LEARNING EFFECTIVE AND ROBUST KNOWLEDGE FOR SEMANTIC QUERY OPTIMIZATION

by

Chun-Nan Hsu

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(Computer Science)

December 1996

To my mother

Goá beh chiong chit-pún phok-sū lūn-bûn hiàn hō͘ ah-mi.

Acknowledgements

I am grateful to many persons for their generous help in this work. First and foremost is my advisor Craig A. Knoblock. He was patient and willing to spend time with me on things as trivial as how to use "a" or "the," and things as profound as how to organize random thoughts into a coherent presentable idea. He was the best role model and confidence builder. He always said "it's a good start" to encourage me even when I screwed up badly.

Then there is my wife Sunny. She took over almost everything of the running our home from South Pasadena to Los Angeles to Tempe, Arizona. Too often, she had to tolerate my bad temper during those stressful days for this work. I wish I can compensate her with the rest of my life.

I wish to thank my thesis committee members, Paul Rosenbloom and Dan O'Leary for their valuable comments on this work. Paul pointed out several problems of this work in its early stage and helped me to shape the empirical evaluation of this work.

Next comes the SIMS project members: Yigal Arens, our project leader, did a wonderful job to make it a marvelous experience to participate in this project. Andrew Philpot and Chin Y. Chee spent days and hours to set up the system for my experiments. José-Luis Ambite, my officemate, kept our office a fun place to work. Thanks also to other researchers and staff members at USC/Information Sciences Institute: Yolanda Gil and Steve Minton reviewed some of my papers that later turned into this dissertation. Wei-Min Shen brought the Bayesian theory to my attention. Milind Tambe patiently answered lots of my trivial questions on utility problems. Next are professors at the department of computer science. Dennis Mcleod reminded me that half of this work is about databases. I am particularly grateful to Alvin Despain, who brought me into the PhD program in the first place.

Many people in the research community contributed important comments to this work: Gregory Paitetsky-Shapiro, Haym Hirsh, Oren Etzioni, Russell Stewart, Wesley Chu, Micheal Kearn, Subbarao Kambhampati, to name a few. Steve Yau, the chair of Computer Science and Engineering department at Arizona State University, provided equipment for me to finish this work.

I also wish to thank fellow graduate students who make my life at ISI delightful. They also helped to generate sample data for my experiments. Also, I would like to thank friends from Taiwanese Student Association. With them I did some exciting things at USC. I will remember Los Angeles, who fed me with drought, flood, riot, and earthquake that enriched my life significantly.

Finally, I thank my parents for their life time support. My decision to pursue this PhD thing is mostly due to my father, who "programmed" me to believe that being a scientist is the greatest career on this planet.

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 Defense Advanced Research Projects Agency under Contract No. F30602-94-C-0210. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of NSF, DARPA, RL, the U.S. Government, or any person or agency connected with them.

Contents

List Of Tables

List Of Figures

Abstract

Optimizing queries to heterogeneous, distributed multidatabases is an important problem. Due to the query complexity and the heterogeneity of databases, it is difficult for conventional optimization approaches to solve the problem satisfactorily. Semantic Query Optimization (SQO) can complement conventional approaches to overcome the heterogeneity and considerably reduce redundant data transmission. SQO optimizers use rules about data regularities to yield significant cost reduction. However, hand coding useful rules for SQO is impracticable. This dissertation presents a machine learning approach to this knowledge bottleneck problem.

Unlike search control rules or classification rules studied extensively in machine learning, two roughly correlated measures must be maximized in the learning of high utility rules for SQO. The first measure is the *effectiveness*. Effective rules must be applicable in many different queries and yield high cost reduction. The second measure is the *robustness* against database changes. That is, they must remain valid regardless of database changes. This dissertation presents a new inductive learning approach to learning effective and robust rules. The learning approach considers both applicability and cost-reduction in rule induction to learn effective rules. The learned rules are robust because the learner is able to guide the learning for robust rules with an approach to estimating the probabilities of database changes.

To evaluate the utility of the learning approach, this dissertation also describes an extended SQO approach for query plans that retrieve data from heterogeneous multidatabases. The experimental results show that the learned rules produce significant savings while being robust against database changes. The learning and optimization approaches provide a complete solution for multidatabase information systems to effectively optimize queries using SQO that does not require an expensive coding effort to produce useful rules.

Chapter 1

Introduction

Query optimization is important for information systems. An effective query optimizer can automatically search for efficient procedures to retrieve data, which allows us to query an information system without the need to understand its internal mechanism. Query optimization is increasingly important in the 1990s and, according to a report on the trends of database research [Silberschatz *et al.*, 1996], it will be even more important in the twenty-first century due to the increasing complexity of queries and the heterogeneity of information sources. However, it is difficult for conventional query optimization techniques to solve all the problems for the next generation information systems. *Semantic query optimization* (SQO) [Hammer and Zdonik, 1980, King, 1981, Siegel, 1988, Shekhar *et al.*, 1988, Shenoy and Ozsoyoglu, 1989, Yu and Sun, 1989, Chakravarthy *et al.*, 1990, Sun and Yu, 1994] is a promising query optimization technique that can complement conventional techniques to overcome the heterogeneity and considerably reduce query execution cost. The essential idea of semantic query optimization is to use semantic rules about data, such as *all California seaports have railroad access*, to reformulate a query into a more efficient but semantically equivalent query. Given this rule, suppose we have a query

> Query: Find all California seaports with railroad access and 2,000,000 ft^3 of storage space.

The system can reformulate the query into a new query

> Query (optimized): Find all California seaports with 2,000,000 ft^3 of storage space.

This optimized query is equivalent to the original query, because from the given rule there is no need to check the railroad access of seaports in California and the optimized query will still return the same answer. Executing the optimized query is less expensive than executing the original query because the system saves the time for the redundant comparisons.

Though semantic query optimization is an effective and promising technique, it requires sufficient semantic knowledge to yield high cost reduction. Previous work in SQO, such as [King, 1981], assume that the optimizer can use *semantic integrity constraints* given by users for the optimization. Semantic integrity constraints express rules that must be followed by the data in any state of a database. Examples of semantic integrity constraints for a hospital database are *the age of a patient must be a nonnegative number smaller than 150*, or *only female patients can be pregnant.* Semantic integrity constraints are useful to guarantee the correct use of a database system, but they do not necessarily reflect the data distributions that affect query execution cost. Also, because users usually already possess some knowledge about the application domains of databases, it is unlikely for them to issue a query where semantic integrity constraints can be applied in query optimization (e.g., it is unlikely for any one to issue a query to a database for the gender of pregnant patients). On the other hand, it is difficult to encode semantic knowledge that both reflects data distributions and matches query patterns. This *knowledge bottleneck* problem is one of the reasons why semantic query optimization is not applied widely in practice.

This dissertation presents a machine learning approach to this problem. To constrain the learning for high utility rules, two important issues must be addressed. The first issue is that the learner needs to generate *effective* semantic rules. Effective rules allow an SQO optimizer to achieve high cost reduction in query optimization. The second issue is that the learner needs to generate *robust* rules. A rule is robust against changes to the database if it remains consistent with the data after changes. Inconsistent rules are not useful for SQO because using inconsistent rules the optimizer may reformulate a query into a new query not equivalent to the original query and cause the system to produce incorrect results. The learner must address both issues to generate high utility rules. This dissertation investigates these issues and presents an inductive learning approach to learning effective and robust rules. To demonstrate the learning approach, the dissertation also describes a novel semantic

```
Q1: assets(?ship_class,?draft):-
       ship_class(?ship_class,_,?draft,_,"Y"),
       ship(_,?ship_class,"Active",_,_),
       ?draft < 50.

Q1.1: assets(?ship_class,?draft):-
       ship_class(?ship_class,_,?draft,_,"Y"),
       ship(_,?ship_class,_,_,_),
       ?draft < 50.
Q1.2: assets(?ship_class,?draft):-
       ship_class(?ship_class,_,?draft,_,"Y"),
       ?draft < 50.
Q1.3: assets(?ship_class,?draft):-
       ship_class(?ship_class,_,?draft,_,"Y"),
       ship(_,?ship_class,"Active",_,?year-built),
       ?year-built > 1945,
       ?draft < 50.
```

Table 1.1: Equivalent queries of Q1 deduced from semantic knowledge

query optimization approach for information systems that integrate heterogeneous databases. The experiments show that using the learned rules in query optimization provides significant cost reduction

The remainder of this chapter describes the problem more precisely and presents an outline of the solution. The next section will explain semantic query optimization and discuss what kind of semantic rules are useful for semantic query optimization.

1.1 Semantic Query Optimization

The goal of semantic query optimization is to search for the more efficient semantically equivalent query of an input query using the given semantic knowledge. Two queries are defined to be *semantically equivalent* if they return identical answers. This section explains how semantic query optimization works using a concrete example.

Consider the conjunctive query `Q1` in Table 1.1. This query retrieves the ship classes and the maximal draft of the ships in those classes which satisfy the following conditions: the ships in the class are capable of carrying containers, their draft is less than 50 feet, and there is at least one active ship in this class. Suppose further that prior to receiving this input query, the optimizer possesses a set of semantic

Rules:

`R1.1`: *If the maximum draft of a ship is less than 50, then its status is active.*
`ship_class(?class,_,?draft,_,_)` ∧ `ship(_,?class,?status,_,_)` ∧ `?draft < 50`
⇒ `?status = "Active"`

`R1.2`: *If a ship class has container capability, then there must exist some ships that belong to that ship class in the database.*
`ship_class(?class,_,_,_,"Y")` ⇒ `ship(_,?class,_,_,_)`

`R1.3`: *If a ship is active, then it is built after 1945.*
`ship(_,?class,_,"Active",?year-built)` ⇒ `?year-built > 1945`

Table 1.2: Applicable semantic rules for Q1

knowledge about the data. Based on the semantic knowledge, the optimizer will propose applicable reformulations to optimize the input query. Possible reformulations are inserting a new literal to the query, deleting a literal and refuting the entire query (asserting that it will return an empty set). Proposing reformulations involves locating applicable semantic rules from the set of given semantic knowledge. A rule is considered *applicable* to a query if the antecedent part of the rule is a logical consequence of the query literals. This can be determined by applying SLD-refutation [Lloyd, 1987] to a target rule and the query. The applicable semantic knowledge for this query is given in Table 1.2.

Once the antecedent of an applicable rule is unified, a set of variable substitutions will be returned and used to substitute variables in the consequent of the rule. The optimizer can proceed to delete a literal from the query if the consequent implies a literal in the query, or insert the consequent as a new literal. Some of the reformulated equivalent queries of `Q1` are shown in Table 1.1. `Q1.1` is reformulated from `Q1` by applying `R1.1`. In this reformulation, the constraint on the status `''Active''` is deleted. This is an example of *constraint elimination* [King, 1981] reformulation. From `Q1.1` and `R1.2`, we can infer that the literal on a database relation `ship` is implied by other literals in `Q1.1` and thus is redundant. Therefore, it can be deleted and the resulting query `Q1.2` is still equivalent to `Q1`. This is an example of *join elimination* [King, 1981] reformulation where the relational join from `ship_class` to `ship` is eliminated. In addition to deletions, we may also add new constraints to a query based on the semantic rules. For example, we can add `?year-built > 1945` to `Q1` from `R1.3` to yield another equivalent query `Q1.3` of `Q1`. Adding new constraints could be useful in many situations. One of them is when the added constraint is specified on an indexed attribute. The optimizer can *refute* a query, when it infers that the query literals contradict a rule (or a chain of rules) and will not be satisfiable by the data. Sometimes the optimizer can assert the answer directly from semantic rules. In either case, there is no need to access the database to answer the query and we could achieve close to 100 percent savings.

In semantic query optimization, when an applicable rule for eliminating a literal is located, the optimizer will not perform the elimination immediately like the example described above, because another applicable rule may no longer be applicable without the presence of that literal in the query. To make sure that no applicable rule, and hence no optimization opportunity, is missed, the optimizer can propagate the results of rule applications and generate a closure of implications, which includes all redundant literals that can be deleted from an input query and all newly inserted literals. An example of such a closure is indicated by the literals underlined in `Q1.3`. Researchers have developed several polynomial algorithms to generate closures of implications [Shenoy and Ozsoyoglu, 1989, Yu and Sun, 1989, Hsu and Knoblock, 1993b]. With the closures, the optimizer can search for the least expensive equivalent query by identifying an optimal combination of insertions and deletions from the closure. In our example, since the equivalent query `Q1.2` does not involve access to the large `ship` relation and thus is the least expensive query based on the applicable rules, the optimizer will return it as the output and send it to the query processor to retrieve data. Since an SQO optimizer can use its knowledge about the content of an information source, the search space of equivalent queries in SQO is much larger than its counterpart in conventional syntactic query optimization. For this reason, it can detect more optimization opportunities and provide higher cost reduction.

A key advantage of semantic query optimization is that it can be easily applied in *information mediators* [Wiederhold, 1992, Arens *et al.*, 1993, Knoblock *et al.*, 1994, Levy *et al.*, 1995, Hammer *et al.*, 1995, Arens *et al.*, 1996] that integrate heterogeneous information sources. Query processing in an information mediator can be briefly summarized as follows. An information mediator possesses a shared, global model that describes a set of integrated information sources. To allow the mediator to retrieve data, each information source is wrapped by a component that translates between the language used by the source and the language used by the mediator. When the information mediator receives a query, it will decompose the query into a set of subqueries expressed in the global model to retrieve and combine data from individual information sources. The wrappers of information sources will translate the subqueries into the corresponding query language before the subqueries are executed. Finally, the information mediator will combine retrieved data and present the results to the user.

Semantic query optimization is especially appropriate for information mediators because it reformulates a query into a new query expressed in the same language, rather than optimizing how a query is executed. As a result, we can use it to perform global optimization on the subqueries before they are sent to individual information sources and overcome the heterogeneity. Meanwhile, existing query optimizers for information sources can still be used to perform local optimization. Semantic query optimization also supports the extensibility of heterogeneous information systems. Since it is independent on how individual sources execute a query, when a new information source is integrated to the information mediator, the SQO optimizer can still be used without change to its code. Chapter 4 of this dissertation describes an extended SQO approach to the query optimization in heterogeneous information systems. An implementation of this approach is used in the experiment to demonstrate the effectiveness of our learning approach.

1.2 High Utility Semantic Knowledge for SQO

Not all semantic rules are useful for semantic query optimization. The first, and basic criterion of a useful semantic rule is that it must be *consistent* with the given database, otherwise, the optimized query might return incorrect answers. Another basic criterion is *operational*, that is, a semantic rule must be in the form ready to be used. For all existing semantic query optimizers, operational rules must be expressed in Horn-clauses. We make a distinction between two types of Horn-clause rules. The first type, referred to as a *range rule*, contains rules whose consequent is a built-in literal, such as `?x > 3`. For instance, `R1.1` in Table 1.2 is a range rule. The second type consists of *relational rules*, whose consequent is a literal on a database relation. `R1.2` in Table 1.2 is an example of relational rules. Range rules are used to derive redundant literals or introduce cost-reducing literals, and relational rules are useful for detecting redundant retrievals and joins of database relations. Since relational joins are expensive in query execution and relational rules allow an SQO optimizer to detect redundant joins, it is important to use relational rules in the optimization.

We note that previous work in SQO cannot apply general relational rules in the optimization [Hammer and Zdonik, 1980, King, 1981, Siegel, 1988, Shekhar *et al.*, 1988, Shenoy and Ozsoyoglu, 1989, Yu and Sun, 1989, Chakravarthy *et al.*, 1990, Sun and Yu, 1994]. To detect redundant joins, they use referential integrity constraints [Ullman, 1988], a restrictive form of relational rules that allow only one literal as the antecedent. One of the new features of the SQO approach presented in Chapter 4 is that it can apply relational rules. To support this feature and achieve high cost reduction, it is important that the learner can learn relational rules.

Consistency and operationality, however, are not sufficient to guide the design of a learning approach since the number of consistent and operational Horn-clause rules that can be derived from a database is enormous. In addition , we need to analyze other characteristics of high utility semantic rules. Generally speaking, high utility rules for semantic query optimization must be effective in producing high savings for a wide range of queries, while require a minimal cost to be used.

The cost to use semantic rules includes the storage space for the rules, the computation time to match and apply the rules during the optimization, and the cost to maintain inconsistent rules in the presence of database changes. If a learning approach can always learn invariant semantic rules that are consistent with all possible database states regardless of how a database changes, then the cost of maintaining inconsistent rules can be eliminated. However, it is prohibitive to guarantee that all the learned rules are invariants, because it is impossible for the learner to verify whether a rule is invariant without complete knowledge about the database application domains. Also, invariant rules might not produce high savings in query optimization. Therefore, a learning approach must address the issue of minimizing the maintenance cost in the presence of database changes.

There are several possible approaches to this issue. As the system detects an inconsistent rule after data modification transactions, it can delete the rule or repair the rule. Deleting inconsistent rules is simple and efficient, however, if the learned rules are not robust against database changes, after a few changes, most of rules will become inconsistent and the remaining rules may not be sufficient to support the optimizer. As a result, the system will have to re-learn semantic rules frequently and incur an expensive cost. Conversely, the system can choose to repair an inconsistent rule. However, repairing rules is not trivial. In some applications, the databases are rarely modified and an additional rule repairing system merely increases the complexity and the size of an information system.

An alternative approach to dealing with the issue of database changes is to learn *robust* semantic rules. A semantic rule is robust if it is *likely* to remain consistent in new database states after database changes. An invariant semantic rule is extremely robust but it is prohibitive to learn invariant rules, so it is a more practical goal to learn robust rules for a learning approach.

A set of semantic rules is extremely effective if, for any query, it allows the optimizer to reformulate the query into the optimal equivalent query. Intuitively, an optimal equivalent query is the one that returns the same answer as a given query and can be executed with the lowest possible cost. However, if we take database changes into account, an optimal equivalent query in one database state might not be equivalent to a given query in another database state, especially when the semantic rules used to infer the equivalent queries are not invariant. Therefore, a set of effective rules might not be robust. Similarly, we can see that a robust rule might not be effective. The learner must balance these two factors — effectiveness and robustness — to maximize the net utility of learning.

1.3 Learning Effective and Robust Rules

In real-world applications, database usage can be modeled as a stochastic sequence of queries and data modification transactions over time. The distribution and density of the sequence may depend on applications. A general solution should not assume any particular distribution or density of queries and transactions. The goal of this research is to develop a general learning approach that maximizes the net utility of semantic rules throughout the life span of a information system. To achieve this goal, I have developed an approach to the estimation of the robustness of semantic rules [Hsu and Knoblock, 1995]. This estimation approach allows the learner to determine the degree of the robustness of a semantic rule and use the results to guide the search for robust rules. This estimation approach will be described in detail in Chapter 2.

Robustness is different from *predictive accuracy* in rule induction (e.g., [Michalski, 1983, Haussler, 1988, Clark and Niblett, 1989, Quinlan, 1990, Muggleton and Feng, 1990, Clark and Boswell, 1991, Pazzani and Kibler, 1992, Quinlan, 1993, Lavrač and Džeroski, 1994]) which mainly concerns data insertions. In contrast, database changes in our learning problem includes updates or deletions. Under the closed-world assumption [Lloyd, 1987], they have the same semantic effect as to assert that the old data are now false and cause the learned rules to become inconsistent. Chapter 2 will compare the difference between robustness and predictive accuracy in detail.

The estimation approach is integrated into the general learning approach to the acquisition of effective and robust semantic rules. The organization of this learning approach is illustrated in Figure 1.1. This learning approach has two components: an inductive learning component and a pruning component. The system is triggered to learn a new set of rules when the query processor encounters an expensive query that can not be satisfactorily optimized with existing rules. Given such an expensive query, the inductive learning component inductively forms an alternative query from the data. Ideally, the resulting alternative query is the optimal equivalent query in the given database state. Based on the alternative query, the learner can proceed to derive a set of semantic rules to allow the optimizer to optimize the given query into the alternative query. The inductive learning component uses knowledge about query execution cost to bias the induction so that the learned alternative query can be evaluated with a minimum cost that is close to the optimal. Another important feature of this inductive learning component is that it can form an alternative query with relational joins from relational data [Hsu and Knoblock, 1994, Hsu and Knoblock, 1996b]. This feature allows the learner to generate relational rules for the optimizer to detect redundant relational joins.

The semantic rules derived from the alternative query may not be robust and could be overly-specific to the given query. To address this problem, the learner uses a rule pruner [Hsu and Knoblock, 1996a] to prune the literals in the derived rules to increase their applicability to a wide range of queries. When pruning for robust

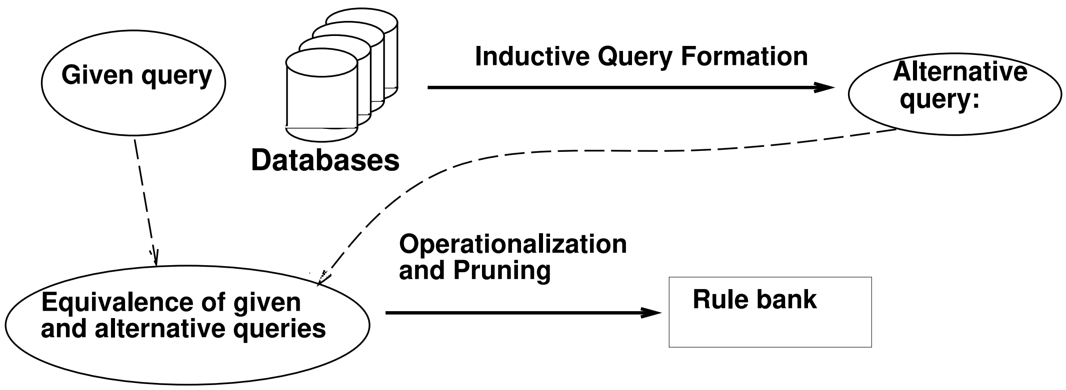

Figure 1.1: Organization of the learning approach for SQO

rules, the learner uses the robustness estimation approach to guide the pruning so that the resulting rules are robust against database changes. This way, the learner will be able to learn effective and robust rules for semantic query optimization.

1.4 Closely Related Work

Previously, three approaches to automating the knowledge acquisition for semantic query optimization were proposed in [Siegel, 1988], [Shekhar *et al.*, 1993] and [Yu and Sun, 1989]. The first approach is a query-driven approach due to [Siegel, 1988, Siegel, 1989]. Siegel's system learns *simple rules*, a limited form of rules that allows exactly one literal on each side of implication. His system uses a set of predefined heuristics combined with example queries to guide the learning for effective rules. The heuristics are identical to those proposed by [King, 1981] to guide semantic query optimization. An example of such a heuristic is as follows:

> If a selection condition on attribute A is implied by another selection condition on attribute B and A is not an indexed attribute, then the selection on A can be removed from the query.

