2), find the minimum number of sets that cover the set of data satisfying $\neg$(A1 = 'Z'). Suppose 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 again to form the rule:

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

17.4 Learning Alternative Queries

The previous section has described a general learning framework and how the operationalization component derives rules from the equivalence of input and alternative queries. This section describes an inductive learning approach to learning low-cost alternative queries. The scenario shown in Figure 17.4 is a simplified example where the database consists of only one table. However, real-world databases usually have many relations, and users can specify joins to associate different relations in a query. The inductive learning approach described below can learn low-cost conjunctive Datalog queries from real-world databases with many relations.

Before we discuss the approach, we need to clarify two forms of constraints implicitly specified in a Datalog query. Consider the geographic database schema in Figure 17.1. Some example constraints for this database are shown in Figure 17.5. Among these constraints, C0 and C1 are *internal disjunctions*, which are constraints on the values of a single attribute. An instance of seaport satisfies C0 if its ?storage value is less than 150,000. An instance of geoloc satisfies C1 if its ?cty value is "Tunisia" or "Italy" or "Libya". The other form of constraint is a *join constraint*, which specifies a constraint on values of two or more attributes from different relations. A pair of instances of geoloc and seaport satisfy a join constraint C2 if 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 from a single-table database proposed by Haussler (1988). Of the many inductive learning algorithms, Haussler's was chosen because its hypothesis description language is the most similar to database query languages. His algorithm starts

```
C0:seaport(?name,_,?storage,_,_,_),
   ?storage <= 150000.
C1:geoloc(?name1,_,?cty,_,_),
   member(?cty,["Tunisia","Italy","Libya"]).
C2:geoloc(?name1,?glc_cd,_,_,_),
   seaport(?name2,?glc_cd,_,_,_,_).
```

Figure 17.5
Two forms of constraints used in queries

from an empty hypothesis of the concept description to be learned. The algorithm proceeds by constructing a set of *candidate constraints* that are consistent with all positive instances, 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 instances satisfies the hypothesis.

The top level algorithm of our inductive learning is shown in Figure 17.6. We extended Haussler's algorithm to allow join constraints in the description of hypotheses, i.e., alternative queries to be learned. 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.

The input of the algorithm is a user query `Q` and the database relations. We use `Q22` in Figure 17.2 and the database fragment shown in Figure 17.7 as an example to explain the algorithm. The primary relation of a query is the relation that must be accessed to answer the input query. For example, the primary relation of `Q22` is `geoloc` because the output variable, `?name`, of the query is bound to an attribute of `geoloc`. If output variables are bound to attributes from different relations, then the primary relation is a relation derived by joining those relations.

Initially, the system determines the primary relation of an input query and labels the instances in the relation as positive or negative. An instance is positive if it satisfies the input query; otherwise, it is negative. In our example, the primary relation is `geoloc` and its instances are labeled according to `Q22` as shown in Figure 17.7. The next subsection will describe how to construct and evaluate candidate constraints, which can be either an internal disjunction or a join constraint. Then subsection 17.4.2 will describe a preference heuristic to restrict the number of candidate constraints in each iteration.

17.4.1 Constructing and Evaluating Candidate Constraints

For each attribute of the primary relation, the system can construct an internal disjunction as a candidate constraint by generalizing attribute values of positive instances. The constructed constraint is consistent with positive instances because it is satisfied by all positive instances. In our example database, for attribute `country`, the system can

```
INPUT Q = input query; DB = database relations;
BEGIN
  LET r = primary relation of Q; LET AQ= alternative query (initially empty);
  LET C = set of candidate constraints (initially empty);
  Construct candidate constraints on r and add them to C;
  REPEAT
    Evaluate gain/cost of candidate constraints in C;
    LET c = candidate constraint with the highest gain/cost in C;
    IF gain(c) > 0 THEN
       Merge c to AQ, and C = C - c;
       IF AQ ⇔ Q THEN RETURN AQ;
       IF c is a join constraint on a new relation r' THEN
          Construct candidate constraints on r' and add them to C;
       ENDIF;
    ENDIF;
  UNTIL gain(c) = 0;
  RETURN fail, because no AQ is found to be equivalent to Q;
END.
```