This heuristic is called the *selection reduction* heuristic. There are seven such heuristics. Based on this heuristic, when his system receives an input query, it checks whether there is a selection on an attribute `A` which is not indexed. If such an `A` exists, then the heuristic will propose deriving a rule so that `A` is implied by another selection condition on attribute `B` in the query and his system will construct a rule

		NaturalGas	RefinedOil	Others
BusinessType	Petroleum	1230	300	0
	NonPetroleum	356	30	523
		CargoType		

Table 1.3: Data-distribution grid

from the data. In this way, suppose the system receives a query with n selection conditions and none of them are on indexed attributes. The system will propose to learn n^2 rules from this heuristic alone. In addition, this approach does not consider the robustness of rules in learning. The fundamental limitation of this approach is that the heuristics depend only on the query specification but do not take properties of data into account, and thus may miss learning many high utility rules.

Shekhar et al. develop a data-driven approach to the learning problem of semantic query optimization [Shekhar *et al.*, 1993]. Their system is based on an assumption that useful semantic rules can be derived from the nonuniform distributions of attribute values. To detect nonuniform data distributions, their system constructs a set of *data distribution grids*, such as the one appears in Table 1.3, adopted from [Shekhar *et al.*, 1993]. The numbers in the cells of a grid are the tuples satisfying the conditions specified on the coordinates. For instance, the second cell on the first row in Table 1.3 shows that the number of tuples that satisfies `BusinessType = Petroleum` and `CargoType = RefinedOil` is 300. From the grid, the system generates two semantic rules as follows:

(BusinessType = Petroleum) $\Rightarrow$ (CargoType $\in$ {NaturalGas, RefinedOil})

(BusinessType = NonPetroleum) $\Rightarrow$ (CargoType $\in$ {NaturalGas, RefinedOil, Others})

It is obvious that given a database a combinatorial number of grids can be constructed to generate rules, and many issues need to be addressed to constrain the number of grids. Their solution, however, is to require users to specify relevant attributes, conditions and join paths as input to build the grids. The requirement creates another knowledge bottleneck problem for users because in real-world databases, there are many attributes, join paths and dynamic query patterns to be considered. It might takes significant effort to specify the required information for a large database. The performance of the generated rules will be sensitive to the quality of the given information from users. As a result, their approach is not practical. Moreover, data distribution is just one of many factors that may affect the effectiveness of rules. Again, the issue of database change is totally neglected in this approach. In brief, this approach requires careful hand-tailoring and does not address important issues of the learning problem.

The knowledge acquisition approach presented in [Yu and Sun, 1989] generates semantic rules from two equivalent or subsumable queries, where the answer of one query is a subset of the others. Unlike our approach, they do not describe any systematic approach to generating two equivalent queries. Therefore, the system must identify equivalent or subsumable queries from input queries to generate rules. This identification process could be expensive but futile, because the system needs to cache the answers of all queries, while the system may end up generating rules from two expensive queries and the learned rules are not useful in cost reduction. In addition to a rule generation approach, [Yu and Sun, 1989] also presents an approach to removing logically redundant rules. That approach could be useful to maintain the number of rules and partly address the utility problem. Similar to the other two approaches, they do not provide any solution to deal with database changes.

1.5 Summary of Contributions

This research will provide a general solution to the problem of learning for SQO. This research also addresses many important issues of autonomous learning in dynamic real-world environments. These issues include learning for both effectiveness and robustness, their interactions, and the probabilistic estimation of robustness.

The contributions are summarized as follows.

- This dissertation defines *the robustness of knowledge*, a new measure of machine-learned knowledge from information sources that change. The definition leads to an approach to estimating the robustness of Horn-clause rules in a relational database. This new measure sheds new light on applying machine learning to other database and information system problems, because it allows a learner to deal with changes to the databases.

- This dissertation presents a general learning approach to the knowledge acquisition problem of semantic query optimization. This approach addresses both the effectiveness and robustness issues of the learning problem. Another important feature of this learning approach is that it can learn relational rules that allow an optimizer to eliminate expensive relational joins in a query.
- This dissertation describes an extended semantic query optimization approach for query plans of heterogeneous multidatabase systems. Compared to previous work in semantic query optimization, this query optimization approach is more flexible because it can optimize a wider range of queries and apply a larger class of Horn-clause rules to detect more optimization opportunities.
- This dissertation provides an implementation of the resulting learning and query optimization approaches, "BASIL in PESTO," as a part of SIMS, an intelligent information mediator [Arens *et al.*, 1993, Knoblock *et al.*, 1994, Arens *et al.*, 1996], and reports the experimental results using the implementation. The results show that the learned rules produce significant cost reduction and outperform hand-coded rules. The results also show that the system provides accurate robustness estimation and thus the learning approach can maximize the net utility of learning throughout the life span of an information system.

1.6 Organization of the Dissertation

Throughout this dissertation, I have tried to adopt the standard terminology in our discussion. However, the topics of this dissertation are related to both databases and artificial intelligence, and some terms in these two fields may have different meanings, so Appendix A explains and defines informally the terminology used in this dissertation.

The remainder of this dissertation is organized as follows. Chapter 2 discusses the robustness of knowledge in a general context of machine learning applications in databases and concludes with a formal definition of robustness and an estimation approach. Chapter 3 presents the inductive learning approach for high utility semantic rules. This chapter describes how inductive learning for effective rules and rule pruning guided by the robustness estimation can be integrated to learn high utility rules for SQO. Chapter 4 presents a novel approach to the query optimization in a heterogeneous multidatabase system. Chapter 5 describes the experiments on the performance of the learning system BASIL and the query plan optimizer PESTO, the implementation of the learning and optimization approaches. This chapter presents the experimental results, including the performance statistics on using learned rules in PESTO, the robustness of learned rules, and the efficiency of the learner BASIL. The results confirm our assumption about effectiveness and robustness, and show that the learned rules are both effective and robust. Chapter 6 reviews the contributions and describes some future work.

Chapter 2

Robustness of Knowledge

Many applications of machine learning and data mining [Piatetsky-Shapiro and Frawley, 1991, Fayyad *et al.*, 1996] such as learning for semantic query optimization require the knowledge to be consistent with data. However, databases usually change over time and make machine-learned knowledge inconsistent with data. Useful knowledge should be *robust* against database changes so that it is unlikely to become inconsistent after database changes. This chapter defines this notion of robustness, and describes how robustness can be estimated.

2.1 Consistency of Rules and Database Changes

Databases are evolving entities. Knowledge discovered from one database state may become invalid or inconsistent with a new database state. Many applications require discovered knowledge to be consistent with the data. Examples are the problem of learning for semantic query optimization, learning integrated domain models of heterogeneous databases, knowledge discovery for decision support, etc. However, most approaches to these problems assume static databases, while in practice, many databases are dynamic, that is, they change frequently. It is important that discovered knowledge is *robust* against data changes in the sense that the knowledge remains valid or consistent after databases are modified.

This notion of *robustness* can be defined as the probability that the database is in a state consistent with discovered knowledge. This probability is different from predictive accuracy, which is widely used in learning classification knowledge, because predictive accuracy measures the probability that knowledge is consistent

Schema:

```
ship_class(class_name,ship_type,max_draft,length,container_cap),
ship(ship_name,ship_class,status,fleet,year_built).
geoloc(name,glc_cd,country,latitude,longitude),
seaport(name,glc_code,storage,rail,road,anch_offshore),
wharf(wharf_id,glc_code,depth,length,crane_qty).
```

Rules:

`R2.1`: *;The latitude of a Maltese geographic location is greater than or equal to 35.89.*
`geoloc(_,_,?country,?latitude,_)` $\wedge$ `?country = ''Malta''`
$\Rightarrow$ `?latitude` $\geq$ `35.89`

`R2.2`: *;All Maltese geographic locations are seaports.*
`geoloc(_,?glc_cd,?country,_,_)` $\wedge$ `?country = ''Malta''`
$\Rightarrow$ `seaport(_,?glc_cd,_,_,_,_)`

`R2.3`: *;All ships built in 1981 belong to either* `''MSC''` *fleet or* `''MSC Lease''` *fleet.*
`ship(_,_,_,?R133,?R132)` $\wedge$ `?R132 = 1981`
$\Rightarrow$ `member(?R133,[''MSC'',''MSC LEASE''])`

`R2.4`: *;If the storage space of a seaport is greater than 200,000 tons, then its geographical ; location code is one of the four codes.*
`seaport(_,?R213,?R212,_,_,_)` $\wedge$ `?R212` $<$ `200000`
$\Rightarrow$ `member(?R213,[''APFD'',''ADLS'',''WMY2'',''NPTU''])`

Table 2.1: Example rules learned from a database

with randomly selected unseen data instead of with an entire database state. This difference is significant in databases that are interpreted using the *closed-world assumption*(CWA). For a Horn-clause rule $C \leftarrow A$, predictive accuracy is usually defined as the conditional probability $Pr(C|A)$ given a randomly chosen data instance [Cohen, 1993, Cohen, 1995b, Cussens, 1993, Furnkranz and Widmer, 1994, Lavrač and Džeroski, 1994]. In other words, it concerns the probability that the rule is valid with regard to a newly inserted data. However, databases also change by updates and deletions, and in a closed-world database they may affect the validity of a rule too. Consider the rule `R2.2` in Table 2.1 and the database fragment in Table 2.2. `R2.2` will become inconsistent if we delete the seaport instance labeled with a " *" in Table 2.2, because the value 8004 for variable `?glc_cd` that satisfies the antecedent of `R2.2` will no longer satisfy the consequent of `R2.2`. To satisfy the consequent of `R2.2` requires that there is a seaport instance with its `glc_cd` value 8004, according to the closed-world assumption.

```
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 ...)
              ⋮                                             ⋮
```

Table 2.2: A database fragment

Closed-world databases are widely used partly because of the limitation of the representation systems, but mostly because of the characteristics of application domains. Instead of representing a static state of past experience, an instance of closed-world data usually represents a dynamic state in the world, such as an instance of employee information in a personnel database. Therefore, closed-world data tend to be dynamic, and it is important to handle dynamic and closed-world data when we apply learning and knowledge discovery approaches to database applications.

This chapter defines this notion of robustness, and describes how robustness can be estimated and applied in knowledge discovery systems. The key idea of our estimation approach is that it estimates the probabilities of data changes, rather than the number of possible database states, which is intractably large for estimation. The approach decomposes data changing transactions and estimates their probabilities using the Laplace law of succession. This law is simple and can bring to bear information such as database schemas and transaction logs for higher accuracy. Our experiments demonstrate the feasibility of our robustness estimation approach. The estimation approach can be used by a rule generation or maintenance system to guide the search for more robust rules so that the rules can be used with a minimal maintenance effort.

This chapter is organized as follows. The next section defines robustness. Section 2.3 describes how to estimate the robustness of a rule. Section 2.4 discusses the implementation issues of the robustness estimation. Section 2.5 demonstrates empirically the feasibility of our estimation approach. Section 2.6 compares robustness with other AI uncertainty measures. Finally, Section 2.7 summarizes contributions and potential applications of the robustness estimation.

2.2 Definitions of Robustness

This section first defines formally our notion of robustness. Intuitively, a rule is robust against database changes if it is unlikely to become inconsistent after database changes. This can be expressed as the probability that a database is in a state consistent with a rule.

Definition 2.1 (Robustness for all states) *Given a rule r, let D be the event that a database is in a state that is consistent with r. The robustness of r is* $Robust_1(r) = \Pr(D)$.

This probability can be estimated by the ratio between the number of all possible database states and the number of database states consistent with a rule. That is,

$$Robust_1(r) = \frac{\#\ of\ database\ states\ consistent\ with\ r}{\#\ of\ all\ possible\ database\ states}$$

There are two problems with this estimate. The first problem is that it treats all database states as if they are equally probable. That is obviously not the case in real-world databases. The other problem is that the number of possible database states is intractably large, even for a small database. Alternatively, we can define robustness from the observation that a rule becomes inconsistent when a transaction results in a new state inconsistent with the rule. Therefore, the probability of certain transactions largely determines the likelihood of database states, and the robustness of a rule is simply the probability that such a transaction is *not* performed. In other words, a rule is robust if the transactions that will invalidate the rule are unlikely to be performed. This idea is formalized as follows.

Definition 2.2 (Robustness for accessible states) *Given a rule r, and a database in a state denoted as d, in which r is consistent. New database states are accessible from d by performing transactions. Let t denote the event of performing a transaction on d that results in new database states inconsistent with r. The robustness of r in accessible states from the current state d is defined as* $Robust(r|d) = \Pr(\neg t|d) = 1 - \Pr(t|d)$.

This definition of robustness is analogous in spirit to the notion of *accessibility* and the *possible worlds semantics* in modal logic [Ramsay, 1988]. Definition 2.2 retains our intuitive notion of robustness, but allows us to estimate robustness without counting the intractably large number of possible database states. If the only way to change database states is by transactions, and all transactions are equally probable to be performed, then the two definitions of robustness are equivalent.

Corollary 2.3 *Given a rule r and a database in a database state denoted as d, if r is consistent with d, and if new database states are accessible from d only by performing transactions, and all transactions are equally probable, then*

$$Robust_1(r) \equiv Robust(r|d)$$

Proof: This is because the set of possible database states is exactly the same as the set of database states accessible from a current database state by transactions. They can be reached with an equal probability. □

However, it is usually not the case in real-world databases that all transactions are equally probable. The robustness of a rule could be different in different database states. For example, suppose there are two database states d_1 and d_2 of a given database. To reach a state inconsistent with r, we need to delete ten tuples in d_1 and only one tuple in d_2. In this case, it is reasonable to have

$$Robust(r|d_1) > Robust(r|d_2)$$

because it is less likely that all ten tuples are deleted. Definition 2.1 implies that robustness is a constant while Definition 2.2 captures the dynamic aspect of robustness.

2.3 Estimating Robustness

We first review a useful estimate for the probability of the outcomes of a repeatable random experiment. It will be used to estimate the probability of transactions and the robustness of rules.

Laplace Law of Succession *Given a repeatable experiment with an outcome of one of any of k classes. Suppose we have conducted this experiment n times, r of which have resulted in some outcome C, in which we are interested. The probability that the outcome of the next experiment will be C can be estimated as $\frac{r+1}{n+k}$.*

A detailed description and a proof of the Laplace law can be found in [Howson and Urbach, 1988]. The Laplace law applies to any *repeatable* experiments (e.g., tossing a coin). The Laplace law is a special case of a modified estimate called *m-Probability* [Cestnik and Bratko, 1991]. A prior probability of outcomes can be brought to bear in this more general estimate.

m-Probability *Let r, n, and C be as in the description of the Laplace law. Suppose* $\Pr(C)$ *is known as the prior probability that the experiment has an outcome C, and m is an adjusting constant that indicates our confidence in the prior probability* $\Pr(C)$. *The probability that the outcome of the next experiment will be C can be estimated as* $\dfrac{r + m \cdot \Pr(C)}{n+m}$.

The idea of *m-Probability* can be understood as a weighted average of known relative frequency and prior probability:

$$\frac{r + m \cdot \Pr(C)}{n+m} = \left(\frac{n}{n+m}\right) \cdot \left(\frac{r}{n}\right) + \left(\frac{m}{n+m}\right) \cdot \Pr(C)$$

where n and m are the weights. The Laplace law is a special case of the m-probability estimate with $\Pr(C) = 1/k$, and $m = k$. The prior probability used here is that k outcomes are equally probable. The m-probability estimate has been used in many machine learning systems for different purposes [Cestnik and Bratko, 1991, Lavrač and Džeroski, 1994]. Convincing results in handling noisy data and pruning decision trees have been achieved. The advantage of the Laplace estimate is that it takes both known relative frequency and prior probability into account. This feature allows us to include information given by a DBMS, such as database schema, transaction logs, expected size of relations, expected distribution and range of attribute values, as prior probabilities in our robustness estimation.

Our problem at hand is to estimate the robustness of a rule based on the probability of transactions that may invalidate the rule. This problem can be decomposed into the problem of deriving a set of invalidating transactions and estimating the probability of those transactions. We illustrate our estimation approach with an example. Consider `R2.1` in Table 2.3, which also lists three mutually exclusive classes of transactions that will invalidate `R2.1`. These classes of transactions cover all possible transactions that will invalidate `R2.1`. Since `T1`, `T2`, and `T3` are mutually exclusive, we have $\Pr(\mathtt{T1} \vee \mathtt{T2} \vee \mathtt{T3}) = \Pr(\mathtt{T1}) + \Pr(\mathtt{T2}) + \Pr(\mathtt{T3})$. The probability

R2.1: `?latitude` $\geq$ `35.89` $\Leftarrow$
`geoloc(_,_,?country,?latitude,_)` $\wedge$
`?country = ''Malta''.`

`T1: One of the existing tuples of geoloc with its ?country = ''Malta'' is updated such that its ?latitude` $<$ `35.89.`

`T2: A new tuple of geoloc with its ?country = ''Malta'' and ?latitude` $<$ `35.89 is inserted to the database.`

`T3: One of the existing tuples of geoloc with its ?latitude` $<$ `35.89 and its ?country` $\neq$ `''Malta'' is updated such that its ?country = ''Malta''.`

Table 2.3: Transactions that invalidate `R2.1`

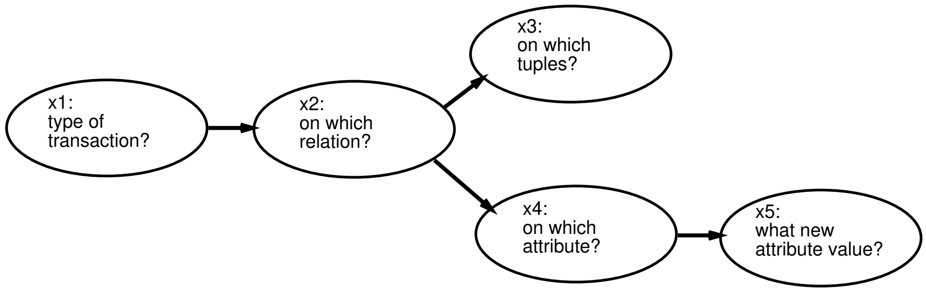

Figure 2.1: Bayesian network model of transactions

of these transactions, and thus the robustness of `R2.1`, can be estimated from the probabilities of `T1`, `T2`, and `T3`.

We require that transaction classes be mutually exclusive so that no transaction class covers another because for any two classes of transactions t_a and t_b, if t_a covers t_b, then $\Pr(t_a \vee t_b) = \Pr(t_a)$ and it is redundant to consider t_b. For example, a transaction that deletes all `geoloc` tuples and then inserts tuples invalidating `R2.1` does not need to be considered, because it is covered by `T2` in Table 2.3.

Also, to estimate robustness efficiently, each class of transactions must be minimal in the sense that no redundant conditions are specified. For example, a transaction similar to `T1` that updates a tuple of `geoloc` with its `?country = "Malta"` such that its `latitude` $<$ 35.89 and its `longitude` $>$ 130.00 will invalidate `R2.1`. However, the extra condition "`longitude` $>$ 130.00" is not relevant to `R2.1`. Without this condition, the transaction will still result in a database state inconsistent with `R2.1`. Thus that transaction is not minimal for our robustness estimation and the extra condition does not need to be considered.

We now demonstrate how Pr(`T1`) can be estimated only with the database schema information, and how we can use the Laplace law of succession when transaction logs and other prior knowledge are available. Since the probability of `T1` is too complex to be estimated directly, we have to decompose the transaction into more primitive statements and estimate their local probabilities first. The decomposition is based on a Bayesian network model of database transactions illustrated in Figure 2.1. Nodes in the network represent the random variables involved in the transaction. An arc from node x_i to node x_j indicates that x_j is dependent on x_i. For example, x_2 is dependent on x_1 because the probability that a relation is selected for a transaction is dependent on whether the transaction is an update, deletion or insertion. That is, some relations tend to have new tuples inserted, and some are more likely to be updated. x_4 is dependent on x_2 because in each relation, some attributes are more likely to be updated. Consider the relations involved in our example rules (see Table 2.1), the `ship` relation is more likely to be updated than other relations. Among its attributes, `status` and `fleet` are more likely to be changed than other attributes. Nodes x_3 and x_4 are independent because, in general, which tuple is likely to be selected is independent of the likelihood of which attribute will be changed.

The probability of a transaction can be estimated as the joint probability of all variables $\Pr(x_1 \wedge \cdots \wedge x_5)$. When the variables are instantiated for `T1`, their semantics are as follows:

- x_1: a tuple is updated.
- x_2: a tuple of `geoloc` is updated.
- x_3: a tuple of `geoloc`, whose `?country = "Malta"`, is updated.
- x_4: a tuple of `geoloc` whose `?latitude` is updated.
- x_5: a tuple of `geoloc` whose `?latitude` is updated to a new value less than 35.89.

From the Bayesian network and the chain rule of probability, we can evaluate the joint probability by a conjunction of conditional probabilities:

$$\begin{aligned}\Pr(\texttt{T1}) &= \Pr(x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5)\\ &= \Pr(x_1) \cdot \Pr(x_2|x_1) \cdot \Pr(x_3|x_2 \wedge x_1) \cdot \Pr(x_4|x_2 \wedge x_1) \cdot \Pr(x_5|x_4 \wedge x_2 \wedge x_1)\end{aligned}$$

We can then apply the Laplace law to estimate each local conditional probability. This allows us to estimate the global probability of `T1` efficiently. We will show how information available from a database can be used in estimation. When no information is available, we apply the principle of indifference and treat all possibilities as equally probable. We now describe our approach to estimating these conditional probabilities.

- *A tuple is updated:*

$$\Pr(x_1) = \frac{t_u + 1}{t + 3}$$

where t_u is the number of previous updates and t is the total number of previous transactions. Because there are three types of primitive transactions (insertion, deletion, and update), when no information is available, we will assume that updating a tuple is one of three possibilities (with $t_u = t = 0$). When a transaction log is available, we can use the Laplace law to estimate this probability.

- *A tuple of* `geoloc` *is updated, given that a tuple is updated:*

$$\Pr(x_2|x_1) = \frac{t_{u,geoloc} + 1}{t_u + R}$$

where R is the number of relations in the database (this information is available in the schema), and $t_{u,geoloc}$ is the number of updates made to tuples of relation `geoloc`. Similar to the estimation of $\Pr(x_1)$, when no information is available, the probability that the update is made on a tuple of any particular relation is one over the number of relations in the database.

- *A tuple of* `geoloc` *whose* `?country = "Malta"` *is updated, given that a tuple of* `geoloc` *is updated:*

$$\Pr(x_3|x_2 \wedge x_1) = \frac{t_{u,a3} + 1}{t_{u,geoloc} + G/I_{a3}}$$

where G is the size of relation `geoloc`, I_{a3} is the number of tuples in `geoloc` that satisfy `?country ="Malta"`, and $t_{u,a3}$ is the number of updates made on the tuples in `geoloc` that satisfy `?country ="Malta"`. The number of tuples that satisfy a literal can be retrieved from the database. If this is too expensive for large databases, we can use the estimation approaches used for conventional query optimization [Piatetsky-Shapiro, 1984, Ullman, 1988] to estimate this number.

- *The value of* `latitude` *is updated, given that the tuple that is updated is a tuple of* `geoloc` *with its* `?country ="Malta"`*:*

$$\Pr(x_4|x_2 \wedge x_1) = \frac{t_{u,geoloc,latitude} + 1}{t_{u,geoloc} + A}$$

where `A` is the number of attributes of `geoloc`, $t_{u,geoloc,latitude}$ is the number of updates made on the `latitude` attribute of the `geoloc` relation. Here we have an example of when domain-specific knowledge can be used in estimation. We can infer that `latitude` is less likely to be updated than other attributes of `geoloc` from our knowledge that it will be updated only if the database has stored incorrect data.

- *The value of* `latitude` *is updated to a value less than 35.89, given that a tuple of* `geoloc` *with its* `?country ="Malta"` *is updated:*

$$\Pr(x_5|x_4 \wedge x_2 \wedge x_1) = \begin{cases} 0.5 & \text{no information available} \\ 0.398 & \text{with range information} \end{cases}$$

Without any information, we assume that the attribute will be updated to any value with uniform probability. The information about the distribution of attribute values is useful in estimating how the attribute will be updated. In this case, we know that the latitude is between 0 to 90, and the chance that a new value of latitude is less than 35.89 should be $35.89/90 = 0.398$. This information can be derived from the data or provided by the users.

Assuming that the size of the relation `geoloc` is 616, ten of them with `?country ="Malta"`, without transaction log information, and from the example schema (see Table 2.1), we have five relations and five attributes for the `geoloc` relation. Therefore,

$$\Pr(\texttt{T1}) = \frac{1}{3} \cdot \frac{1}{5} \cdot \frac{10}{616} \cdot \frac{1}{5} \cdot \frac{1}{2} = 0.000108$$

Similarly, we can estimate $\Pr(\texttt{T2})$ and $\Pr(\texttt{T3})$. Suppose that $\Pr(\texttt{T2}) = 0.000265$ and $\Pr(\texttt{T3}) = 0.00002$, then the robustness of the rule can be estimated as $1-(0.000108+0.000265+0.00002) = 0.999606$.

The estimation accuracy of our approach may depend on available information, but even given only database schemas, our approach can still come up with some estimates. This feature is important because not every real-world database system keeps transaction log files, and those that do exist may be at different levels of granularity. It is also difficult to collect domain knowledge and encode it in a database system. Nevertheless, the system must be capable of exploiting as much available information as possible.

$$\theta(?x) \Leftarrow \bigwedge_{1 \leq i \leq I} A_i \wedge \bigwedge_{1 \leq j \leq J} B_j \wedge \bigwedge_{1 \leq k \leq K} L_k,$$

where A_i's and B_j's are database literals, $\mathcal{L}_k$'s are built-in literals.

Transaction templates:

T1: Update a tuple of A_i or B_j covered by the rule so that a new $?x$ value satisfies the antecedent but does not satisfy $\theta(?x)$.

T2: Insert a new tuple to a relation A_i or B_j so that the tuple satisfies all the antecedents but not $\theta(?x)$.

T3: Update one tuple of a relation A_i or B_j not covered by the rule so that the resulting tuple satisfies all the antecedents but not $\theta(?x)$.

Table 2.4: Templates of invalidating transactions for range rules

Deriving transactions that invalidate an arbitrary logic statement is not a trivial problem. Fortunately, most knowledge discovery systems have strong restrictions on the syntax of discovered knowledge. Hence, we can manually generalize the invalidating transactions into a small sets of transaction templates, as well as templates of probability estimates for robustness estimation. The templates allow the system to automatically estimate the robustness of knowledge. This section briefly describes the derivation of those templates.

Recall that we have defined two classes of rules based on the type of their consequents. If the consequent of a rule is a built-in literal, then the rule is a range rules (e.g., `R2.1`), otherwise, it is a relational rule with a database literal as its consequent, (e.g., `R2.2`). In Table 2.3 there are three transactions that will invalidate `R2.1`. `T1` covers transactions that update the attribute value used in the consequent, `T2` covers those that insert a new tuple inconsistent with the rule, and `T3` covers updates on the attribute values used in the antecedents. The invalidating transactions for all range rules are covered by these three general classes of transactions. We generalize them into a set of three transaction templates and express them in plain English in Table 2.4. For a relational rule such as `R2.2`, the invalidating transactions are divided into another five general classes different from those for range rules. Table 2.5 shows the transaction templates for relational rules. These two sets of templates are sufficient for any Horn-clause rules on relational data. The complete templates are presented in detail in Appendix B.

$$C(?x) \Longleftarrow \bigwedge_{1 \le i \le I} A_i \wedge \bigwedge_{1 \le j \le J} B_j \wedge \bigwedge_{1 \le k \le K} L_k,$$

`where` $C(?x)$, A_i`'s and` B_j`'s are database literals,` $\mathcal{L}_k$`'s are built-in literals.`

- `T1:` `Update an attribute of a tuple of` A_i `or` B_j `covered by the rule so that the new value allows a new` $?x$ `value satisfies the antecedents but not the consequent.`
- `T2:` `Insert a new tuple to` A_i `or` B_j `so that the new tuple allows a new` $?x$ `value satisfies the antecedents but not the consequent.`
- `T3:` `Update an attribute of a tuple of` A_i `or` B_j `not covered by the rule so that the new value allows a new` $?x$ `value satisfies the antecedents but not the consequent.`
- `T4:` `Update` $C.x$ `of all` C `tuples that share a certain` $C.x$ `value that satisfies the antecedents to a new value that does not satisfies the antecedents.`
- `T5:` `Delete all` C `tuples that share a certain` $C.x$ `value that satisfies the antecedents of the rule.`

Table 2.5: Templates of invalidating transactions for relational rules

2.4 Templates for Estimating Robustness

From the transaction templates, we can derive the templates of the equations to compute robustness estimation for each class of rules. The parameters of these equations can be evaluated by accessing database schema or transaction log. Some parameters can be evaluated and saved in advance (e.g., the size of a relation) to improve efficiency. For rules with many antecedents, a general class of transactions may be evaluated into a large number of mutually exclusive transactions whose probabilities can be estimated separately. In those cases, our estimation templates will be instantiated into a small number of approximate estimates. As a result, the complexity of applying our templates for robustness estimation is always proportional to the length of the rules.

For example, consider `R2.1.1` shown in Table 2.6. This rule is a range rule similar to `R2.1` except that there is an additional literal on the variable `?longitude` as an antecedent. Table 2.6 also shows a general class of transactions that update attribute values used in the antecedent. For `R2.1`, there is only one such attribute value, and thus we only need a minimal transaction `T3` to cover this general class (see Table 2.3). However, for `R2.1.1`, since there are two attribute values, `?country` and `?longitude`, involved in the antecedents, we have three cases for this class of transactions: updating `?country` (`T3.1`), updating `?longitude(T3.2)`, or updating

```
R2.1.1: ?latitude ≥ 35.89 ⇐
                geoloc(_,_,?country,?latitude,?longitude) ∧
                ?country = "Malta" ∧
                ?longitude > 130.00.
  T3.1: One of the existing tuples of geoloc with its ?latitude < 35.89 and its
        ?country ≠ "Malta" is updated such that its ?country = "Malta".
  T3.2: One of the existing tuples of geoloc with its ?latitude < 35.89 and its
        ?longitude ≯ 130.00 is updated such that its ?longitude > 130.00.
  T3.3: One of the existing tuples of geoloc with its ?latitude < 35.89 and its
        ?country ≠ "Malta" and ?longitude ≯ 130.00 is updated such that its
        ?country = "Malta" and ?longitude > 130.00.
```

Table 2.6: Three invalidating transactions of `R2.1.1`

both (`T3.3`), as shown in Table 2.6. In general, if there are N attribute values used in the antecedents, there will be $2^N - 1$ cases need to be considered, although many of the cases are extremely unlikely.

In our template, we ignore the cases that update more than one attribute value, and consider the cases that update just one attribute value. For `R2.1.1`, we only estimate the probability of `T3.1` and `T3.2`, but not `T3.3`. Because the class of transactions covered by `T3.3` is the intersection of those covered by `T3.1` and `T3.2`, from set theory, we have

$$\Pr(T3.1 \vee T3.2 \vee T3.3) = \Pr(T3.1) + \Pr(T3.2) - \Pr(T3.3) \leq \Pr(T3.1) + \Pr(T3.2)$$

and the estimated probability will be slightly greater than the actual probability. Therefore, the system will not underestimate the robustness and mislead the search in discovery. This approximation applies to other situations that may require large numbers of transactions to cover all possibilities.

2.5 Empirical Demonstration

We estimated the robustness of the sample rules on the database that were shown in Table 2.1. This database stores information on a transportation logistic planning domain with twenty relations. Here, we extract a subset of the data with five relations for our experiment. The database schema contains information about the number of relations and attributes in this database, as well as ranges of some attribute values. For instance, the range of `year` of `ship` is from 1900 to 1996. In addition, we also

Relation	geoloc	seaport	wharf	ship	ship_class	Total
Size	616	16	18	142	25	
Updates	0	1	1	10	1	13
Insertions	25	6	1	22	12	66
Deletions	0	2	1	10	6	19

Table 2.7: Database relation size and transaction log information

have a log file of data updates, insertions and deletions over this database. The log file contains 98 transactions. The size of relations and the distribution of the transactions on different relations are shown in Table 2.7.

Among the sample rules in Table 2.1, `R2.1` seems to be the most robust because it is about the range of `latitude` which is rarely changed. `R2.2` is not as robust because it is likely that the data about a geographical location in Malta that is not a seaport may be inserted. `R2.3` and `R2.4` are not as robust as `R2.1`, either. For `R2.3`, the fleet that a ship belongs does not have any necessary implication to the year the ship was built, while `R2.4` is specific because seaports with small storage may not be limited to those four geographical locations.

Figure 2.2 shows the estimation results. We have two sets of results. The first set shown in black columns is the results using only database schema information in estimation. The second set shown in grey columns is the results using both the database schema and the transaction log information. The estimated results match the expected comparative robustness of the sample rules.

The absolute robustness value of each rules, though, looks high (more than 0.93). This is because the probabilities of invalidating transactions are low since they are estimated with regard to all possible transactions. In situations where a set of n transactions is given and the task is to estimate the probability that a rule remains consistent after the completion of the transactions, we can use the estimated robustness of the rule ρ to estimate the probability as ρ^n, assuming that the transactions are probabilistically independent. Table 2.8 shows the estimated probabilities of consistency of the four example rules after the completion of 50 transactions. That way, the relative robustness values of the rules are normalized so that they are uniformly distributed between 0 and 1.

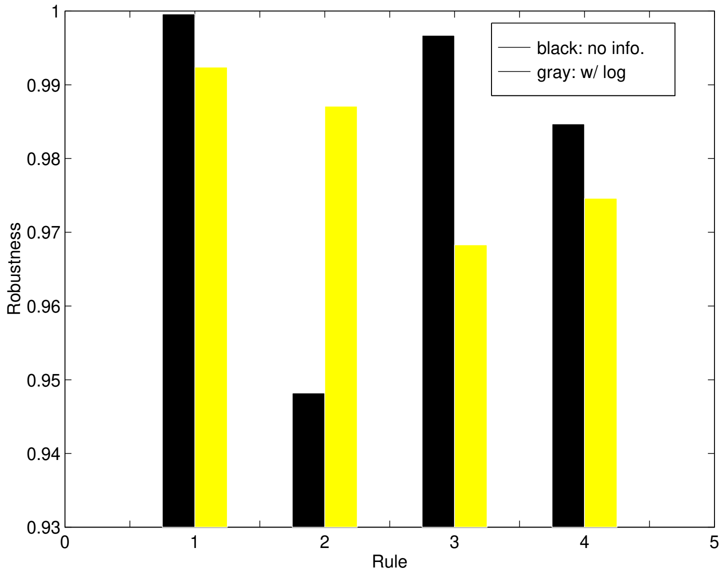

Figure 2.2: Estimated robustness of sample rules

The results show that transaction log information is useful in estimation. The robustness of R2.2 is estimated lower than other rules without the log information because the system estimated that it is not likely for a country with all its geographical locations as seaports. (See Table 2.1 for the contents of the rules.) When the log information is considered, the system increases its estimation because the log information shows that transactions on data about Malta are unlikely. For R2.3, the log information shows that the fleet of ships may change and thus the system estimated its robustness significantly lower than when no log information is considered. A similar scenario appears in the case of R2.4. Lastly, R2.1 has a high estimated robustness as expected regardless of whether the log information is used.

2.6 Related Uncertainty Measures

Robustness and predictive accuracy are closely related. For a Horn-clause rule $C \leftarrow A$, predictive accuracy is usually referred to as the conditional probability $Pr(C|A)$ given a randomly chosen data instance [Cussens, 1993, Furnkranz and Widmer, 1994, Lavrač and Džeroski, 1994, Cohen, 1993, Cohen, 1995b]. In other words, it concerns the probability that the rule is valid with regard to a newly

	R1	R2	R3	R4
Robustness (w/o log)	0.9996	0.9482	0.9967	0.9847
Prob. of consistency	0.9802	0.0699	0.8476	0.4626
Robustness (w/o log)	0.9924	0.9871	0.9683	0.9746
Prob. of consistency	0.6829	0.5225	0.1998	0.2763

Table 2.8: Robustness and probability of consistency and after 50 transactions

inserted data. This is not enough for dynamic closed-world databases where updates and deletions may affect the validity of a rule, as we discussed earlier.

In addition to predictive accuracy, [Clark and Niblett, 1989] proposed using *significance* in rule induction to measure the correlation between the antecedents and consequent of a rule by computing the likelihood ratio of the data coverage of a rule. Rough set theory [Pawlak, 1991] has been applied in many knowledge discovery applications [Ziarko, 1995] and is useful for measuring whether a given set of attributes is sufficient to represent a target concept. Like predictive accuracy, however, the significance measure and the theory of rough sets are defined with regard to data instances rather than database states, and thus do not address our problem.

Reasoning about the consistency of beliefs and knowledge after changes to closed-world relational data is an important research subject in nonmonotonic and uncertain reasoning [Ginsberg, 1987, Shafer and Pearl, 1990]. Our emphasis on transactions in our definition of robustness is analogous in spirit to the notion of *accessibility* in the *possible worlds semantics* of modal logic [Ramsay, 1988]. The formalism proposed by [Bacchus, 1988],[Halpern, 1990], and [Bacchus *et al.*, 1992, Bacchus *et al.*, 1993, Bacchus *et al.*, 1994] for uncertain reasoning, in spite of the different motivation, is quite similar to robustness. [Bacchus *et al.*, 1992] defines the *degree of belief* in a given logic sentence φ as *the probability of the set of worlds where φ is true.* They further define this probability as the ratio between the number of all possible worlds and worlds where φ is true. This is the same as Definition 2.1, if we consider a database as a model of "worlds." [Bacchus *et al.*, 1992] also surveys early philosophical work on probability that discuss related uncertainty measures.

The main difference of our work on robustness and Bacchus et al.'s work on degree of belief is in our assumption about the certainty of transactions. [1] We assume deterministic transactions and that the only change to the world is by transactions, but we do not assume that the system is certain what transaction will be performed. Therefore, we propose an estimation approach to assign the robustness of rules based on the probability of transactions. Their work, in contrast, allows nondeterministic transactions, but their system assumes that a transaction will be taken definitely and tries to figure out the probabilities of different outcomes. Both views do not capture all aspects of uncertainty but since database transactions are indeed deterministic, our assumption is more appropriate for database applications.

The research work described in [Widmer and Kubat, 1993] and [Helmbold and Long, 1994] attempts to solve the problem of learning *drifting concepts* from dynamic environments where a target concept may gradually change over time. Their solution is to incrementally modify a learned concept description to minimize its disagreement with most recently observed examples. In particular, [Helmbold and Long, 1994] shows that the error rate of a learned description is guaranteed to be smaller than a constant if the number of recent examples that are taken into account is larger than a polynomial of the error rate. The problem of learning semantic rules from changing databases can be considered as learning drifting concepts. However, to apply their results to the problem, the learner will need to take a set of recent database states to achieve a low error rate. Database states could be very large and make it impractical to take even a small number of them as input. Instead, since performing transactions is the only way to generate new database states, we can use a transaction log to inversely derive recent database states, and verify whether learned description is consistent with them. Robustness estimation uses transaction logs to predict the probability of inconsistency between learned knowledge and future database states. Since transaction logs are considerably smaller than database states, robustness estimation is more practical and appropriate for learning from large databases. It

[1]Transactions in general can be considered as actions that change world states and we will refer to actions as transactions in the following discussion.

seems that we can derive a similar bound on the number of transactions required to guarantee a small error rate. This, however, would require further investigation.

2.7 Discussion

Robustness is an appropriate and practical measure for machine learning and knowledge discovery from closed-world databases that change frequently over time. An efficient estimation approach for robustness enables effective knowledge discovery and maintenance. This chapter has defined robustness as the complement of the probability of rule-invalidating transactions, and described an approach to estimating robustness. We demonstrated empirically the feasibility of our robustness estimate approach.

Our study of robustness will allow a learning system for semantic query optimization to deal with the problem of database changes. The learner can apply the robustness estimation approach to guide the learning and maintenance of semantic rules. Based on this estimation approach, we have developed a rule pruning approach which can help to prune antecedents of a machine-generated rule into a highly robust and widely applicable rule. This approach estimates the robustness of a partially pruned rule and searches for the pruning that yields robust rules. This approach will be discussed in Section 3.4 of Chapter 3. We can also apply the robustness estimation approach to rule maintenance in a manner similar to our rule pruning approach. When detecting an inconsistent rule, the rule maintainer may propose and search for a set of rule repairing operators to repair the rule.

Robustness estimation can guide the search so that the repaired rule is more robust than the original one. As the learner collects more transaction log information, robustness estimation may be increasingly accurate and eventually minimizes the need of rule maintenance. Robustness estimation can also be applied to other AI applications for information gathering and retrieval from heterogeneous, distributed environment on the Internet. These applications require the system to extract a compressed description (e.g. a temporary concept description) of data and the consistency of the description with the database is important. Robustness can guide the system to extract robust descriptions so that they can be used with a minimal maintenance effort.

Chapter 3

Learning for Semantic Query Optimization

This chapter presents our learning approach to the knowledge acquisition problem for semantic query optimization. This approach is developed to learn operational semantic rules that are effective in cost-reduction and robust against database changes. The robustness estimation approach will be applied here to guide the learning for robust rules.

3.1 Overview of the Learning Approach

Figure 3.1 illustrates the organization of an information system equipped with an SQO optimizer and its learning system. The optimizer uses semantic rules stored in a rule bank to optimize an input query, and then sends the optimized query to a database management system (DBMS) to retrieve data. When the DBMS encounters an expensive input query, it triggers the learning system to learn a set of new rules from the data, and saves them in the rule bank to optimize subsequent queries. This way, the system will gradually collect sufficient rules for query optimization. This model of learning is similar to that in LEX [Mitchell *et al.*, 1983], which uses example problems to trigger the learning of heuristics for solving mathematical integration.

Figure 3.2 illustrates a simplified scenario for learning semantic rules. The learning system consists of two components, an inductive learning component, and a pruning component. An expensive query is given to the learner as the *training query* to trigger the learning. Receiving a training query, the learner applies an inductive learning algorithm to induce a low-cost *alternative query* equivalent to

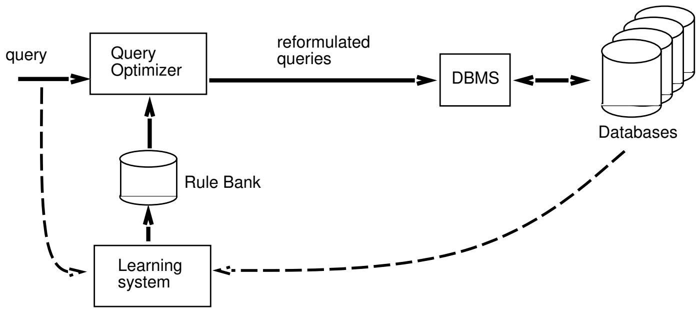

Figure 3.1: Structure of information system with SQO optimizer and learner

the training query. The pruning component then takes the training query and the learned alternative query to derive a set of semantic rules. Previously, [Yu and Sun, 1989] showed that semantic rules for SQO can be derived from two equivalent queries. However, they do not show how to automatically generate equivalent queries. Our approach can automatically induce a low-cost equivalent query for an expensive training query. The resulting rules will thus match query patterns and be effective for SQO in optimizing expensive queries into low-cost equivalent queries.

In Figure 3.2, data instances (or tuples) are labeled as positive (`+`) if they satisfy the training query and negative (`-`) otherwise. The learned alternative query must cover all positive instances but no negative instances so that it will retrieve the same data as the training query and thus is equivalent to the training query. Given a set of data instances classified as positive or negative, the problem of inducing a description that covers all 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 potentially be used in our approach. In this example, the learner constructs an alternative query with a single condition `(A1 = 'Z')` which covers the positive instance and excludes all negative instances. This alternative is less expensive since it is simpler than the input query.