Figure 17.6
Inductive algorithm for learning alternative queries

```
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 17.7
A database fragment

generalize from the positive instances a candidate constraint:
`geoloc(?name,_,?cty,_,_), ?cty = "Malta",`
because the `country` value of all positive instances is `Malta`.

Similarly, the system considers a join constraint as a candidate constraint if it is consistent with all positive instances. This can be verified by checking whether all positive instances satisfy the join constraint. Suppose the system is verifying whether join constraint `C2` in Figure 17.5 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 evaluation 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. Note that the negative instances excluded in previous iterations will not be counted as gain for the constraints being evaluated. The gain/cost

Table 17.1
Cost estimates of constraints in a query

Internal disjunctions, on NON-indexed attribute	$\lvert\mathcal{D}_1\rvert$
Internal disjunctions, on indexed attribute	$\mathcal{I}$
Join, over two NON-indexed attributes	$\lvert\mathcal{D}_1\rvert \cdot \lvert\mathcal{D}_2\rvert$
Join, over two indexed attributes	$\frac{\lvert\mathcal{D}_1\rvert \cdot \lvert\mathcal{D}_2\rvert}{max(\mathcal{I}_1, \mathcal{I}_2)}$

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 lowest evaluation cost. We define the *gain* part of the heuristic as the number of excluded negative instances in the primary relation, and define the *cost* of the function as the estimated evaluation cost of the candidate constraint.

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 (Coremen et al. 1989), 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 cover 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 here.

The evaluation cost of individual constraints can be estimated using standard query size estimation techniques (Ullman 1988). A set of simple estimates is shown in Table 17.1. For an internal disjunction on a non-indexed attribute of a relation $\mathcal{D}$, a query evaluator has to scan the entire relation to find all satisfying instances. Thus, its evaluation cost is proportional to $|\mathcal{D}|$, the size of $\mathcal{D}$. If the internal disjunction is on an indexed attribute, then its cost is proportional to the number of instances satisfying the constraint, denoted as $\mathcal{I}$.

For join constraints, let $\mathcal{D}_1$ and $\mathcal{D}_2$ denote the relations that are joined, and $\mathcal{I}_1$, $\mathcal{I}_2$ denote the number of the distinct attribute values used for join. Then the evaluation cost for the join over $\mathcal{D}_1$ and $\mathcal{D}_2$ is proportional to $|\mathcal{D}_1| \cdot |\mathcal{D}_2|$ 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, $\frac{|\mathcal{D}_1| \cdot |\mathcal{D}_2|}{max(\mathcal{I}_1, \mathcal{I}_2)}$. This estimate assumes that distinct attribute values distribute uniformly in instances of joined relations. If possible, the system may sample the database for more accurate estimation.

For the example at hand, two candidate constraints are the most promising:

```
C3:geoloc(?name,_,"Malta",_,_).
C4:geoloc(?name,?glc_cd,_,_,_),
   seaport(_,?glc_cd,_,_,_,_).
```

Suppose `|geoloc|` is 30,000, and `|seaport|` is 800. The cardinality of `glc_cd` for `geoloc` is 30,000 again, and for `seaport` is 800. Suppose both relations have indices on `glc_cd`. Then the evaluation cost of `C3` is 30,000, and `C4` is $30,000 * 800/30,000 = 800$. The gain of `C3` is $30,000 - 4 = 29,996$, and the gain of `C4` is $30,000 - 800 = 29,200$, because only 4 instances satisfy `C3` (See Figure 17.7) while 800 instances satisfy `C4`. (There are 800 seaports, and all have a corresponding `geoloc` instance.) So the gain/cost ratio of `C3` is $29,996/30,000 = 0.99$, and the gain/cost ratio of `C4` is $29,200/800 = 36.50$. The system will select `C4` and add it to the hypothesis.

17.4.2 Searching the Space of Candidate Constraints

When a join constraint is selected, a new relation and its attributes are introduced into 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 levels, as illustrated in Figure 17.8. 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 17.8 shows the situation when a new relation, say `channel`, is selected, the search space will be expanded one level 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 level, until the system constructs a hypothesis that excludes all negative instances (i.e., reaches the goal) or no more consistent constraints in the level with positive gain are found. In the later case, the system will backtrack to search the remaining constraints on previous levels. 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 level have been selected, and it is more likely that useful candidate constraints

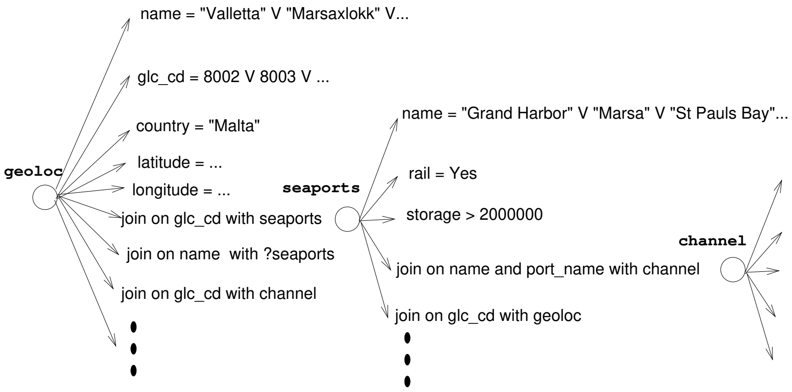

Figure 17.8
Candidate constraints to be selected

are on attributes of newly joined relations. This bias works well in our experiments. But certainly there are cases when the heuristic prunes out useful candidate constraints.

The complexity of the algorithm is briefly analyzed as follows. When a new relation r is introduced as a primary relation or by selection of a join, the number of relation scans is bounded by $(1 + J(r)) + (A(r) + J(r))$, where $J(r)$ is the number of legal join paths to r and $A(r)$ is the number of attributes of r. Constructing candidate constraints requires scanning the relations $1 + J(r)$ times, because constructing all internal disjunctions on r needs one scan over r and constructing join constraints needs an additional scan over each joined relation. Each iteration of gain/cost evaluation and selection needs to scan r once. In the worst case, if all candidate constraints are selected to construct the alternative query, it will require scanning the relations $A(r) + J(r)$ times. Since usually a query involves a small number of relations and expansions in learning are rare, the number of relation scans is linear with respect to the number of attributes in most cases.

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 candidates on `geoloc` again and select the constraint on `country` (see Figure 17.8). Now, 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 the equivalence of the input query `Q22` and the learned query `Q3` as input:

Table 17.2
Database features

Databases	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

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

and will deduce a new rule that can be used to reformulate `Q22` to `Q3`:

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

This is the rule `R2` we have seen in Section 17.2. 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.

17.5 Experimental Results

Our experiments are performed in the SIMS information mediator (Arens et al. 1993; Knoblock et al. 1994). 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. Figure 17.2 shows the domain of the contents and the sizes of these databases. Together with the databases are 28 sample queries written by the users of the databases. However, among these queries, only 7 are multidatabase queries, and 4 of them return NIL because the data in `Assets` and `Fmlib` databases are incomplete. To test the effect of data transmission cost reduction, we wrote 6 additional multidatabase queries. Therefore, we have a total of 34 sample queries for the experiments.

We classified 28 sample 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 semantic rules. To reduce the learning cost, a multidatabase query will be decomposed into single-database subqueries by the SIMS query planner (Arens et al. 1993; Knoblock et al. 1994) before being fed into the learning system. The learned rules were used to optimize the remaining 26 queries. In addition to rules, the system also used 163 attribute range facts (e.g., the range of the `age` attribute of `employee` is from 17 to 70) compiled from the databases.

Table 17.3
Performance statistics

	All	NIL Queries	$\leq$ 60s.	$>$ 60s.