Most supervised inductive learning algorithms are designed for accurate classification of unseen data instances. In our problem, however, the learning 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

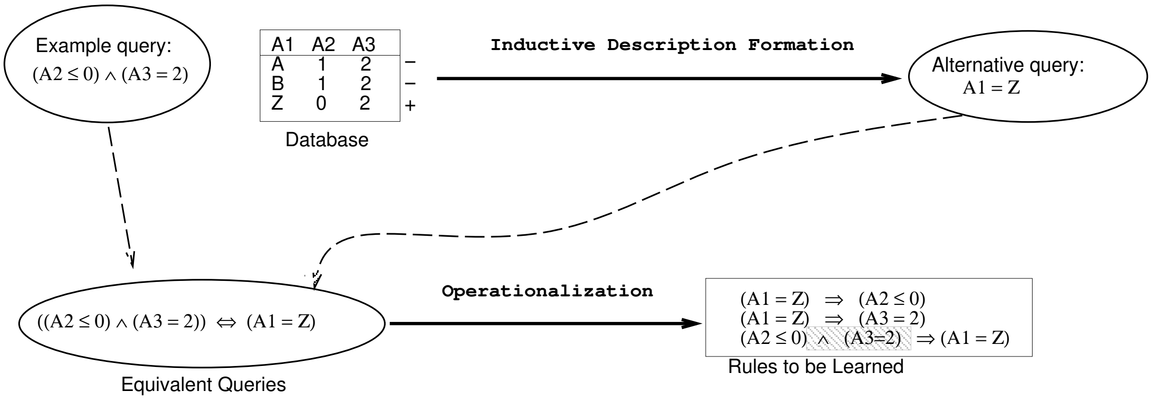

Figure 3.2: A simplified learning scenario

databases [Hsu and Knoblock, 1993a]. Section 3.2 will describe in detail a more advanced algorithm that learns conjunctive Datalog queries from relational databases. This algorithm allows the learner to learn *relational rules*, a very important class of rules for detecting redundant relational joins in the optimization. We can extend this algorithm further for databases with more advanced data models, such as object-oriented and deductive databases.

The pruning component generates operational semantic rules that allow an optimizer to deduce the equivalence of an training query and the induced alternative query. This involves two stages. The first stage is to transform the equivalence of two queries into a set of Horn-clause rules. We call this stage *operationalization* because it turns induced queries into operational rules that are ready to be used in the optimization. The second stage is to prune the antecedent literals of the resulting operational rules. The pruning is guided by the robustness estimation, so that the learner can obtain robust semantic rules. The pruning also increases the applicability of learned rules by reducing the number of antecedent literals.

Back to the example in Figure 3.2. In the operationalization stage, 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, the learner has generated rules (1), (3), and (4) that are operational for the optimization. In the second stage, the learner tries to prune the antecedents of the operationalized rules. In our example, rules (3) and (4) have only one literal as antecedent, so no further pruning will be performed. If a rule has many antecedent literals, then the learner will search for the maximally robust semantic rules by selecting a combination of the antecedent literals to prune. To prune rule (1), the learner can prune the first literal (A2 $\leq$ 0) or the second literal (A3 = 2) based on the estimated robustness of the resulting rules. During the pruning, the learner will check to see if the pruned rule is still consistent with the data and will not prune a literal if it results in an inconsistent rule. The only pruning that yields a consistent rule is to prune (A3 = 2) from rule (1). If its estimated robustness is higher than the original rule (1), the learner will perform the pruning and generate the rule

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

Note that this is an overly simplified example. In actual learning situations, the number of the antecedents of operationalized rules is usually 5 to 8 and could be as large as 20, depending on input training queries. Pruning is important for increasing the applicability of the learned rules. Also, there will be plenty of rules during the pruning for the learner to search for robust ones.

The remainder of this chapter is organized as follows. Section 3.2 describes the inductive learning algorithm for generating an alternative query given a training query. Section 3.3 presents the algorithm to transform an equivalence between two queries into operational Horn-clause rules. Section 3.4 describes the pruning algorithm. The last section discusses two potential issues of our learning approach, the utility problem and rule maintenance.

3.2 Learning Alternative Queries

The previous section has described an overview of our learning approach which consists of two component: an inductive learning component and a pruning component. This section describes the inductive learning component that inductively

```
geoloc(name,glc_cd,country,latitude,longitude),
seaport(name,glc_code,storage,rail,road,anch_offshore),
wharf(wharf_id,glc_code,depth,length,crane_qty).
```

Table 3.1: Schema of a geographic database

forms low-cost alternative queries. The scenario shown in Figure 3.2 is a simple example where the database consists of only one table. In contrast, in real-world information systems, such as a relational database, there are usually many tables of relations, and users can specify joins to associate different relations in a query. Consider a database with three relations: `person`, `car`, and `company`. A query about persons might involve the companies they work for, or the cars they drive, or even the manufacturers of their cars. An effective semantic rule should reflect these associations. Our inductive learning algorithm can learn low-cost conjunctive Datalog queries with relational joins to generate effective relational rules. This algorithm can select relevant join paths and attributes automatically instead of requiring users to do this difficult and tedious task. With relational rules, the SQO optimizer is able to delete redundant database literals or insert beneficial database literals and achieve higher optimization performance.

Before we discuss the approach, we need to distinguish two forms of constraints implicitly specified in a Datalog query. Consider the database schema for the geographic location database `Geo` in Table 3.1. Some example constraints for this database are shown as fragments of a query in Table 3.2. 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. In this case, there is only one disjunct. In another example `C1`, there are three disjuncts defined on the same attribute. An instance of `geoloc` satisfies `C1` if its `?cty` value is `"Tunisia"` or `"Italy"` or `"Libya"`. The other form of constraints 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 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 developed by [Haussler,

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

Table 3.2: Two forms of constraints used in queries

1988]. Of the many inductive learning algorithms, Haussler's was chosen because its concept description language is the most similar to database query languages. His algorithm starts 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 instance satisfies the hypothesis.

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 result of our extension is Algorithm 3.1. The input of the algorithm includes a training query and data in a given database. To explain the algorithm we use `Q3.1` shown in Table 3.3 and the database fragment in Table 2.2 as an example.

We define the *primary relation* of a query as the relation that must be accessed to answer the training query. For example, the primary relation of `Q3.1` is `geoloc` because the output variable, `?name`, of the query is bound to an attribute of `geoloc`. If there is more than one output variable and they are bound to attributes from different relations, then there will be more than one primary relation. In this case, a pre-processor can be used to partition the output variables into disjoint sets of the variables bound to attributes of the same primary relation. Then for each set of the partitioned variables, the preprocessor will generate a new query whose parameter list is the set of variables, and the query body is the same as that of the original training query. As a result, each newly generated training query will have exactly

```
Q3.1: answer(?name):-
        geoloc(?name,_,"Malta",_,_).
```

Table 3.3: Example training query

one primary relation. Therefore, we can assume that the input training query has exactly one primary relation.

In order to guarantee that the semantic rules generated by the operationalization component is consistent with data, the induced alternative query must be equivalent to a training query in the sense that they are satisfied by the same set of instances in the primary relation. We will explain why this is the case in Section 3.3 (see also [Yu and Sun, 1989]). Therefore, when identifying the primary relation and the parameter list of the training query, the learner also needs to make sure that the output variables can uniquely determine the instances in the primary relation so that the learner can determine whether the induced alternative query is equivalent to the training query by sending them to the database and comparing the answers (line 10). A straightforward approach to ensuring that the output variables can uniquely determine the instances in the primary relation is to include variables bound to the primary key of the relation in the parameter list.

Initially, the learner determines the primary relation of a training query and labels the instances in the relation as positive or negative. An instance is positive if it satisfies the training query; otherwise, it is negative. In our example, the primary relation is `geoloc` and its instances are labeled according to `Q3.1` as shown in Table 2.2. The next subsection will describe how to construct and evaluate candidate constraints, which can be either an internal disjunction or a join constraint. Then Section 3.2.2 will describe a preference heuristic to restrict the number of candidate constraints in each iteration.

3.2.1 Constructing and Evaluating Candidate Constraints

For each attribute of the primary relation, the learner can construct an internal disjunction as a candidate constraint by generalizing attribute values of positive instances. A constructed constraint is consistent with positive instances if it is

Algorithm 3.1 (Inducing an alternative query)

```
INPUT Q = training query; DB = database relations;
  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 with 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;
  UNTIL gain(c) = 0;
  RETURN fail, because no AQ is found to be equivalent to Q;
```

satisfied by all positive instances. In our example database, for attribute `country`, the learner can 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 learner considers a join constraint as a candidate constraint if it is consistent with all positive instances. Given a relation r, we can construct a join constraint over any attribute of r and another attribute of a relation in the database, but very few of them are meaningful. For example, we can construct a join between `geoloc` and `seaport` over their attributes `glc_code`. This join is meaningful, because the `glc_code` values of `geoloc` and `seaport` are defined in the same domain, and the join associates a seaport with a geographic location. In contrast, joins over the attributes `name` is not meaningful because names of geographic locations may not associate with names of seaports. To avoid constructing meaningless join constraints, the learner can limit its search to the joins specified in the training query. This way, however, the resulting alternative query will not include joins not specified in the training query and may fail to build an optimally efficient alternative query. In some databases, the information about meaningful joins is available as a part of database schema. If that is the case, the learner can use the information to construct join constraints. Otherwise, the learner will simply construct join constraints based on the joins given in the training query. In our example, suppose the learner has the information that `geoloc` can be joined with `seaport` and `wharf` over the attribute `glc_code`, it will construct join constraints based on the information.

To determine whether a join constraint can be a candidate, the learner will check whether all positive instances satisfy the join constraint. Suppose the learner is verifying whether the join constraint `C2` in Table 3.2 can be a candidate. Since for all positive instances, there is a corresponding instance in `seaport` with a common `glc_cd` value, the join constraint `C2` is satisfied and is considered as a candidate constraint that will be evaluated later.

Once we have constructed a set of candidate internal disjunctive constraints and join constraints, we need to evaluate 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 the gain for the constraints being evaluated. The gain/cost heuristic is based on the generalized problem of minimum set covering where each set is assigned a constant cost. Haussler uses this heuristic to bias the learning for short hypotheses. In our problem, we want the learner to learn a query specification with the lowest execution 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 execution 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 [Cormen *et al.*, 1989], where n is the number of input sets. 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. However, 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 expensive. We therefore use an approximation here.

The execution cost of individual constraints can be estimated using standard query size estimation techniques [Ullman, 1988]. A set of simple estimates is shown in Table 3.4. For an internal disjunction on a non-indexed attribute of a relation $\mathcal{D}$, the query execution system has to scan the entire relation to find all satisfying instances. Thus, its execution cost is proportional to $|\mathcal{D}|$, the size of $\mathcal{D}$. If the

Type of the constraints	Cost estimates
internal disjunctions, on non-indexed attribute	$\lvert\mathcal{D}\rvert$
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	$\frac{\lvert\mathcal{D}_1\rvert \cdot \lvert\mathcal{D}_2\rvert}{max(\mathcal{I}_1,\mathcal{I}_2)}$

Table 3.4: Cost estimates of constraints in a query

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 the join. Then the execution 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 processor must compute a cross product to locate pairs of satisfying instances. If the join is over indexed attributes, the execution cost is proportional to the number of instance pairs returned from the join, $\frac{|\mathcal{D}_1| \cdot |\mathcal{D}_2|}{\max(\mathcal{I}_1,\mathcal{I}_2)}$ [Ullman, 1988]. This estimate assumes that distinct attribute values distribute uniformly in instances of joined relations. If possible, the learner 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 execution 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 Table 2.2) 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 learner will select `C4` and add it to the hypothesis.

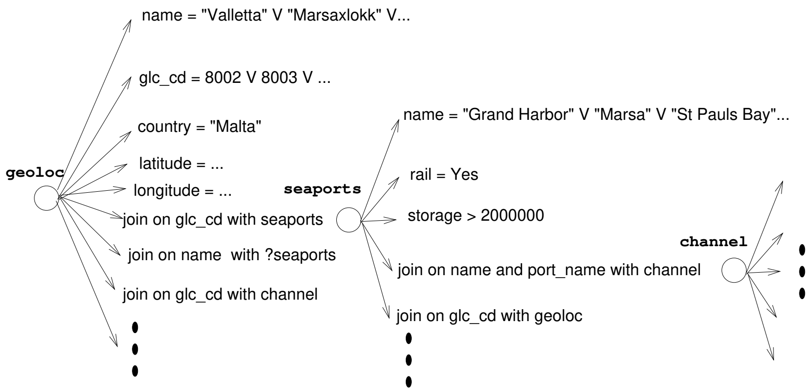

Figure 3.3: Candidate constraints to be selected

3.2.2 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 learner can consider adding constraints on attributes of the newly introduced relation to the partially constructed alternative query. 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 3.3. The expanded constraints include a set of internal disjunctions on attributes of `seaport`, as well as join constraints from `seaport` to other relations. If a new join constraint has the maximum gain/cost ratio and is selected later, the search space will be expanded further. Figure 3.3 shows the situation when a new relation, say `channel`, is selected, and the search space is expanded one level deeper. At this point, 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 learner will estimate only those candidate constraints in the newly expanded level, until the learner constructs an alternative query 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 learner 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 gives strong evidence that all useful internal disjunctions in the current level 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 the heuristic prunes out useful candidate constraints.

A time complexity analysis of Algorithm 3.1 is briefly given as follows. Let r be the largest relation in the database, A be the maximal number of attributes for a relation and J be the maximal number of meaningful joins associated with a relation. Constructing candidate constraints (line 4 and line 12 of Algorithm 3.1) requires accessing the database $O(1 + J)$ times, because constructing internal disjunctions on a relation needs one database access and each join constraint needs an additional database access. Each gain/cost evaluation requires a database access (line 6). The maximal number of new candidates that can be constructed after each new relation is introduced is $A + J$, because there are at most A internal disjunctions for each attribute and at most J join constraints. Since we constrain the search for the new candidates generated after each new relation is introduced, line 6 takes $O(A + J)$ database accesses. Therefore, in each iteration of starting from line 5, the maximal number of database accesses is bounded by $1 + J + A + J = O(A + J)$, In the worst case, each iteration only excludes a negative instance and we will need at most $|r|$ iterations to exclude all negative instances. Therefore, let d be the maximal time for a database access, Algorithm 3.1 requires $O(|r| \cdot (A + J) \cdot d)$ database accesses to complete learning in the worst case. Without search constraints, the complexity would be $O(|r|^2 \cdot (A + J) \cdot d)$ because the number of candidate constraints needed to be evaluated may increase by $(A + J)$ in each iteration. We note that $|r|$ could be as large as 10,000 in real applications.

Returning to the example, since the candidate constraint `C4` was selected, the learner will expand the search space by constructing consistent internal disjunctions and join constraints on `seaport`. Assuming that the learner 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 3.3). Now, all

```
Q3.2: answer(?name):-
        geoloc(?name,?glc_cd,"Malta",_,_),
        seaport(_,?glc_cd,_,_,_,_).

Q3.1 ⇔ Q3.2:
  geoloc(?name,_,"Malta",_,_)
  ⇔ geoloc(?name,?glc_cd,"Malta",_,_) ∧
     seaport(_,?glc_cd,_,_,_,_).

Learned rule:
  R3.1: geoloc(_,?glc_cd,"Malta",_,_)
        ⇒ seaport(_,?glc_cd,_,_,_,_).
```

Table 3.5: Results of learning given the example training query `Q3.1`

negative instances are excluded. the learner thus learns the query `Q3.2` shown in Table 3.5. The pruning component will then take the equivalence of the input query `Q3.1` and the learned query `Q3.2` as input and will transform this statement into a new operational rule that can be used to reformulate `Q3.1` into `Q3.2`. Since the size of `geoloc` is considerably larger than that of `seaport` (30,000 vs. 800), next time when a query asks about geographic locations in Malta, the optimizer can reformulate the query to access the `seaport` relation first and save the cost of scanning the entire `geoloc` relation.

We can apply the result of the gain/cost heuristic of the minimal set covering to analyze the asymptotic quality of alternative queries. The problem of minimal set covering is to search for a minimal-cost set cover containing a collection of sets whose union covers a given collection of elements. As we have discussed earlier, the cost of a set cover is the sum of the cost of each set and the cost of each member set is constant. [Cormen *et al.*, 1989] shows that the gain/cost heuristic allows an algorithm to generate a set cover within a small ratio bound $(\ln n + 1)$ of the optimal set covering cost, where n is the number of input sets. Based on this result, given a training query, if the execution cost of a query is proportional to the sum of the execution cost of each literal, we can have the same ratio bound between the cost of the optimal equivalent query and alternative query generated by Algorithm 3.1. In this case, the ratio bound is $(\ln l + 1)$ where l is the number of candidate constraints considered during the learning. Nevertheless, this is an approximation because usually query execution cost is not additive with regard to the cost of literals.

3.2.3 Related Work in Rule Induction and Data Mining

Inductive Logic Programming (ILP) [Muggleton and Feng, 1990, Quinlan, 1990, Lavrač and Džeroski, 1994, Raedt and Bruynooghe, 1993] is closely related to the problem of learning alternative queries. 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. A difference between our approach and FOIL is 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-gain heuristic while our approach uses a set-covering heuristic for learning a low-cost specification.

Research work on data mining for association rules [Agrawal *et al.*, 1993, Mannila *et al.*, 1994] is related to our work in that they also generate rules from large databases. 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 learner 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.

Algorithm 3.2 (Operationalize equivalent queries)

INPUT Q_i = training query, Q_a = alternative query;
  LET P = primary database literal of Q_a; R = empty;
  FOR each literal L in Q_a except P
    LET L_{jp} = the shortest join path from P to L;
    LET consequent of r = L;
    LET antecedent of r = (literals in Q_i) $\cup$ (L_{jp} $-$ L);
    IF antecedent of r $\not\models$ L THEN add r to R;
  RETURN R

3.3 Operationalization

Once the learner induces an alternative query from the given data, it needs to transform the equivalence of two queries into operational rules. This transformation problem can be generalized as follows. Given a statement of the equivalence of two conjunctive formulas:

$$P_1 \wedge \ldots \wedge P_m \Leftrightarrow Q_1 \wedge \ldots \wedge Q_n,$$

where P_i's and Q_i's are literals, deductively transform the statement into a set of Horn-clause rules (implications with exactly one literal as the consequent). The transformed implication rules must be the logical consequences of the equivalence. If the conjuncts in the statement are propositional, then the transformation is simply to decompose a two-directional implication into a set of one-directional implications for each literal:

$$P_1 \wedge \ldots \wedge P_m \Rightarrow Q_i \text{ , for } 1 \leq i \leq n$$
$$Q_1 \wedge \ldots \wedge Q_n \Rightarrow P_j \text{ , for } 1 \leq j \leq m$$

However, if the conjuncts are predicate logic literals that involve variables and relations, like the database queries discussed here, then care must be taken to arrange variables during the transformation. This section describes how to transform an equivalence of relational queries into operational rules.

Algorithm 3.2 lists the top level algorithm of the operationalization. This algorithm takes a training query and the corresponding learned alternative query as input, and then for each literal in the alternative query, generates a Horn-clause rule with the literal as the consequent. The same algorithm can be applied to generate rules with the literals in the training query as the consequents.

Training query:

```
Q3.3: answer(?name,?code):-
 1      seaport(?name,?code,_,_,_,_) ∧
 2      wharf(_,?code,?depth,_,?crane) ∧
 3      geoloc(?name,_,?country,_,_) ∧
 4      ?country = "Malta" ∧
 5      ?depth ≤ 50 ∧
 6      ?crane > 0.
```

Alternative query:

```
Q3.4: answer(?name,?code):-
 7      seaport(?name,?code,_,_,_,_) ∧
 8      wharf(_,?code,_,?length,_) ∧
 9      ?length ≥ 1200.
```

Table 3.6: Equivalent queries to be operationalized into Horn-clause rules

The main task of the algorithm is to compute a *join path* for each literal from the primary relation. A *join path* in a query from a database literal L_d to another literal L is a sequence of literals $(L_d, \ldots, L)$ so that for any two consecutive database literals in this sequence, there is a common variable , or a relational join, occurring in both database literal. A join path indicates how one literal associates with another literal by joins of relations. For example, a join path between literal 2 and literal 3 of the query in Table 3.6 is (literal 2, literal 1, literal 3), because literal 2 and literal 1 share the common variable `?code`, and literal 1 and literal 3 share the common variable `?name`.

We use the example in Table 3.6 to explain the algorithm. Suppose that `Q3.3` is a training query and `Q3.4` is the induced alternative query by the learner for `Q3.3`. The primary relation in this case is `seaport`, and the primary database literal is literal 1 and literal 7 for `Q3.3` and `Q3.4`, respectively. The loop from line 3 to line 8 attempts to generate a rule for each literal in each iteration. To generate a rule with literal 9 as the consequent, the system first computes the shortest *join path* from literal 7 to literal 9. In this case, the resulting join path is (literal 7, literal 8, literal 9). This join path shows how a range constraint on `?length` can be associated with `?name` and `?code` values of `seaport`. Next, the literals on the join path except the last literal will be combined with the literals of `Q3.3` to form the antecedent and generate a new rule. Duplicate database literals that refer to the same instances in a

```
R3.2: ?length ≥ 1200 ⇐
 1      wharf(_,?code,?depth,?length,?crane) ∧
 2      seaport(?name,?code,_,_,_,_) ∧
 3      geoloc(?name,_,?country,_,_) ∧
 4      ?country = "Malta" ∧
 5      ?depth ≤ 50 ∧
 6      ?crane > 0.

R3.3: geoloc(_,?code,_,_,_) ⇐
 7      wharf(_,?code,_,_,_) ∧
 8      seaport(?name,?code,_,_,_,_) ∧
 9      ?name = "Long Beach".
```

Table 3.7: Example rules to be pruned

relation will be removed from the resulting antecedent. In this example, since literal 7 and literal 1 are exactly identical, and literal 8 and literal 2 are defined on the same relation `wharf`, and the common variable `?code` bound to the key attribute of `wharf` so that the two literals refer to the same `wharf` instances. Therefore, neither literal in the join path is included in the antecedent. Line 8 of the algorithm is to exclude tautologies. The resulting rule is shown as `R3.2` in Table 3.7.

We note that to guarantee that the rules generated from Algorithm 3.2 are consistent with the database, the input queries must return the same answer. Furthermore, the answer must uniquely determine the instances in their primary relation. Otherwise, even though two queries returns the same answer, they might be satisfied by two different sets of instances of the primary relation. If that is the case, we can not assert the implications between the constraints specified in the two queries. This is why we require the output variables of a training query must uniquely determine the instances of the primary relation.

3.4 Pruning Rules for High Utility

Since training queries tend to have many literals, the rules generated from the operationalization are usually too long and overly specific to a training query. For instance, `R3.2` contains six literals as the antecedent, which is unlikely to be applicable to queries other than `Q3.3`. The section presents a rule pruning approach which can increase the applicability of the learned rules by pruning the antecedent literals. The pruning is guided by robust estimation so that the resulting rules are more widely applicable than the original rule as well as more robust. This rule pruning approach can be applied on top of other rule induction and data mining systems to prune overly specific rules into highly robust and applicable rules.

3.4.1 Background and Problem Specification

Although robustness is a desirable property of machine-generated knowledge, using robustness alone is not enough to guide the learning. The tautologies such as

```
False ⇒ seaport(_,?glc_cd,_,_,_,_),  and
seaport(_,?glc_cd,_,_,_,_) ⇒ True
```

are extremely robust (have a robustness equal to one), but they are not useful. Therefore, we should use robustness together with other measures of usefulness to guide the learning. One of the measures of usefulness is applicability, which is important no matter what our application domains are. This section focuses on the problem of pruning learned rules so that they are both highly applicable and robust. In particular, we will use length to measure the applicability of rules. Generally speaking, a rule is more applicable if it is shorter. In other words, if the number of antecedent literals of a rule is smaller, then it is more widely applicable because it is less specific.

In addition to our inductive learning approach for SQO, inductive logic programming also concern the problem of generating Horn-clause classification rules from data represented in relations similar to those in relational databases. However, the learned rules are usually too specific and not robust against database changes. Instead of generating desired rules in one run, we propose using these existing algorithms to generate rules, and then use a rule pruning algorithm to prune the antecedent literals so that it is highly robust and applicable (short). The rationale is that rule construction algorithms tend to generate overly-specific rules, but taking the length and robustness of rules into account in rule construction could be too expensive. This is because the search space of rule construction is already huge and evaluating robustness is not trivial. PRODIGY-EBL [Minton, 1988], a speedup learning approach for problem solving, includes a *compressor* to prune learned rules. Previous work in classification rule induction [Cohen, 1993,

Algorithm 3.3 (Pruning rule literals)

```
INPUT R = rules (initially the rule to be pruned), B = beam size;
  LET O = results; (initially empty);
  WHILE (R is not empty) DO
    move the first rule r in R to O;
    prune r, LET R' = resulting rules;
    remove visited, dangling or inconsistent rules in R';
    estimate and sort on the robustness of rules in R';
    retain top B rules in R' and remove the rest;
    merge sorted R' into R in sorted order of the robustness;
  RETURN O;
```

Cohen, 1995b, Furnkranz and Widmer, 1994] also shows that dividing a learning process into a two-stage rule construction and rule pruning can yield better results in terms of classification accuracy as well as the efficiency of learning. These results may not apply directly to our rule induction problem, nevertheless, a two-stage system is simpler and more efficient. Another advantage is that the pruning algorithm can be applied on top of existing rule generation systems.

The specification of our rule pruning problem is as follows: take a machine-generated rule as input, which is consistent with a database but potentially overly-specific, and remove antecedent literals of the rule so that it remains consistent but short and robust.

3.4.2 The Pruning Algorithm

Algorithm 3.3 searches for a subset of antecedent literals to remove until any further removal will make the rule inconsistent with the database. Since the search space can be exponentially large with respect to the number of literals in a rule, and checking the consistency of a partially pruned rule needs a database access, which could be expensive, we present a beam-search algorithm to trim the search space.

The algorithm applies the estimation approach described in Chapter 2 to estimate the robustness of a partially pruned rule and guide the pruning search. The main difference of our pruning problem from previous work is that there is more than one property of rules that the learner is trying to optimize, and these properties — robustness and length — may interact with each other. In some case, a long rule may be more robust, because a long rule is more specific and covers fewer instances in the database. These instances are less likely to be selected for modification, compared to the case of a short rule, which covers more instances. On the other hand, since a long rule has more literals, it is more likely that a simple change would violate one of the literals and make the rule inconsistent. To address this issue, we propose a beam search algorithm so that for each set of equally short rules, the algorithm will search for the rule that is as robust as possible while still being consistent. We will use rules `R3.2` and `R3.3` shown in Table 3.7 to illustrate how the algorithm works.

Let B denote the input beam size, Algorithm 3.3 expands the search by pruning one literal from the input rule in each search step (starting from line 3), preserves the top B robust rules, and repeats the search until no further pruning is possible. The pruner will return all rules being expanded and then we can use an additional filter to selects those with a good combination of length and robustness. The selection criterion may depend on how often the application database changes.

In line 6 of Algorithm 3.3, the pruner removes the pruned rules that are inconsistent. To identify an inconsistent rule, the pruner can consult the database directly. The pruner also discards those pruned rules with any *dangling literals*. In a rule, a set of literals are dangling if the variables occurring in those literals do not occur in any other literals (including the parameter list). For example, in the following rule, P_2`(?z,?w)` is dangling:

Q`(?x)` $\Leftarrow$ P_1`(?x,?y)`,P_2`(?z,?w)`,`?y > 100`.

Dangling literals are not desirable because they may mislead the search and complicate the robustness estimation. Removing a built-in literal in a query never results in dangling literals. To ensure that removing a database literal L in the rule does not yield dangling literals, L must satisfy the following conditions:

1. No built-in literal in the antecedents of the rule is defined on the variables occurring in L.

2. If a variable occurring in the consequent of r also occurs in L, this variable must occur in some other database literals in the rule.

3. Removing L from the rule does not disconnect existing join paths between any database literals in the rule.

We use examples to explain these conditions. Condition 1 is clear, because otherwise, there will be a dangling built-in literal. For Condition 2, consider literal

Rule	Antecedents (abbr.)	Robustness	Remarks
R3.2	W S G Cr D Ct	0.9784990	
r1	W S G D Ct	0.9814620	
r2	W S G Cr Ct	0.9784990	
r3	W S G Cr D	0.9784991	
r4	W S Cr D		Inconsistent
r5	W S G Ct	0.9814620	
r6	W S G D	0.9814620	
r7	W S G Cr	0.9896200	
r8	W D C		Inconsistent
r9	W S Cr		Inconsistent
r10	W S G	0.9814620	
r11	W S D		Inconsistent
r12	W G Cr		Dangling
r13	W G D		Dangling
r14	W Cr		Inconsistent
r15	W S		Inconsistent
r16	W G		Dangling
r17	W D		Inconsistent
r18	W		Inconsistent

Table 3.8: Result of rule pruning on a sample rule

1 of `R3.2` in Table 3.7. It is not removable because the variable `?length` in this literal is used in the consequent. But literal 7 in `R3.3` is removable, even though its variable `?code` is used in the consequent. This is because `?code` also occurs in literal 8 of the same rule and the variable can still be associated with the antecedents. An example where a literal is not removable due to Condition 3 is literal 2 of `R3.2`. This literal is not removable because the join path between literal 1 and literal 3 will be disconnected if we remove it, and as a result, literal 3 will be dangling. Therefore, literal 2 is not removable. Note that if later the pruner removes literal 3 from `R3.2` first, literal 2 will become removable because no join path would be disconnected if it were dropped.

3.4.3 Empirical Demonstration of Rule Pruning

We conducted a detailed empirical study on rule `R3.2` using the same database as in Section 2.5. Since the search space for this rule is not too large, we ran an exhaustive search for all pruned rules and estimated their robustness. The entire

```
r7: ?length ≥ 1200 ⇐
        wharf(_,?code,?depth,?length,?crane) ∧
        seaport(?name,?code,_,_,_,_) ∧
        geoloc(?name,_,?country,_,_) ∧
        ?crane > 0.

r10:?length ≥ 1200 ⇐
        wharf(_,?code,?depth,?length,?crane) ∧
        seaport(?name,?code,_,_,_,_) ∧
        geoloc(?name,_,?country,_,_).
```

Table 3.9: Pruned rules

search process took less than a second (0.96 seconds). In this experiment, we did not use the transaction log information in the robustness estimation.

The results of the experiment are listed in Table 3.8. To save space, we list the pruned rules with their abbreviated antecedents. Each term represents a literal in the conjunctive antecedents. For example, "W" represents the database literal on `wharf` (literal 1 in Table 3.7), "Cr" and "Ct" represent the literals on `?crane` and `?country`, respectively. Inconsistent rules and rules with dangling literals are identified accordingly. In this example, the pruner detected three pruned rules with dangling literals.

The relationship between length and robustness of the pruned rules is illustrated in Figure 3.4. The best rule will be the one located in the upper right corner of the graph, with short length and high robustness. On the top of the graph is the shortest rule `r10`, whose complete specification is shown in Table 3.9. Although this is the shortest rule, it is not desirable because it is too general. The rule states that wharves in seaports will have a length greater than 1200 feet. However, we expect that there will be data on wharves shorter than 1200 feet. Instead, with the robustness estimation, the pruner can select the most robust rule `r7`, also shown in Table 3.9. This rule is not as short but still its length is short enough to be widely applicable. Moreover, this rule makes more sense in that if a wharf is equipped with cranes, it is built to load/unload heavy cargo carried by a large ship, and therefore its length must be greater than some certain value. Finally, this pruned rule is more robust and shorter than the original rule. This example shows the utility of the rule pruning with the robustness estimation.

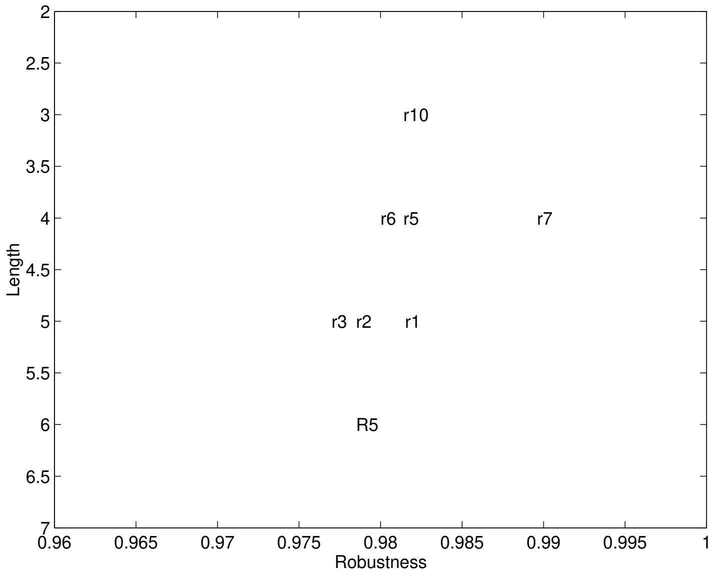

Figure 3.4: Pruned rules and their estimated robustness

3.5 Utility Problem and Rule Maintenance

This section discusses the utility problem that may arise when we consider the collective effect of learned semantic rules in query optimization. The utility problem, originally identified in [Minton, 1988], refers to situations when learning degrades the performance of a system. Though this problem is identified in problem solving systems that apply explanation-based learning (EBL) [Mitchell *et al.*, 1986, DeJong and Mooney, 1986, Rosenbloom and Laird, 1986, Minton *et al.*, 1989], it may arise in our learning problem because a similar rule application approach is used for performance improvement. The utility problem may arise in a rule-based system because the utility of a rule is determined not only by its saving and applicability, but also by the cost for the system to locate it. This observation leads to the following formula of utility:

Utility of Search Control Rules [Minton, 1988]

$$Utility = Average\_Saving \times Application\_Frequency - Match\_Cost \tag{3.1}$$

According to this formula, we need to decrease the match cost to increase the utility of a rule, otherwise, the utility problem may arise. Roughly speaking, the match cost may increase either because there are too many rules (average growth effect [Tambe and Rosenbloom, 1993]) or because individual rules are too complex to be matched efficiently. Many approaches have been developed to decrease the match cost and address the utility problem. They can be classified into two general approaches. One approach is to organize rules so that an applicable rule can be efficiently matched. For example, the RETE algorithm [Forgy, 1982] and its more advanced descendants [Tambe, 1991, Doorenbos *et al.*, 1992] are in this category. The other approach is to use a rule maintainer to remove redundant and low utility rules so as to decrease the number of rules. Redundant rules are logical consequences of other rules in a rule set. [Greiner and Likuski, 1989] and [Etzioni, 1992] show that redundant macro-operators are guaranteed to *slow down* a problem solver. [Yu and Sun, 1989] extends the SQO optimizer to identify logically redundant rules. Low utility rules are those rarely applicable or yield small saving. A simple approach to removing this class of rules is to monitor the saving yielded by rules and remove those with small saving [Minton, 1988].

The utility problem in SQO can be alleviated by an alert query optimizer that can give up its attempt to match more rules and send the original query to the database system. This strategy is adopted in [Shekhar *et al.*, 1988] so that the query execution cost is always at most as high as or slightly higher than the cost without optimization. Therefore, even in the worst case an SQO optimizer will not slow down the query execution significantly. Our learning approach may constrain the the number of learned rules by triggering the learning selectively, and the pruning approach may reduce the complexity of rules and thus the match cost of an individual rule. If the match cost is still so high to cause the utility problem, we can apply the approaches surveyed in this section to our query optimizer.

Another issue that we have not addressed is rule maintenance. Although learned semantic rules are robust, they may not be invariants and some of them may become inconsistent in a new database state. We can use a rule maintainer to repair inconsistent semantic rules. The simpliest approach to repairing inconsistent rules is to discard them, because our learner can learn robust rules and only few of the rules will become inconsistent after database changes. After removing inconsistent rules, the optimizer will still have sufficient rules for query optimization and minimize the need of learning from similar queries that have been used in training. In other rule-based applications where consistency of rules is important but the learned rules are not robust, we may need a more sophisticated rule maintainer. As an additional advantage, the robustness estimation approach can also be used to guide the rule repairing. A rule repairing system can estimate the robustness of a partially repaired rule and search for the one that is the most robust. With the robustness estimation, an inconsistent rule can be repaired into a robust rule and eventually the need for repairing will be reduced.

Chapter 4

Semantic Optimization of Query Plans

Semantic rules generated by the learning approach described in the previous chapter can be applied in semantic query optimization for a stand-alone database system as well as an information system with multiple sources. New applications of information systems, such as electronic commerce and healthcare information systems, need to integrate heterogeneous information sources. A promising solution to this problem is through the use of information mediators that can automatically generate a query plan to retrieve and combine data from heterogeneous information sources. Existing query optimization techniques can provide local optimization for a query plan by optimizing subqueries to individual databases. However, due to the complexity of queries and the heterogeneity of databases, it is difficult for these techniques to provide global optimization. This chapter presents an extended semantic query optimization approach that uses learned semantic rules to optimize a query plan and describes how our learning approach can support this new query optimization approach.

4.1 Information Mediators and Query Plans

Integrating heterogeneous multidatabases is an important problem for the next generation information systems. A wide-area health-care information systems, for example, would require integrating many different types of information for physicians in the course of their work. A promising approach to integrating heterogeneous multidatabases is through the use of *information mediators* [Wiederhold, 1992, Arens *et al.*, 1993, Knoblock *et al.*, 1994, Levy *et al.*, 1995, Hammer *et al.*, 1995, Arens *et al.*, 1996] that can select appropriate databases over which to retrieve data and generate an efficient plan to combine the data automatically for users. With information mediators, users can access heterogeneous databases without knowing the implementation details such as their locations, query languages, platforms, etc.

Given a declarative query, the main task of an information mediator is to generate a correct and efficient query plan to retrieve and combine data. Generally speaking, a *query plan* is a graph with its nodes as plan steps and its edges as the order constraints on the plan steps. Each plan step contains a subquery to be sent to a source database site where the data will be retrieved or manipulated. The order constraints of a plan specifies data flow directions as well as the order in which the plan steps should be executed.

For example, suppose an information mediator receives a query as follows:

> Query: Retrieve the classes of active ships with container capability that can dock in the wharves with cranes at Long Beach seaport; list by ship class name and wharf id.

To be precise, a ship can dock in a wharf if the wharf is long and deep enough to accommodate the ship. Suppose the data required to answer this query are spread over two remote databases: `Geo` for the data about geographical locations, seaports and wharves, and `Assets` for ships, ship classes, aircrafts, etc. The schema of these databases is given in Table A.1. Given this query, the information mediator will generate a query plan illustrated in Figure 4.1, which shows a query plan as a diagram of subqueries and their data flow order.

In this example query plan, the first subquery retrieves data about ships from the remote database `Assets`, the second subquery retrieves data about seaports and wharves from database `Geo`, and the third subquery compares data retrieved by the previous subqueries and joins the results. A subquery begins with a predicate name as the site that the data will be retrieved, followed by a list of parameters. For example, Subquery 1 begins with the site name `assets` followed by the parameters `?ship_class`, `?draft` and `?length`. To execute this query plan, the mediator will send the first two subqueries to their corresponding remote database servers to retrieve data, and then join the partial results in the local system according to the conditions specified in Subquery 3. Each remote database server has a *wrapper*

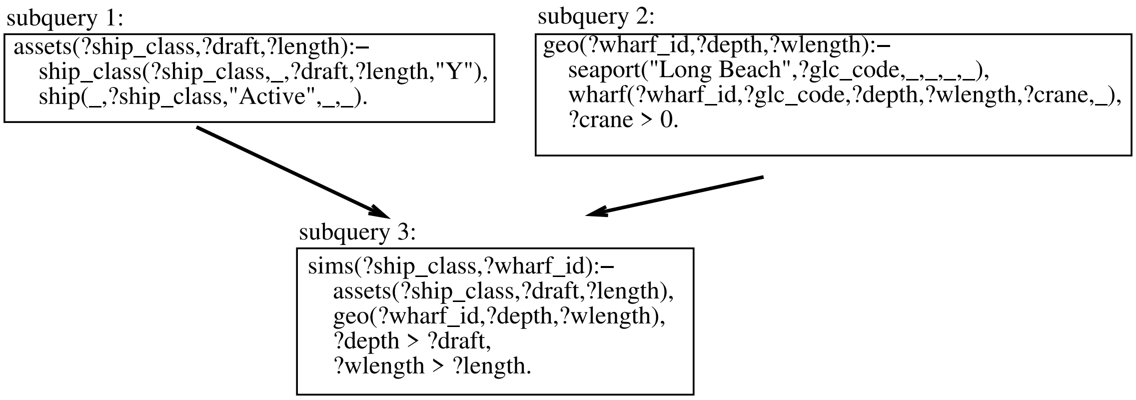

Figure 4.1: Example query plan that retrieves heterogeneous multidatabases

that will translate a subquery into the query language, say, SQL, of that particular server, and translate the retrieved data into a format that is readable by the mediator. This allows the information mediator to retrieve and combine the data from heterogeneous sources using subqueries expressed in some uniform language. In our example, this uniform language is Datalog.

The query plan is optimal in terms of the number of subqueries because the desired data are spread over two databases `Assets` and `Geo` and those subqueries cannot be merged. However, this plan could still be expensive because the system needs to retrieve and transmit lots of redundant ship class and wharf data that will eventually be discarded when executing Subquery 3. Also in Subquery 1, the system needs to execute a join over the large `ship` relation. Conventional query optimizers [Apers *et al.*, 1983, Jarke and Koch, 1984, Ullman, 1988] can be applied to optimize individual subqueries, but still, the amount of redundant intermediate data would not be reduced.

This chapter presents an extension of semantic query optimization that can complement conventional techniques to overcome the heterogeneity and considerably reduce redundant data transmission. The remainder of this chapter describes our optimization approach. The next section reviews the basic semantic query optimization algorithm, which is used as a part of our optimization approach. Section 4.3 describes the new optimization approach. Section 4.4 compares our approach with related work in query optimization. Section 4.5 discusses the issues of applying our learning approach to support query plan optimization. The last section reviews the contributions.

4.2 The Basic SQO Algorithm

The goal of semantic query optimization is to search for the optimal semantically equivalent query of an input query based on the given semantic knowledge. Two queries are defined to be *semantically equivalent* if they return identical answers from a database state that is consistent with the semantic knowledge. Generally speaking, the basic semantic query optimization consists of two stages:

1. proposing sequences of one or more reformulation operations (e.g., to delete a literal or insert a new literal) from semantic knowledge; and

2. selecting and applying the optimal reformulation based on a cost model of query execution.

These two stages may be repeated until the optimizer determines that it has found the optimal equivalent query [Chakravarthy *et al.*, 1990, King, 1981, Shekhar *et al.*, 1988, Siegel, 1988]. However, such a generate-and-test algorithm may miss applicable rules and hence miss optimization opportunities because an applicable rule to a query may become inapplicable if some literals are deleted from the query. To address this problem, when the optimizer detects an redundant literal, instead of deleting the literal immediately, it should retain the literal and delay the deletion until all applicable rules have been located. This can be achieved by computing an *implication closure* of semantic knowledge to propagate the results of rule applications [Sun and Yu, 1994, Hsu and Knoblock, 1993b, Shenoy and Ozsoyoglu, 1989, Yu and Sun, 1989]. An implication closure contains all redundant literals existing in an input query that are implied by other literals and all of the new literals derived from semantic rules. This way, the optimizer can consider all possible literal deletions and insertions implied by the semantic rules and will not miss any optimization opportunities.

Algorithm 4.1 lists the abstract-level steps of the state-of-the-art semantic query optimization algorithm. We illustrate how this algorithm works using the semantic

Algorithm 4.1 (Basic semantic query optimization)

```
 1 INPUT Q = conjunctive query; KB = semantic knowledge;
 2   LET I = implication closure (initially empty);
 3   derive the most restrictive ranges of the variables in Q
       using the range facts in KB;
 4   update I and Q with the derived range constraints;
 5   IF a literal in Q contradicts a range fact THEN RETURN NULL;
 6   FOR all applicable rules in KB
 7     LET A → B be the applicable rule after variable substitution;
 8     IF Q refuted THEN RETURN NULL;
 9     ELSE add B to I, LET Q = Q ∪ {B};
10   search for a subset D of literals in I, so that Q' = Q − D is optimal;
11   RETURN Q' and I;
```

knowledge in Table 4.2 and the example query `Q4` in Table 4.1 as input. This query retrieves all ship classes and their maximal draft so that their container capability is ``Y'', their draft is less than 50 feet, and that there is at least one active ship in this class.

Initially, the optimizer uses range facts and the literals in the query to derive the most restrictive ranges of the variables in the query (line 3 to 5). For example, from the range fact `F1` and the literal `?draft < 50`, the optimizer derives that the most restrictive range of the variable `?draft` is the interval $[12, 50)$ and inserts a new literal `?draft >= 12` to the query. Similarly, the optimizer derives ranges of other variables and inserts two new literals on `?length` to the query. The optimizer also saves the derived literals in the implication closure I.