# of queries	26	4	17	5
Time(sec), w/out reformulation	51.79	41.51	7.97	209.03
Time(sec), with reformulation	20.80	2.37	6.66	83.57
Overall % time saved	59.84%	94.28%	16.38%	60.00%
Average % time saved	29.36%	79.62%	12.56%	46.28%
Average overhead(sec)	0.08	0.07	0.07	0.11
Times range facts applied	3.84	5.25	2.82	6.2
Times rules applied	1.15	0.75	1.35	0.80

Table 17.3 shows the performance statistics. 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 multidatabase queries that we wrote to test the reduction of data transmission cost by reformulation.

In Table 17.3, 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. The overall percentage time saved is the ratio of the total time saved due to the reformulation over the total execution time without reformulation. The next row shows the average percentage saving of designated sets of queries. That is, the sum of percentage time saved of each query divided by the number of queries. The savings is 59.84 percent overall and 29.36 percent on average. The reformulation yields significant cost reduction for high-cost queries, but not so high for the low-cost queries. This is not unexpected, because the queries in this group are already very cheap and the cost cannot be reduced much further. The average overhead listed in the table shows the time in seconds spent on reformulation. The overhead is very small compared to the total query processing time.

On average, the system applies range facts 3.84 times and semantic rules 1.15 times for reformulation. Note that the same range fact or rule may be applied more than once during the reformulation procedure. In fact, the system reformulates many of the queries using the range facts only. To distinguish the effect of learned rules in reformulation,

Table 17.4
Performance statistics for queries optimized using learned semantic rules

	All	NIL Queries	$\leq$ 60s.	$>$ 60s.
# of queries	11	1	9	1
Time(sec), w/out reformulation	21.25	67.90	8.86	86.09
Time(sec), with reformulation	12.01	4.15	6.99	65.01
Overall % time saved	43.48%	93.88%	21.09%	24.49%
Average % time saved	23.67%	93.88%	15.78%	24.49%
Average overhead(sec)	0.09	0.03	0.09	0.21
Times range facts applied	4.00	2.00	3.44	11.0
Times rules applied	2.72	3.00	2.55	4.00

we separate the queries for which the system applies at least one rule in reformulation. Range facts are still necessary for reformulating these queries because the system uses them in the rule matching for numerically typed attributes. (Hsu and Knoblock 1993b) describes in detail the usage and acquisition of range facts. The performance statistics on those queries are shown in Table 17.4. There are 11 out of 26 testing queries in this set. The overall saving of this class is 43.48 percent, comparable to the SQO systems using hand-coded rules (King 1981; Shekhar et al. 1988; Shenoy and Ozsoyoglu 1989).

17.6 Related Work

Previously, systems for learning background knowledge for semantic query optimization were proposed by Siegel (1988) and by Shekhar et al. (1993). Siegel's system uses predefined heuristics and an example query to drive the learning. This approach is limited because the heuristics are unlikely to be comprehensive enough to detect missing rules for various queries and databases. Shekhar et al.'s system uses 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 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.

The problem of Inductive Logic Programming (ILP) (Muggleton and Feng 1990; Quinlan 1990; Lavrač and Džeroski 1994) is closely related to our problem of learning alternative queries in that both problems learn definitions from databases with multiple relations. Our inductive learning approach uses a top-down algorithm similar to FOIL (Quinlan 1990) to build an alternative query. The differences between our approach and FOIL are that they learn descriptions in a different language. FOIL learns Horn-clause definitions where each clause covers a subset of positive instances but no negative instances. Our approach learns conjunctive queries which must cover all positive instances but no negative instances. Another difference is their search heuristics. FOIL uses an information-theoretic heuristic while our approach uses a set-covering heuristic for learning a low-cost specification.

Approaches to mining association rules (propositional conjunctive rules) from a single table database are described by Agrawal et al. (1993) and Mannila et al. (1994). Their approach generates a set of data patterns from a table, and then converts those patterns into association rules. The data patterns are generated after the system scans the database a few times. In each pass, the system revises a set of candidate patterns, by proposing new patterns and eliminating existing patterns, as it reads in a data tuple. A "support" counter for each pattern that counts the number of tuples showing a given pattern is used to measure the interestingness of patterns. A tuple scanning approach is not appropriate when joins are allowed to express a rule because the system must consider data patterns in many relations at the same time. Also, in their approaches, the "support" counters for measuring interestingness of rules can be efficiently updated and estimated during the tuple scanning process, while the effectiveness of semantic rules for SQO is difficult to measure and estimate in that manner.