Next, from the semantic rules, the optimizer derives more new literals as well as redundant literals and saves the results in the implication closure (line 6 to 9). In this example, the optimizer detects that the literal on the database relation `ship` is redundant because from `R4.1` in Table 4.2, the literal `?status = "Active"` is redundant, and because the relational rule `R4.2` states that if a ship class has container capability (specified as `"Y"`), then there must exist some ships in that ship class. That is, the join over the relations `ship_class` and `ship` on the variable `?class` is redundant and can be eliminated [1]. The optimizer also derives a new

[1] See [Sun and Yu, 1994] for a detailed discussion on detecting redundant relational joins in semantic query optimization.

```
Q4: assets(?ship_class,?draft):-
       ship_class(?ship_class,_,?draft,_,?container),
       ship(_,?ship_class,?status,_,_),
       ?status = "Active",
       ?container = "Y",
       ?draft < 50.
```

stage 1:

```
Q4.1: assets(?ship_class,?draft):-
       ship_class(?ship_class,?length,?draft,_,?container),
       ship(_,?ship_class,?status,_,?year-built),
       ?length >= 580,
       ?length <= 950,
       ?status = "Active",
       ?year-built > 1945,
       ?draft >= 12,
       ?container = "Y",
       ?draft < 50.
```

stage 2:

```
Q4.2: assets(?ship_class,?draft):-
       ship_class(?ship_class,_,?draft,_,?container),
       ?container = "Y",
       ?draft < 50.
```

Table 4.1: Example results of semantic query optimization in different stages

literal `?year-built < 1945` from `R4.3`. The resulting query is given as `Q4.1` in Table 4.1, where the underlined literals are those derived from the semantic knowledge and saved in the implication closure.

The optimizer then searches for a subset of the literals in the implication closure to retain and deletes the rest so that the resulting query is the least expensive (line 10). Since it could be difficult to identify the optimal combination in all cases (an $\mathcal{NP}$-complete problem [Sun and Yu, 1994]), the optimizer needs to apply heuristics to guide the search. Usually, the heuristics are derived from a cost model of query execution. In this case, the optimizer chooses to delete all of them and optimize the input query into `Q4.2` shown in Table 4.1.

When the optimizer applies semantic knowledge, it also checks whether there exists any literal that is not satisfiable. If this is the case, the optimizer can conclude that the entire query is not satisfiable and return `NULL` as the answer of the query without accessing the database at all (line 5 and 8).

Semantic Rules:

R4.1: *If the maximum draft of a ship is less than 50 then its status is active.*

ship_class(?class,_,?draft,_,_) $\wedge$ ship(_,?class,?status,_,_) $\wedge$?draft < 50
$\Rightarrow$?status = "Active"

R4.2: *If a ship class has container capability, then there must exist some ships that belong to that ship class in the database.*

ship_class(?class,_,_,_,"Y") $\Rightarrow$ ship(_,?class,_,_,_)

R4.3: *If a ship is active, then it was built after 1945.*

ship(_,?class,_,"Active",?year-built) $\Rightarrow$?year-built > 1945

R4.4: *The depth of wharves at Long Beach is at most 50 feet.*

seaport("Long Beach",?code,_,_,_,_) $\wedge$ wharf(_,?code,?depth,_,_)
$\Rightarrow$?depth $\leq$ 50

R4.5: *The length of wharves with at least one crane at Long Beach is greater than 1200 feet.*

seaport("Long Beach",?code,_,_,_,_) $\wedge$
wharf(_,?code,_,?length,?crane) $\wedge$?crane > 0
$\Rightarrow$?length $\geq$ 1200

Range Facts:

F1: 12 $\leq$ ship_class.draft $\leq$ 72.
F2: 325 $\leq$ ship_class.length $\leq$ 950.
F3: ship_class.container_cap $\in$ {"Y","N"}.
F4: ship.status $\in$ {"Active","Inactive","Resigned"}.
F5: 7 $\leq$ wharf.depth $\leq$ 100.
F6: 580 $\leq$ wharf.length $\leq$ 2700.
F7: 0 $\leq$ wharf.crane_qty $\leq$ 7.

Table 4.2: Example semantic rules and range facts

Since an implication closure contains all literals implied by other query literals given the semantic knowledge, we can extract the most restrictive ranges of variables from implication closures and use the information to optimize complex query plans. The next section presents an optimization approach for query plans based on this idea.

4.3 The Optimization Approach for Query Plans

Our approach extends the basic semantic query optimization to optimize query plans. The optimizer in our approach optimizes each subquery using the basic SQO algorithm for conjunctive queries, as the one described in the previous section, and at the same time, uses semantic knowledge to derive range constraints of the data that will be transmitted among database sites. From the derived ranges, the optimizer

Algorithm 4.2 (Query plan optimization)

```
INPUT P = query plan; KB_d = semantic knowledge learned from databases;
  LET KB = KB_d, S_b = an empty stack;
  LET S_f = a stack of subqueries in the data flow order specified in P
  WHILE S_f is not empty DO
    LET s = pop(S_f);
    optimize s by calling Algorithm 4.1 with input s and KB;
    update KB with newly inferred more restrictive ranges of variables;
    push optimized s into S_b;
  WHILE S_b is not empty DO
    LET s = pop(S_b);
    IF s is a subquery that combines or manipulates intermediate data THEN
      determine required variables V;
      extract and move newly inferred literals to L;
    ELSE (s is a subquery that retrieves data from a remote database)
      remove variables not in V from the parameter list of s;
      insert literals in L to s if they are defined on variables in s;
      IF s is changed THEN
      optimize s by calling Algorithm 4.1 with input s and KB_d;
    push s onto S_f;
  update P with S_f;  RETURN P;
```

can reduce data transmission by inserting new literals to appropriate subqueries to filter out redundant data, or by eliminating useless variables in the parameter lists of subqueries. Algorithm 4.2 gives the top level algorithm of our optimization approach.

Algorithm 4.2 takes a query plan and semantic knowledge, including semantic rules and range facts of attribute values as input. The algorithm traverses the input query plan twice by maintaining two stacks of subqueries. The first traversal optimizes each subquery and propagates inferred range information forward in the data flow order. The second traversal propagates backward the insertions and deletions of query literals made by the optimizer in the first traversal to perform global optimization on the query plan. We explain the algorithm as it takes the example query plan in Figure 4.1 and the semantic knowledge shown in Table 4.2 as input.

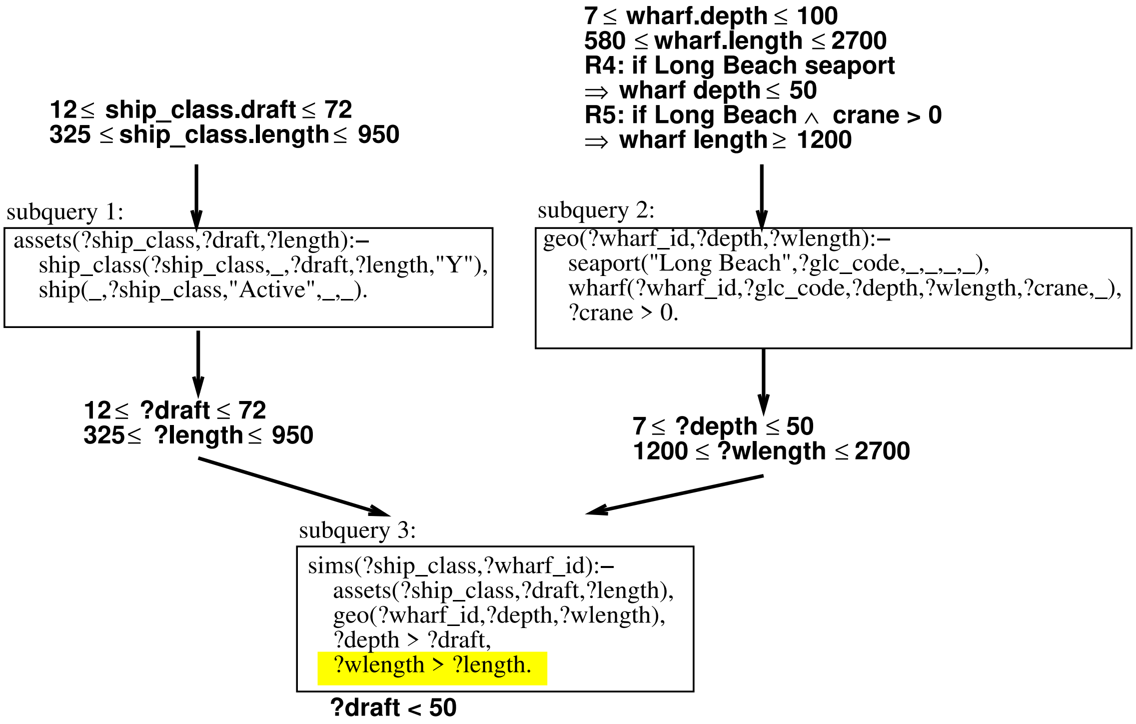

Figure 4.2: Propagating inferred range information forward to optimize subqueries

4.3.1 Forward Propagation

In the first traversal, that is, the loop from line 4 to line 8, the plan graph is traversed forward in the data flow order. During the traversal, each subquery is optimized by calling Algorithm 4.1, the basic semantic query optimization algorithm. The optimizer also infers the range of each variable in the parameter list of the subquery using the semantic knowledge. If a more restrictive range is inferred, it will be propagated forward for the optimization of the succeeding subqueries (line 7). This inference is computed at the same time when the optimizer applies Algorithm 4.1 to optimize the subquery. Algorithm 4.1 computes an *implication closure* from the semantic knowledge and the subquery (line 4 and line 6 to 9 in Algorithm 4.1). The optimizer can extract the most restrictive range of a variable that can be inferred from the semantic knowledge from the implication closure and the literals in the subquery.

This step is illustrated in Figure 4.2. Subquery 1 is optimized but no reformulation is found to be appropriate. However, based on the range facts, the ranges

Given $L = ?x_1 > ?x_2$ in a query:
1 IF $min(?x_1) > max(?x_2)$ THEN delete L.
2 IF $(max(?x_1) = max(?x_2)) \wedge (max(?x_1) > min(?x_2) > min(?x_1))$
THEN insert $?x_1 > min(?x_2)$.
3 IF $(max(?x_2) > max(?x_1) > min(?x_1)) \wedge (min(?x_1) = min(?x_2))$
THEN insert $?x_2 < max(?x_1)$.
4 IF $max(?x_2) > max(?x_1) > min(?x_2) > min(?x_1)$
THEN insert $?x_1 > min(?x_2)$ and insert $?x_2 < max(?x_1)$.
5 IF $max(?x_2) > min(?x_2) > max(?x_1) > min(?x_1)$
THEN refute L.
6 IF $max(?x_2) > max(?x_1) > min(?x_1) > min(?x_2)$
THEN insert $?x_2 < max(?x_1)$.
7 IF $max(?x_1) > max(?x_2) > min(?x_2) > min(?x_1)$
THEN insert $?x_1 > min(?x_2)$.
8 IF $(max(?x_2) > min(?x_2) > min(?x_1)) \wedge (min(?x_2) = max(?x_1))$
THEN refute L.
9 IF $(max(?x_1) = min(?x_1) = max(?x_2)) \wedge (min(?x_1) > min(?x_2))$
THEN insert $?x_2 < max(?x_1)$.
10 IF $(max(?x_1) = max(?x_2) = min(?x_2)) \wedge (max(?x_1) > min(?x_1))$
THEN refute L.
11 IF $(max(?x_1) = min(?x_1) = min(?x_2)) \wedge (max(?x_2) > min(?x_2))$
THEN refute L.
12 IF $(max(?x_2) > min(?x_2) > min(?x_1)) \wedge (max(?x_1) > min(?x_1))$
THEN insert $?x_1 > max(?x_2)$.
13 IF $(max(?x_1) > min(?x_1)) \wedge (max(?x_2) > max(?x_1) > min(?x_2))$
THEN insert $?x_2 < max(?x_1)$.
14 IF $(max(?x_2) > min(?x_2)) \wedge (max(?x_1) > max(?x_2) > min(?x_1))$
THEN insert $?x_1 > max(?x_2)$
15 IF $max(?x_1) = max(?x_2) = min(?x_1) = min(?x_2)$ THEN refute L.
16 OTHERWISE no action.

Table 4.3: Axioms for $?x_1 > ?x_2$

of variables `?draft` and `?length` are inferred and propagated to Subquery 3. Similarly, the optimizer does not reformulate Subquery 2, but it uses semantic knowledge and the constraints specified in the subquery to infer the most restrictive ranges of `?depth` and `?wlength`, and propagates the results to optimize Subquery 3.

A subquery might include literals that the basic SQO algorithm cannot optimize, such as comparisons between two variables (e.g., `?depth > ?draft`), set operators (e.g., `intersection` and `union`), and other data manipulation operators. To optimize these literals, we include a set of axioms for reasoning about these operators. Table 4.3 gives the axiom for the operator $>$. Given the ranges of the variables occurring in a special literal, the optimizer will propose one of the following four reformulations using the axioms:

- **deleting the literal** when the literal is found redundant,
- **adding new built-in literals** when more restrictive range is inferred,
- **refuting the literal** when the given ranges show that the literal is unsatisfiable,
- **no action**.

In our example, when optimizing Subquery 3, the optimizer infers that `?wlength` is always greater than `?length` because the minimal value of `?wlength` is greater than the maximal value of `?length`. Therefore, the literal `?wlength > ?length` is redundant and can be deleted (from Axiom 1 in Table 4.3). Meanwhile, for the literal `?depth > ?draft`, the optimizer inferred a new literal `?draft < 50`, because the maximal value of `?depth` is 50. This inferred literal is inserted into the subquery (from Axiom 13).

4.3.2 Backward Propagation

After the optimizer finishes optimizing a subquery and inferring ranges of variables, it pushes the subquery into another stack S_b for the second query plan traversal (line 9 to line 19). The optimizer pops out subqueries from S_b during the traversal, which corresponds to a backward traversal in the data flow order of the query plan. Each subquery is processed differently depending on whether a subquery retrieves data or process intermediate data. This step is illustrated in Figure 4.3.

There are two cases. The first case (line 11 to 13) is when the subquery is to combine or process intermediate data (e.g., Subquery 3). The optimizer examines the optimized subquery and determines the required variables (line 12). Since some literals were deleted by the optimizer during the forward traversal, there is no need to retrieve or compute the values of the variables occurring in those literals. The optimizer can propagate this information to optimize the preceding subquery that retrieve data values of those variables. In our example, the optimizer takes optimized Subquery 3 and determines that in this subquery, all the variables are required except `?wlength` and `?length`, because the literal `?wlength > ?length` has been deleted

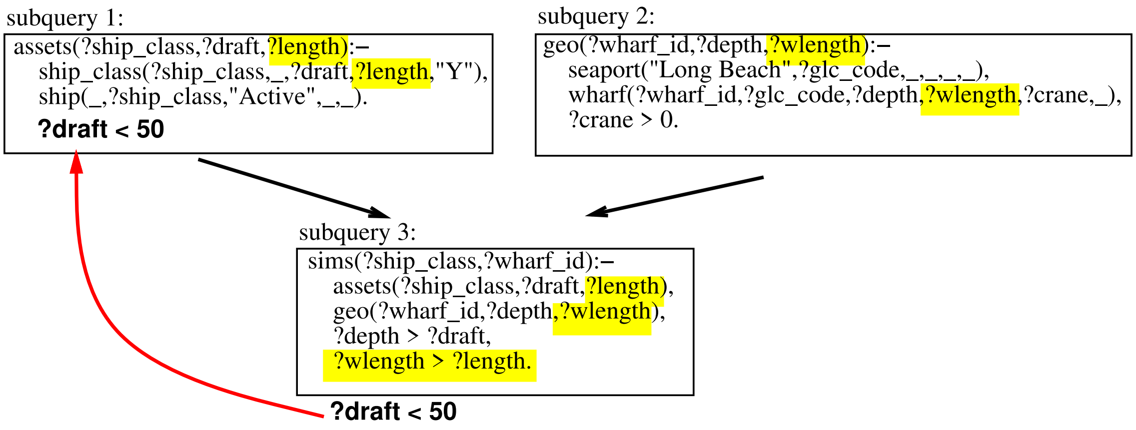

Figure 4.3: Propagating newly derived literals backward to optimize query plan

from Subquery 3 and neither variable is used for output or in any other constraints. The optimizer saves the required variables in V and removes the references to the two redundant variables from the database literals in Subquery 3.

Next, the optimizer extracts newly inserted built-in literals from the optimized subquery if the values of their variables are retrieved in one of preceding subqueries. If a built-in literal satisfies this condition, then the optimizer will move the literal from the subquery to a set L temporarily so that later in the traversal, the optimizer can insert it into an appropriate preceding subquery. This allows the system to evaluate the literal as early as possible to reduce intermediate data. In Subquery 3 there is a newly inserted built-in literal `?draft < 50`, which involves a variable initially defined in Subquery 1. The optimizer thus moves this literal to L for further processing.

The second case (line 14 to 18) is when the subquery is to retrieve data from a remote database. The optimizer first removes any variable in the parameter list not in the set V of the required variables, and then inserts literals collected in L previously into the subquery if the literals involve variables generated in this subquery. In our example, when the optimizer encounters Subquery 2 — a subquery that retrieves data from a remote database, the optimizer finds that the variable `?wlength` is not a required variable and removes it from the parameter list, as well as the reference to this variable in the database literal on `wharf`. The optimizer continues its traversal and encounters Subquery 1. Similarly, it finds that `?length` in this subquery is not required and removes the variable from the subquery. The

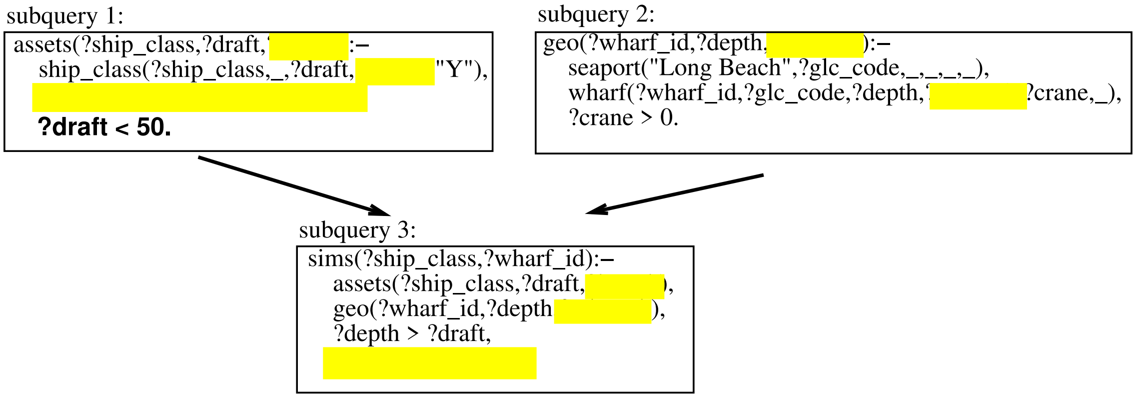

Figure 4.4: Optimized query plan

optimizer also finds that in the set L, there is a literal `?draft < 50` involving the variable `?draft` that is defined initially in Subquery 1. Therefore, the optimizer move the literal to this subquery and completes the traversal.

If a subquery is modified in the previous steps (line 15 and 16), the optimizer invokes the basic semantic query optimization algorithm to optimize this subquery again to see if there are additional optimization opportunities. Since both Subquery 1 and 2 are modified, the optimizer will invoke Algorithm 4.1 to optimize both of them. At this moment, Subquery 1 is identical to the example query `Q4` in Table 4.1. The optimization for Subquery 1 is also the same as that for `Q4` as we have discussed in the previous section. The resulting plan is shown in Figure 4.4, where terms in bold font are inserted new terms, and terms under shaded regions are deleted.

4.3.3 Analysis of the Query Plan Optimization

The rationale of the forward-propagation is to use literals specified in subqueries to specialize the semantic knowledge as much as possible for more effective optimization on succeeding subqueries. The backward-propagation step attempts to detect unnecessary data retrieval and move newly derived literals to subqueries that retrieves remote databases so that the literals can be evaluated as early as possible in a query plan and reduce intermediate data.

Since the loop for each traversal repeats the time proportional to the number of subqueries, and in each repetition, the time required for the basic SQO is the dominant factor, the time complexity of Algorithm 4.2 is $O(2 \cdot n \cdot T_{sqo})$, where n

is the number of subqueries in the input query plan, and T_{sqo} is the maximal time required to perform the basic semantic query optimization for each subquery.

The correctness of Algorithm 4.2 can be verified as follows. In each step, the inferred ranges of variables propagated forward are always larger than the ranges of variables in the answers; and therefore, no data will be lost due to the modifications to subqueries. Since the inferred ranges of variables are always at least as restrictive as the literals specified in the query, no undesired data will be retrieved due to the modifications to subqueries. We can also establish that moving literals backward will not change the semantics of a query access plan. The proof is similar to the proof for the correctness of the predicate push-down techniques [Ullman, 1988, Levy *et al.*, 1994]. Consequently, the resulting query plan of Algorithm 4.2 will return the same answer as an input query plan, as long as the given database state is consistent with the semantic knowledge.

The resulting plan may also be much less expensive than an input plan. In our example, since the system does not need to transmit `?wlength` and `?length`, and a more restrictive constraint `?draft < 50` is inserted to Subquery 1, the intermediate data transmitted from remote databases to the local system is significantly reduced. Furthermore, Subquery 1 can be executed more efficiently because the system does not need to retrieve and compute a join over the large relation `ship`. In our experiment, it takes 3.17 seconds to execute the original plan while it takes only 1.78 seconds to execute the optimized plan, including 0.03 second of optimization time. This amounts to a 43.8 percent reduction. Therefore, our optimization approach can effectively optimize this query and reduce the execution cost.

4.4 Related Work in Query Optimization

The section compares this approach with related work in semantic query optimization for conjunctive queries, predicate move-around and semi-joins, a conventional syntactic query optimization technique for distributed databases.

The query plan optimization approach presented here is elaborated from our prototype approach described previously in [Hsu and Knoblock, 1993b]. Our approach extends previous work in SQO (e.g., [Yu and Sun, 1989, Shenoy and Ozsoyoglu, 1989, Sun and Yu, 1994]) to optimize query plans generated by an information mediator.

The main extensions include the ability to make inference about data manipulation operators (e.g., comparisons, disjunctions, set operators, etc.) and move literals inferred from semantic knowledge between subqueries in a query plan to provide global optimization. As a result, our approach can optimize complex queries that includes complex data operations, as long as a complex query is decomposed into a query plan of conjunctive subqueries.

Optimizing a query plan by moving literals is previously studied in the work on *predicate push-down* [Ullman, 1988], and *predicate move-around* [Levy *et al.*, 1994]. Predicate push-down is a commonly used query optimization technique. By pushing data selection predicates down the hierarchical access graph of a query, predicate push-down allows the selections to be applied as early as possible during query execution. Predicate move-around is a generalization of predicate push-down. This technique optimizes queries that involve views by moving predicates across subqueries in a query graph. Similar to the forward and backward propagations of query literals, predicate move-around moves predicates up in a query graph as an intermediate step before pushing them down.

Our optimization approach for query plans differs from those techniques in the use of semantic knowledge. Since semantic knowledge may enlarge the search space of optimization, the potential savings of their knowledge-free techniques may not be as much as what our algorithm can achieve. Though predicate move-around does not apply semantic rules in optimization, it can apply functional dependencies of attribute values to infer the ranges of variables. For example, if there exists a functional dependency $A \rightarrow B$, and both A and B are used in a query, their optimizer can replace B with $f(A)$ and infer new predicates. Our approach does not apply functional dependencies in this manner. However, since Horn-clause rules can express functional dependencies, it should be straightforward to extend our approach to apply functional dependencies.

The most significant difference between predicate move-around and our approach is that they assume that literals are not expensive. Their system may insert a new literal to a subquery even if it is very expensive. Our approach, in contrast, is able to apply the basic semantic query optimization algorithm to replace an expensive new literal with less expensive ones if necessary.

Compared to conventional syntactical optimization techniques for distributed database systems, such as *semi-joins* [Apers *et al.*, 1983, Jarke and Koch, 1984, Ullman, 1988], our approach is more appropriate in a heterogeneous environment. The result of a semi-join of relation R by relation S is the R tuples extracted from the result of the join of R and S. The semi-join technique optimizes a cross-database join of R and S by joining S and the semi-join of R by S computed in the database server where R resides. This requires merging data from one remote database to the other. However, in many applications of heterogeneous information systems, database servers might have write-protection against external data and prohibit an optimizer from computing semi-joins. Even if there is no write-protection, in a heterogeneous environment, two databases could be quite different and data may need to be translated before they can be merged. This implies that we need to build translators between each pair of databases integrated, which amounts to $O(n^2)$ translators if there are n databases. Our approach optimizes query plans for an information mediator that requires only n wrappers of translators for n databases. The overhead at run time for our approach is smaller because the optimization does not require access to remote databases. By the same token, our approach applies to the applications that involve databases with write-protection.

More importantly, since our optimization approach does not depend on how an individual database server executes a subquery, it can be incorporated easily on top of existing query optimizers. When a new database is integrated to a multidatabase system, the optimizer can be used without changing its code. This unique feature is crucial for the extensibility of an information mediator.

4.5 Learning for Query Plan Optimization

Though a query plan retrieves data from multidatabases, semantic rules that express regularities in a single database can still be used by the query plan optimization approach to perform global optimization. Therefore, semantic rules required for optimizing multidatabase queries can be learned by the learning approach developed in Chapter 3 from each individual database integrated in the multidatabase system. Information mediators that access the same databases can share the semantic rules learned from that database, regardless of what other databases are integrated.

To trigger the learning for optimizing a multidatabase query, the learner can use a query planner to generate subqueries to single databases, and trigger the learning using the subqueries.

Since our optimization approach for query plans is more effective when the optimizer is able to infer more accurate ranges of variables, we can "bias" the learning of the alternative query to prefer a built-in literal defined on an output variable. That way, the learner will generate range rules about output variables that allow the optimizer to infer accurate ranges.

4.6 Discussion

This chapter presented a novel query optimization approach to reducing the cost of query plans generated by an information mediator. Our approach optimizes a query plan by modifying subqueries in the query plan using semantic knowledge about data. To efficiently execute a complex multidatabase query, it is crucial to reduce redundant intermediate data. The approach presented here can use semantic knowledge to infer the ranges of intermediate data accurately and yield arbitrarily large additional savings for complex multidatabase queries.

In addition to its effectiveness, the approach is more general and flexible than previous work in semantic query optimization in many aspects. This approach optimizes a larger class of queries, exploits more expressive semantic knowledge, and detects more optimization opportunities than previous work. This global optimization approach can be implemented on top of existing query optimizers in a heterogeneous environment and hence supports the extensibility of multidatabase systems.

Chapter 5

Empirical Evaluation

This chapter describes the empirical evaluation on the proposed learning and query optimization approaches. For this purpose, these approaches were implemented in a learning system and a query optimization system and incorporated with an information mediator that integrates heterogeneous databases.

The evaluation consists of four experiments. Section 5.2 describes the first experiment which is to evaluate the effectiveness of the rules generated by the learning approach when they are applied to optimize queries in a given database state. The experiment compares the optimization performance produced by the rules learned by BASIL with hand-coded rules.

The second experiment, described in Section 5.3, is to evaluate our approach to dealing with database changes. Since it is too expensive to replay and control a continuous sequence of the mix of queries and data modification transactions that is sufficiently long to simulate real-world database usage, we cannot fully demonstrate the net savings yielded by applying our learning and optimization approach. However, by showing the accuracy of the robustness estimation, it suffices to affirm that a learner can minimize its effort in rule learning and maintenance while provide high utility rules for the optimizer.

The third experiment examines the interaction between effectiveness and robustness of a semantic rules. The experiment compares a variety of properties of the learned rules — optimization performance, robustness and applicability. Section 5.4 describes and reports the results of this experiment.

The fourth experiment compares the utility of relational rules and range rules. This experiment aims to verify that in general, relational rules are more widely applicable and produce higher savings, but less robust against database changes. Section 5.5 reports the results of the comparisons.

To demonstrate the feasibility of the learning and optimization approaches, Section 5.6 reports the execution time statistics of the learning and optimization systems. Finally, Section 5.7 summarizes the conclusions drawn from the results of these experiments.

5.1 Environment for the Experiments

The rule induction system BASIL[1] is an implementation of the learning approach to the acquisition of high utility semantic rules for SQO. BASIL learns semantic rules for PESTO,[2] an implementation of the query plan optimization approach. PESTO uses semantic rules learned by BASIL to optimize query plans for an information mediator. These systems are developed to empirically evaluate the approaches developed in this research. They are incorporated with the SIMS information mediator [Arens *et al.*, 1993, Knoblock *et al.*, 1994, Arens *et al.*, 1996]. SIMS applies a variety of AI techniques to build an integrated intelligent mediator between users and distributed, heterogeneous multidatabases so that users can access those databases without knowing the implementation details such as their locations, query languages, platforms, etc. SIMS invokes PESTO to optimize query plans, and PESTO in turn invokes BASIL to learn the required semantic rules. Figure 5.1 shows the organization of SIMS with the query plan optimizer and the learner.

SIMS takes as input a query expressed in the LOOM knowledge representation language [MacGregor, 1990], which is also used as the representation language to build an integrated model of databases. To optimize queries for SIMS, PESTO has a component to translate a LOOM subquery into an internal representation similar to Datalog to facilitate optimization, and a component to translate the result back to LOOM. The semantic rules are expressed in the same internal representation. By attaching different translation component, PESTO can optimize queries in other query languages. BASIL uses the same internal representation to express the semantic

[1]BAyesian Speedup Inductive Learning.
[2]Plan Enhancement by SemanTic Optimization.

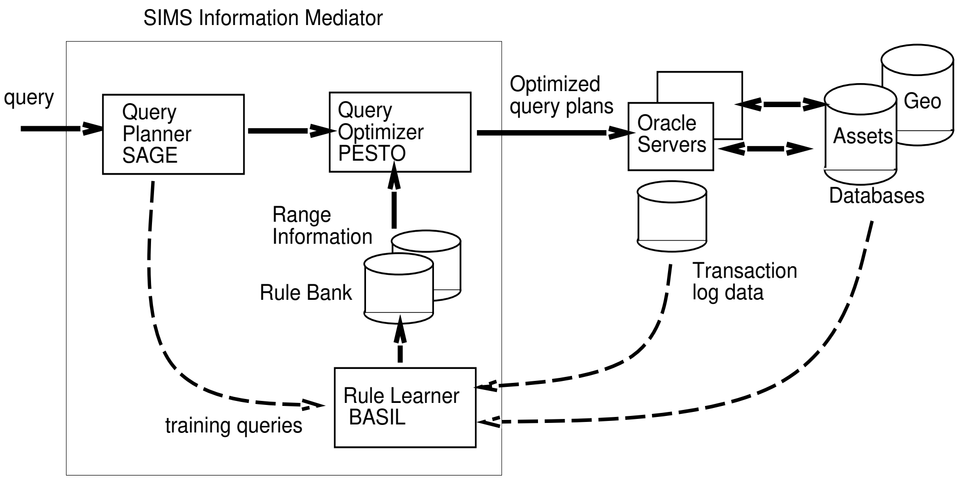

Figure 5.1: Complete organization of SIMS with BASIL and PESTO

rules it learns. When it needs to read data or information about database (e.g., schema), BASIL sends a query to SIMS to obtain the required information.

To monitor the generation, optimization, and execution of query plans, SIMS provides a graphical interface for users to compose and send queries to SIMS. After optimizing a query plan, PESTO can *explain* how query plans are optimized by tracing rule application sequences that reformulate a query literal. Figure 5.2 shows an example snapshot of the interface screen. The SIMS interface is divided into three main panes: the interface/trace pane (lower right quadrant), the query pane (lower left quadrant), and the graph pane (upper half). In this example, the graph pane displays a graph of the query plan generated by SIMS. The user can select to optimize the query plan through the interface by invoking PESTO. The subqueries that are reformulated by the optimizer will be highlighted and the user can click on a highlighted subquery to see an explanation on how it was reformulated. To display the explanation, the interface will open a new window to show the original subquery and the optimized subquery. Meanwhile, the interface will highlight query literals that are different in the two subqueries and display the explanation of the reformulation if the user clicks on a highlighted query literal, as shown in Figure 5.2. This explanation allows the user to examine the rules used by PESTO in query optimization.

For the purpose of our experiments, SIMS is connected with two remote ORACLE relational databases via the Internet. These databases originally are part of a

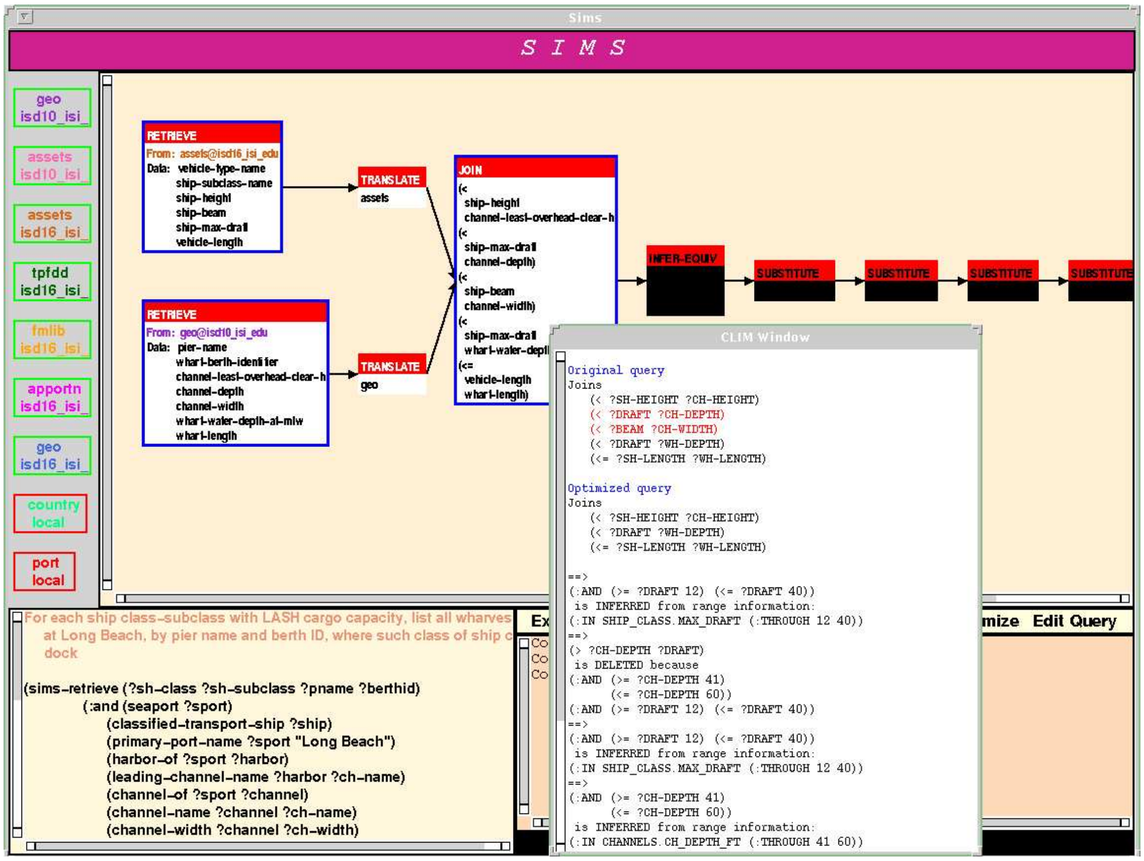

Figure 5.2: SIMS interface provides explanations of optimization operations

real-world transportation logistic planning application. Table 5.1 summarizes the contents and the sizes of these databases. Together with the databases, there are 29 sample queries written by the users of the databases. We also have 3 queries written for the purpose to test different functionalities of the SIMS query planner, and 4 queries to test PESTO, especially to test its ability to detect null queries (i.e., queries that return an empty set). That is a total of 36 queries. Among these 36 queries, 18 are multidatabase queries that require access to multiple databases to retrieve the answer. Table 5.2 lists some properties of the multidatabase queries. To train the learner, BASIL, 23 queries are selected to server as the training queries The selection is based on the similarity of queries. Because we found that BASIL learns nearly identical sets of rules using similar queries, to save experimentation time, we remove some similar queries from the training set. In addition to the learned rules, PESTO uses 271 range facts compiled from the databases for the optimization. SIMS,

Databases	Contents	Relations	Tuples	Size(MB)	Server
Geo	Geographical locations	15	56124	10.48	HP9000s
Assets	Air and sea assets	16	4881	0.51	Sun SPARC 4

Table 5.1: Sample databases in a transportation logistic planning domain

PESTO and BASIL were running on a Sun SPARC-20 workstation during the experiments. We synthesized 123 sample transactions that represent possible transactions of the experimental databases based on the semantics of the application domain to evaluate the accuracy of the robustness estimation. The set of transactions contains 27 updates, 29 deletions and 67 insertions, a proportion that matches the likelihood of different types of transactions in this domain.

5.2 Effectiveness of Learned Rules

The first experiment is to evaluate whether the learning approach can generate effective rules for cost reduction in a given database state. This experiment applies a *k-fold cross validation* [Cohen, 1995a] to test the effectiveness of the learned rules. The 23 training queries are randomly divided into four sets, three of them contain 6 queries, and one contains 5 queries. For each set of queries, BASIL takes the remaining three sets of queries as training queries to learn a set of semantic rules. The selected set of queries is combined with the 13 additional queries to form the test set of queries. Next, SIMS takes the test set as input and invokes PESTO to optimize the queries using the learned semantic rules. After collecting performance data, the learned rules are discarded and the process repeats for the next set of queries. The experiment will thus generate four sets of performance data.

The beam size for the rule pruning search in BASIL is set to three. To avoid learning identical rules, each query is optimized using existing learned rules before the learner generates training queries. We also remove logically redundant rules after the learning to reduce the number of learned rules.

Prior to this experiment, we have hand-crafted a set of 112 semantic rules to demonstrate the effectiveness of the query plan optimizer PESTO, These rules were carefully designed after several iterations of debugging and modifications to allow

ID	Short description	Number of Subquery	Query Length	Size of Answer
205	T-countries' airports, by name, where C-5 aircraft can land at wartime	3	22	7
206	T-countries' airports, by name, where fully loaded DC-8-61s can takeoff at wartime	3	22	1
213	wharves with container cranes at Long Beach, by pier name and berth ID, where slow Container/Breakbulk ships can dock	3	51	108
214	wharves at Long Beach with RORO ramps, by pier name and berth ID where METEOR ships can dock	3	43	54
215	For each ship class-subclass with LASH cargo capacity, list all wharves at Long Beach, by pier name and berth ID, where such class of ship can dock	3	51	864
216	ships which can handle RORO cargo and can dock in T-country	3	30	3
217	ship classes, by ship class name, seaport name, and berth-type name which can handle container and can dock at S-port or T-port	3	25	272
218	ship classes, by ship class name, seaport name, and berth-type name which can handle container and can dock at S-port or T-port	3	27	360
226	low-altitude airports where a C5 can land"	3	21	4
227	airports in T-country where a C5 can land"	3	19	9

Table 5.2: Multidatabase queries used in the experiments

the optimizer to explore as much optimization opportunity as possible for the sample queries. We report the optimization performance produced by the hand-coded rules for the purpose of comparison.

The performance data contains the total elapsed time of each query execution, which includes the time for database accesses, network latency, as well as the overhead for semantic query optimization. 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 their medians. Then for each query, the percentage time savings are obtained by computing the ratio of the total time saved due to the optimization over the total execution time without optimization.

Table 5.3 shows the average of the savings for all queries, the average of savings for multidatabase queries and the standard deviations. The data shows that the learned rules can produce a significant savings on the test queries, with a ten percent higher savings for multidatabase queries. The data also shows that the learned rules

	Test 1	2	3	4	Average savings	hand-coded rules
All	28.99%	31.60%	33.94%	29.86%	31.07% s=2.20%	25.84%
Multidb	39.43%	42.51%	42.61%	39.63%	41.05% s=1.75%	36.19%
# of Rules	101	119	106	118	111 s=91	112
opt time (s)	0.038	0.047	0.041	0.054	0.045 s=0.007	0.044

Table 5.3: Performance data of learned rules and hand-coded rules

outperform hand-coded rules in all four tests. We note that some of our test queries are already very cost-effective, and there is not much room for optimization for those queries. But for some expensive multidatabase queries, the savings can reach as high as 70 to 90 percent. This result is significant given that the standard deviations are low, and because all tests use about the same number of rules and optimization time.

To examine the cumulative effects of training, Figure 5.3 presents the result of our investigation on the relation of the number of training queries and the coverage of rules, that is, the number of queries for which the learned rules are found to be applicable. The result is obtained from an incremental k-fold cross-validation procedure. As in the k-fold cross-validation, we randomly divide the 23 test queries into four sets of five or six queries and combine one set and the additional queries as the test set, and the other three sets as the training set. Recall that in Chapter 4, we have discussed that given a training query, if it is a multidatabase query, the learner will invoke the query planner to decompose the query into subqueries to single databases before it can be used to trigger the learning. Also, if there is more than one primary relation in a training query, the learner will use a preprocessor to partition the parameter list and generate a set of training queries with one primary relation. As a result, for each query used for training, there are about four to five queries generated to train the learner. We call the generated queries *trigger queries* to distinguish them from the given queries that are used for the purpose for training.

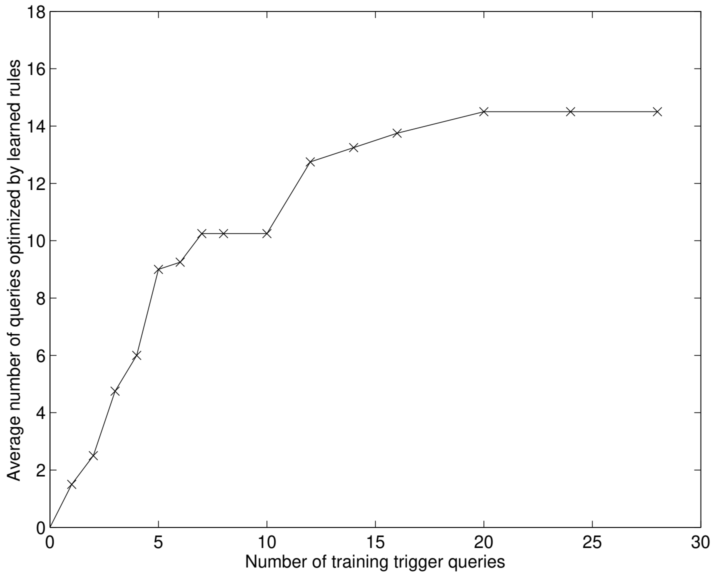

Figure 5.3: The effect of training and rule coverage in BASIL

Therefore, for each set of training queries, there are about 28 trigger queries actually used to train the learner.

The incremental k-fold cross-validation proceeds as follows. For each set of 28 trigger queries, the procedure starts by shuffling the set of trigger queries. The first trigger query is selected and included in the training set to train BASIL. Then the procedure calls PESTO to optimize test queries and counts the number of queries covered by the learned rules. The procedure includes the next trigger query in the training set and repeats the train-and-count process. This repeats for eight times. After that, the procedure increases the size of the training set by two trigger queries for four times, and then by four for three times. This yields a sequence 15 data of the rule coverage after BASIL is trained on $1, 2, \ldots, 8, 10, \ldots, 16, 20, 24, 28$ trigger queries. After all four sets of trigger queries are used, the procedure will yield four sets of 15 data for each size of training trigger queries. Figure 5.3 is a plot of the average rule coverage data over the four sets of trigger queries against the size of training trigger queries.

Figure 5.3 shows that the learning curve of BASIL has a characteristic shape, which grows quickly and flattens out. The curve shows two plateaus, the first and short one is between 6 to 10 trigger queries, and the next one is after there are 20 trigger queries. This is probably because our sample queries can be roughly divided into two categories: queries about seaports and ships, and queries about airports and airplanes. Most of queries in one category are covered after BASIL is trained with 6 queries and then it takes on average another 4 trigger queries to learn more rules that cover queries in the other category. Remarkably, it takes just 20 trigger queries, which could be generated from about four or five original training queries, to cover most of the 18 test queries. This result supports the claim that the learning approach can generate general rules that can be applied to a wide range of queries for query optimization with a small number of training queries.

5.3 Accuracy of Robustness Estimation

This experiment evaluates the accuracy of robustness estimation so as to establish the claim that using the robustness estimation allows a learner to minimize the cost of dealing with database changes. The experiment design can be outlined as follows: train BASIL to learn a set of rules and estimate their robustness, use the 123 synthesized data modification transactions to generate a new database state, then check if high robust rules have a better chance to remain consistent with the data in the new database state.

Table 5.4 converts numeric estimation of robustness into four discrete levels based on the number of transactions. Suppose the robustness of a rule is estimated as ρ. From the definition of robustness, it is equivalent to say that the probability that an invalidating transaction of this rule will *not* be performed is estimated as ρ. Assuming that each transaction is probabilistically independent, then given 123 transactions, the probability that *none* of these transactions will invalidate the rule can be estimated as ρ^{123}. That is, the probability that the rule will remain consistent after 123 transactions based on the estimated robustness is ρ^{123}. We denote this probability as $P_c(r,n) \equiv (\text{robust}(r))^n$, the probability of consistency for a rule r after n transactions, or simply P_c. Clearly, we have $0 \leq P_c \leq 1$. We divide P_c values into four quarters to determine four levels of robustness thresholds. Table 5.4 shows the P_c values and their corresponding estimated robustness values.