17.7 Conclusions and Future Work

This chapter demonstrates that the knowledge required for semantic query optimization can be learned inductively under the guidance of input queries. We have described a general learning framework in which inductive learning is triggered by queries, and an inductive learning algorithm for learning from many relations. Experimental results show that query optimization using learned semantic knowledge produces substantial cost reductions for a real-world multidatabase system.

A limitation to our approach is that there is no mechanism to deal with changes to databases. After a database is changed, some learned semantic rules may become inconsistent with a new database state and not useful for optimization. Our planned approach to this issue is to estimate the robustness of candidate rules and learn those rules with high robustness confidence. When the database is changed, a maintenance system will be used to update the confidence and delete those rules with low confidence. Meanwhile, as new queries arrive, the system keeps triggering learning for new rules for new database states. This way, the system can autonomously maintain a set of effective and consistent rules for optimization. We are currently developing an estimation approach to implement this idea.

Acknowledgements This chapter is extended from (Hsu and Knoblock 1994). We wish to thank the SIMS project members: Yigal Arens, Wei-Min Shen, Chin Y. Chee and José-Luis Ambite for their help on this work. Thanks also to Yolanda Gil, Dennis McLeod, Dan O'Leary, Gregory Piatetsky-Shapiro, Paul Rosenbloom, Milind Tambe and the anonymous reviewers for their valuable comments. 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.

Bibliography

Agrawal, R., Imielinski, T., and Swami, A. 1993. Mining Association Rules between Sets of Items in Large Databases. In Proceedings of ACM SIGMOD, Washington, D.C., 207–216.

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.

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

Hammer, M. and Zdonik, S. B. 1980. Knowledge-based Query Processing. In Proceedings of the Sixth VLDB Conference, Washington, D.C., 137–146.

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. 1994. Rule Induction for Semantic Query Optimizer. *In Machine Learning, Proceedings of the 11th International Conference,* 112–120. San Mateo, California: Morgan Kaufmann.

Hsu, C.-N. and Knoblock, C. A. 1993a. Learning Database Abstractions for Query Reformulation. In Proceedings of KDD-93, AAAI-93 Workshop on Knowledge Discovery in Databases, Washington, D.C., 276–290.

Hsu, C.-N. and Knoblock, C. A. 1993b. Reformulating Query Plans for Multidatabase Systems. In Proceedings of the Second International Conference on Information and Knowledge Management, Washington, D.C., 423–432.

Jarke, M. and Koch, J. 1984. Query Optimization in Database Systems. *ACM Computer Surveys*, 16:111–152.

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

Knoblock, C. A., Arens, Y., and Hsu, C.-N. 1994. Cooperating Agents for Information Retrieval. In Proceedings of the Second International Conference on Intelligent and Cooperative Information Systems, Toronto, Ontario, Canada.

Lavrač, N. and Džeroski, S. 1994. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood.

Mannila, H., Toivonen, H., and Verkamo, A. I. 1994. Efficient Algorithms for Discovering Association Rules. In Proceedings of KDD-94, AAAI-94 Workshop on Knowledge Discovery in Databases. Seattle, Washington, 181–192.

Muggleton, S. and Feng, C. 1990. Efficient Induction of Logic Programs. In Proceedings of the First Conference on Algorithmic Learning Theory. Tokyo, Japan.

Quinlan, J. R. 1990. Learning Logical Definitions from Relations. *Machine Learning* 5:239–266.

Shavlik, J. and Dietterich, T. A., eds. 1990. *Readings in Machine Learning.* San Mateo, California: 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. Los Angeles, California.

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 Optimizer. In Kerschberg, L., editor, Proceedings of the Second International Conference on Expert Database Systems, 371–385. Fairfax, VA.:George Mason Foundation.

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

Yu, C. T. and Sun, W. 1989. Automatic Knowledge Acquisition and Maintenance for Semantic Query Optimizer. *IEEE Trans. Knowledge and Data Engineering* I(3):362–375.