In this experiment, we let BASIL exhaust the search space during the rule pruning by setting the beam size to positive infinity so that it will not remove low robust partially pruned rules. We use all 23 training queries to train BASIL, which results

	very high	high	low	very low
P_c	0.75	0.50	0.25	0.00
Robustness	0.99766	0.99438	0.98879	0.00000

Table 5.4: Probability of consistency and corresponding robustness after 123 transactions

	Consistent	Inconsistent	Total
very high	40	7	47
high	49	13	62
low	19	22	41
very low	151	54	205
Total	259	96	355

Table 5.5: The joint distribution of the actual and estimated robustness

in 355 rules. Meanwhile, BASIL estimates the robustness of these rules during the pruning. We use another set of 202 sample transactions to assist the robustness estimation. Most of those transactions are synthesized for our earlier experiment in Chapter 2. After generating the rules and collecting their robustness, we apply the set of 123 transactions to the two relational databases connected to SIMS and generate a new database state. Next, we check the consistency of all 355 rules and identify 96 inconsistent rules in the new database state. Table 5.5 shows the number of rules in each levels of robustness against the number of actual consistency of rules. We perform a statistic significance test on the result in the table. Since we obtain $\chi^2 = 19.4356$ from this table, and under the null hypothesis that the consistency of a rule and its estimated robustness are independent, the probability to get a χ^2 value this high is less than 0.01, we conclude with a 99 percent confidence that the robustness estimation accurately reflects the likelihood of whether a rule may become inconsistent after data modification transactions.

In order to evaluate the predictive power of the robustness estimation, we define two measures

$$\text{recall} = \frac{|I \cap L|}{|I|}$$

	Consistent ($\neg I$)	Inconsistent (I)	Total	
$P_c > 0.75$ ($\neg L$)	40	7	47	
$P_c \leq 0.75$ (L)	219	89	308	precision = 28.89%
Total	259	96	355	
	recall =	92.70%		

Table 5.6: The joint distribution of the actual and estimated robustness

$$\text{precision} = \frac{|I \cap L|}{|L|}$$

where I is the set of inconsistent rules and L is the set of rules that are estimated as likely to become inconsistent. The definitions are analogous to their definitions in natural language processing and information retrieval research. Intuitively, *recall* indicates the proportion of inconsistent rules being identified as likely to become inconsistent rules, and *precision* indicates the proportion of the estimatedly low robust rules that actually become inconsistent.

Consider that a threshold for low robust rules is set to be $P_c \leq \frac{3}{4}$. That is, if the probability of consistency for a rule is less than 0.75, then it is predicted to become inconsistent after 123 transactions. From Table 5.6, this threshold produces a recall of 92.7 (= 89 / 96) percent and a precision of 28.89 (= 89 / 308) percent. That is, with this threshold, BASIL can accurately point out 92.7 percent of inconsistent rules. But on the other hand, among all those rules that are classified as likely to become inconsistent, only 28.89 percent actually become inconsistent. This is not surprising because the robustness estimation may overestimate the probability of invalidating transactions of a rule in situations where enumerating all possible invalidating transactions is too expensive. In fact, by raising the threshold, we can obtain a higher recall while maintain the precision to be around 28 percent. For example, if we set $P_c \leq 0.95$ as the threshold, then we can obtain a high recall of 98.95 percent, and a precision of 28.27 percent. Consequently, since the robustness estimation can accurately point out low robust rules, by properly adjusting the

	Average savings		Average # of rules		Avg opt. time (s)	
very high ($P_c > 0.75$)	12.54%	s=1.89%	28.5	s=6.45	0.0287	s=0.008
high ($P_c > 0.50$)	13.11%	s=2.71%	50.25	s=10.24	0.0393	s=0.010
low ($P_c > 0.25$)	13.54%	s=0.93%	63.0	s=11.1	0.0362	s=0.004
very low ($P_c > 0.00$)	31.07%	s=2.20%	111.0	s=8.91	0.038	s=0.038

Table 5.7: Performance data of rules with different robustness thresholds

threshold, the estimated robustness values can provide the sufficient information for rule learning and maintenance to deal with database changes.

5.4 Effectiveness versus Robustness

Intuitively, a low robust rule that expresses specific data regularity in a given database state might produce a high cost reduction, while a high robust rule such as an integrity constraint on the gender of pregnant patients in a hospital information system might not be effective for query optimization. However, there is no empirical data that verifies this intuition. This section describes an empirical study of the interaction between effectiveness and robustness of semantic rules. For the purpose of this study, BASIL uses the four robustness levels as shown in Table 5.4 in the rule pruning to filter learned rules into four different levels of robustness. We designed three experiments to compare the utility of the learned rules. The first experiment compares their average savings. The second experiment compares the converging rate of the coverage of the learned rules. The third experiment verifies our assumption on the interaction between the robustness, length and applicability of the learned rules.

We note that it is possible that an individual rule yields high cost reduction together with a set of rules but low with another set. Since how much an individual rule can contribute to the cost reduction can only be determined in the context of the entire rule set used for the optimization, we do not have any experiment to compare the effectiveness and robustness on an individual rule basis.

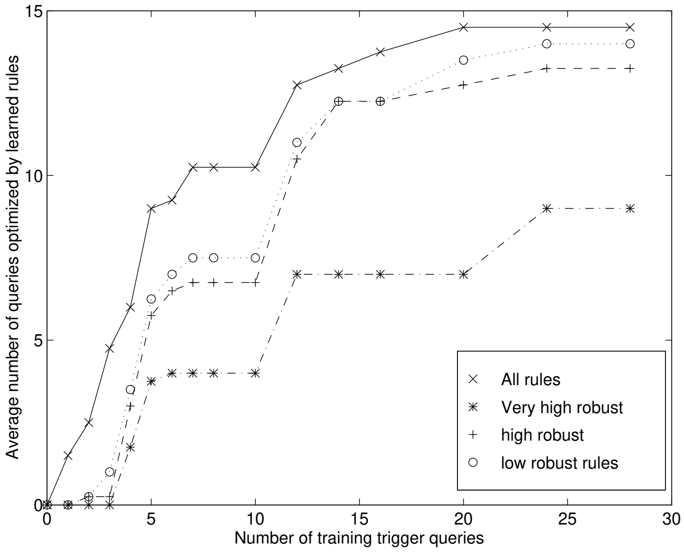

Figure 5.4: Rule coverage rates for rules of different robustness levels

In the first experiment, for each level of robustness threshold, BASIL is modified so that it will discard a partially pruned rule if its estimated robustness is below the threshold during the rule pruning search. Then we apply a k-fold cross-validation as described in Section 5.2 to obtain the average savings produced by the learned rules. Table 5.7 shows the average performance data with the standard deviations. The data show that the rules learned with the robustness thresholds may not produce savings as high as those with no robustness threshold. However, since there are more than twice as many rules in the "very low" (i.e., no threshold) case as in other cases, the low savings might be ascribed to the lack of sufficient number of rules. It is remarkable that using the set of 28.5 very high robust rules can still produce a 12.54 percent savings.

We apply the incremental k-fold cross-validation as described in Section 5.2 to obtain the data on the converging rate for different robustness thresholds. Figure 5.4 shows the plot of the data. Interestingly, the four curves for different thresholds have almost the same shape, that is, they converge at about the same rate. But with higher thresholds, the number of optimized queries is smaller than the case where no robustness threshold is applied.

In Section 3.4, we use length to measure the applicability of a rule and we assume that robustness may also interact with applicability. The next experiment attempts

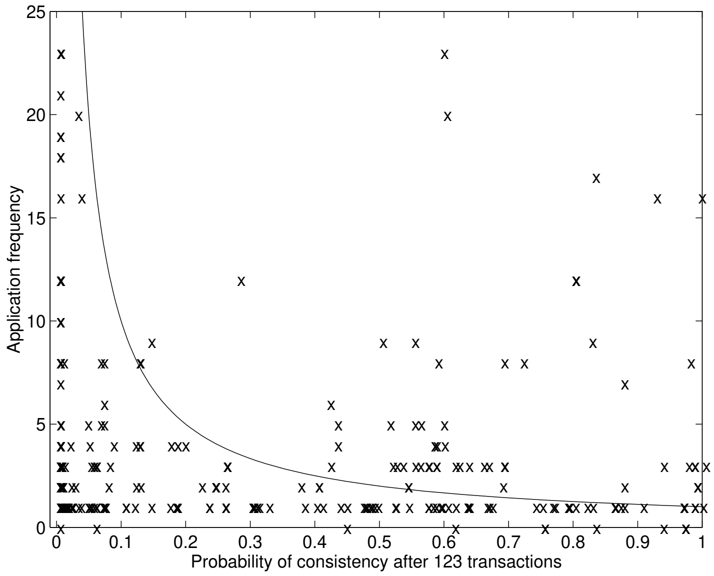

Figure 5.5: Relation between application frequency and estimated robustness

to verify this hypothesis. In this experiment, PESTO uses the 355 rules learned for the experiment in Section 5.3 to optimize all of our 36 queries, and count the application frequency for each rule, that is, how many times a rule is located as an applicable rule during the query optimization. Based on the data, we produce a scatterplot to visualize the relation between the P_c values and the applicability of a rule, as shown in Figure 5.5. As we expect, for most of rules, the probability of consistency is inversely proportional to their application frequency, because a high density of the rule population is distributed below the curve $y = 1/x$. We note that there is a significant population of rules positioned on the right-upper corner — they are both effective and robust. Figure 5.6 shows the scatterplot of the length of rules against their application frequency. The plot suggests that with few exceptions, widely applicable rules are short, but short rules are not necessarily widely applicable.

5.5 Utility of Relational Rules

One of the important features of BASIL and PESTO is their capability to learn and use relational rules for semantic query optimization. The consequent of a relational rule is a database literal, while the consequent of a rule in the other class (i.e., the range rules) is a built-in literal. Relational rules allow the optimizer to identify redundant

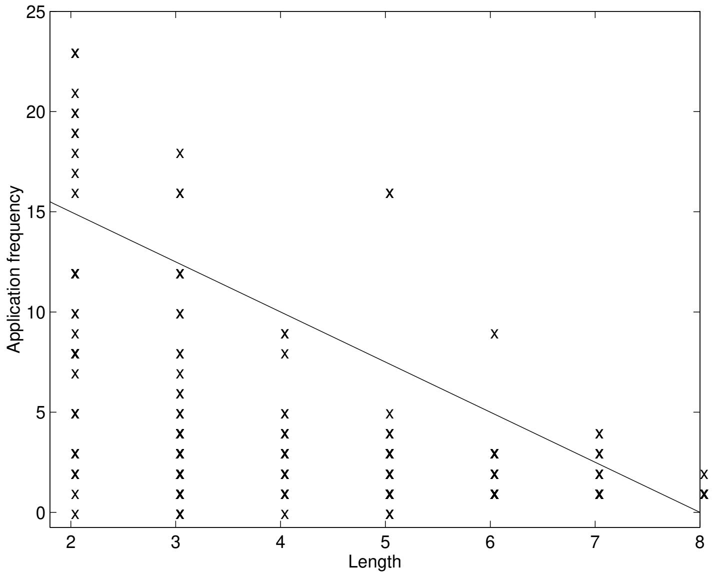

Figure 5.6: Relation between application frequency and length of rules

	Average savings	# of rules	Avg. app. frequency	Avg est. robustness	# invalid
Range rules	13.22% s=1.14%	163	2.219 s=2.57	0.98466 s=0.019	32
Relational rules	26.28% s=9.94%	192	5.135 s=5.26	0.82115 s=0.2329	66

Table 5.8: Comparing range rules and relational rules

relational joins and eliminate the cost to access a redundant relation. Relational rules also allow the optimizer to infer a cost-reducing relation join. Relation joins are usually the dominant factor in query execution cost. As a result, relational rules may potentially produce higher savings. However, relational rules are less robust compared with range rules. One explanation is that relational rules are more sensitive to database changes than range rules because they express the subset relationship between attribute values of two different relations. In contrast, range rules are less sensitive to database changes because they express the value range of a single attribute under certain conditions. This section describes an empirical comparison between relational rules and range rules to verify our assumptions.

	Length of query	Number of rules learned	Elapsed real time for learning (s)	Database accesses
Average	8.39	6.02	57.94	50.46
s=	3.92	11.17	58.08	36.14
Max	18	61	378.15	157
Median	7	2	44.73	42
Min	3	0	5.7	9

Table 5.9: Learning time statistics of BASIL on training trigger queries

Table 5.8 shows the average performance data and the standard deviations. The average savings data are obtained using a k-fold cross-validation as described in Section 5.2, except that before PESTO optimizes the test queries, a filter is used to remove range rules or relational rules from the rule bank. The data shows that using only relational rules yields about twice as much savings as using only range rules. The other data in the table are collected from the 355 rules learned for the experiment in Section 5.3. The average robustness and the actual number of inconsistent rules after 123 transactions show that relational rules are on average less robust than range rules, while the average application frequency data, together with the average savings, provide strong evidence that relational rules are much more effective. This may suggest using different robustness thresholds for relational rules and range rules during the rule pruning so that the learner can generate more effective relational rules and filter out more low robust range rules.

5.6 Efficiency of Learning and Optimization

When conducting the experiment in Section 5.3, we collect the execution time statistics of BASIL, which takes 59 trigger queries generated from the set of all 23 training queries as input and generate 355 rules. Table 5.9 shows the average learning time statistics. Since the standard deviations are quite large, the table also provides the maximum, median, and minimum values. The median learning time is less than a minute but in some case learning could take up to 7 minutes.

Previous sections have reported the average optimization time data whenever they present the data on average savings. The data shows that in all our experiments, with about 100 rules, PESTO takes about 0.03 seconds to optimize a query plan. This optimization time is very small compared to the average query execution time, which is about 1.2 seconds without optimization.

5.7 Discussion

This section describes our experiments to evaluate the learning and optimization approaches. The results show the utility of the learned rules for query optimization. The experiments also verify some of our assumptions in the previous chapters. In summary, we review the results and the conclusions we can draw from these experiments.

- The performance data on average savings shows that BASIL learns sufficient rules for PESTO to produce substantial savings and outperform the hand-coded rules. PESTO provides better savings for multidatabase queries. The learning curve shows that the learned rules quickly converge to cover most of 18 test queries after BASIL is trained by about four training queries. Therefore, the learning approach can generate general rules that can be applied to a wide range of queries for query optimization.

- The robustness estimation of BASIL achieves a recall score of 92.7 percent and precision of 28.89 percent in the experiment that compares the estimated robustness and the actual inconsistent rules in a new database state generated from 123 transactions. The estimated robustness values allow us to increase the recall score without sacrificing the precision by raising the robustness thresholds. Since the robustness estimation accurately reflects the actual robustness of rules, it allows a learner to minimize the cost of dealing with database changes.

- Comparisons between the performance of the rules learned with different robustness thresholds reinforces the intuition that robustness may interact with effectiveness. The scatterplot of the probability of consistency against the application frequency matches our assumption that the robustness is roughly inversely proportional to the applicability of a rule. The scatterplot of the length against the application frequency shows that long rules are unlikely to be widely applicable.

- Comparisons between different properties of range rules and relational rules provide evidence that relational rules may on average produce higher savings than range rules for query optimization, but they are also on average less robust than range rules.

- The performance data shows that BASIL and PESTO can efficiently learn semantic rules and optimize queries in our experiments.

Chapter 6

Conclusions and Future Work

Fragrant and **robustly** *flavored,* **basil** *is* **pesto***'s key ingredient.*
Its sweet blend of anise, clove and mint is essential
to many French, Southeast Asian,Italian and Greek dishes.
— How to Use Herbs, *The Green House*

This dissertation presents a solution to the knowledge acquisition problem of semantic query optimization and a novel query plan optimization approach that can utilize the learned rules effectively. The integrated learning and optimization approaches allow an information system to reduce the query execution cost significantly. In addition to their contributions to query optimization, the individual approaches developed in this dissertation can potentially solve many important problems in applying machine learning to information system applications, including rule maintenance, learning view definitions and imprecise query answering. Meanwhile, this research raises many questions that deserve further investigation. This chapter summarizes the research results and briefly discusses the potential applications and future work.

6.1 Summary of Results

This section summarizes the results obtained from this research and its contributions.

6.1.1 Robustness of Knowledge

A practical approach to learning from a real-world database must address the issue of database changes. Chapter 2 formalizes the notion of the robustness against database changes by defining the robustness of a rule r in a given database state d as

$$Robust(r|d) = \Pr(\neg t|d) = 1 - \Pr(t|d),$$

where t represents the transactions on d that invalidate r. This definition localizes the database states of concern to those that are accessible from a given database state, and thus allows a learner to estimate the robustness efficiently. The robust estimation problem otherwise would be intractable because a learner must estimate combinatorial numbers of database states that are inconsistent with a rule.

The robustness estimation approach estimates probabilities of rule invalidating transactions in a relational database environment. This approach decomposes the probability of a transactions into local probabilities that can be estimated using Laplace law or m-probability. Users do not need to provide additional information for the estimation because the estimator can utilize information such as transaction logs, database schema, and ranges of attribute values that is available from a database management system. Even if the information is incomplete or unavailable, the approach can still derive a reasonable estimation. Our experiments show that the approach can accurately estimate the robustness of semantic rules.

6.1.2 Learning for Semantic Query Optimization

Chapter 3 presents a two-stage rule induction approach to learning effective and robust rules for semantic query optimization. The first stage of the rule induction is an inductive learning approach that forms a desirable optimized query, called an *alternative query*, for a training query. The second stage generates semantic rules that allow an optimizer to reformulate the training query into the alternative query. The inductive learning approach uses a general minimal set covering heuristic that guarantees a ratio bound $O(\ln l)$, where l is the number of candidate literals, between the execution cost of a resulting alternative query and the optimal execution cost of the equivalent queries composed of the candidate literals. Therefore, semantic rules learned by this approach will allow an optimizer to reformulate a training query into a low cost equivalent query.

The pruning approach used in the second stage prunes literals in a semantic rule so that they are not specific to the training query, but can be applied to a wide class of similar queries. The pruning approach uses a beam search algorithm guided by the estimated robustness of partially pruned rules to search for highly robust and widely applicable rules.

6.1.3 Semantic Optimization of Query Plan

Chapter 4 presents an extension of semantic query optimization to optimize query plans for integrated heterogeneous information sources. Using semantic knowledge allows our approach to significantly reduce redundant data transmission. A key advantage of our approach over previous work in query optimization is that it can infer the range constraints of intermediate data from semantic knowledge and propagate them around a query plan graph. Another advantage of this approach is that it can use a set of axioms to optimize comparisons, set operators and other complex data manipulation operators. As a result, the approach can effectively optimize a wide range of complex queries using learned semantic rules.

A new feature of our learning and optimization approaches is that they can learn and use relational rules. Our experiment confirms that relational rules are useful in semantic query optimization but less robust on average.

6.2 Potential Applications and Extensions

This section briefly discusses potential applications of the ideas developed in this research. These ideas may provide solutions to a wide variety of problems of learning for knowledge-intensive database services. In addition, the robustness of knowledge might provide an answer to a long sought quest in epistemology: the justification of induction.

6.2.1 Machine Learning in Databases

A database system can apply machine learning algorithms to automatically learn required knowledge from data for knowledge-intensive services. However, since the learned knowledge may become inconsistent after changes to the data, a database system may incur an additional overhead of knowledge maintenance in applying machine learning. The robustness estimation approach solves this key problem in applying machine learning in database systems. Though it does not eliminate the need for knowledge maintenance, robustness estimation allows a database system to control and minimize the overhead of maintaining learned knowledge. In addition, the inductive learning approach and the pruning approach can also be applied to learn other types of knowledge from data. We briefly discusses some of specific applications as follows.

Rule maintenance

Rule maintenance involves two issues: identifying inconsistent rules and repairing inconsistent rules. The robustness estimation approach can provide a solution to both issues with minor modifications. For example, the templates of invalidating transactions for a rule used for robustness estimation can be applied to allow a system to identify inconsistent rules efficiently. When a transaction changes the data in a database, the system can identify an inconsistent rule by check whether the transaction implies an invalidating transaction of a rule. We note that a similar approach to identifying inconsistent rules has been proposed and studied substantially for deductive databases [Lloyd, 1987]. The difference is that previous work requires computing the information that allows the system to identify affected rules each time the system performs a transaction. In our case, the information for robustness estimation is ready and can be reused directly for inconsistent rule identification.

To repair an inconsistent rule, the system can guide the repair using estimated robustness of partially pruned rules so that the repaired rule can become more robust and eventually converge on rules that remain consistent. The algorithm for rule repair in this approach will be similar to Algorithm 3.3 for pruning rule literals. A preliminary algorithm is to propose a set of rule repairing operators, such as, changing constants specified in a literal, or eliminating a literal, and then apply the one that yields the most robust new rule until a consistent rule is found.

Learning View Definitions

A *view* in a relational database [Ullman, 1988] is a virtual relation that is not present physically in the database. Data tuples of a view will be extracted from physical relations during query evaluation time. For example, we can define a view called `military-airports` as a subset of the physical relation `airports` for airports used for military purposes. A view is usually defined in the same language as a query. Executing a query to a view efficiently is an important problem. Some views may be expensive to access. In that case, the system can apply the inductive learning approach for alternative queries to learn a less expensive *alternative view* definition from data so that the system can access a view efficiently. The alternative view would be used repeatedly, so even a small improvement could provide significant savings.

Similarly, we can apply the inductive learning algorithm to learn less expensive definitions of information sources [Levy *et al.*, 1996], or less expensive concept definitions in an integrated domain model in an information mediator [Arens *et al.*, 1993, Knoblock *et al.*, 1994, Arens *et al.*, 1996]. In both cases, definitions are expressed as queries. When applying the inductive learning approach to these problems, we can extend the robustness estimation approach to estimate the robustness of learned definitions so that they can be robust against database changes.

Imprecise Query Answering

Imprecise query answering retrieves data that does not precisely satisfy a given query. Examples of imprecise query answering include intentional query answering [Chu and Chen, 1994], which assumes that an input query only specifies a subset of the data that a user intends to retrieve, and query answering with uncertainty, which retrieves data that satisfies a given query along some measure of uncertainty. The approaches developed in this dissertation can potentially be extended for imprecise query answering as follows. The system learns in advance a set of rules and assigns to each rule a certainty factor, such as robustness. When answering a query, the system can reformulate the query as in semantic query optimization, but using those uncertain rules that may not be exactly consistent with the data. The system can control the quality of retrieved data by reasoning about the certainty factors of the rules applied during the query reformulation. This way, a system can achieve desired behavior of query answering depending on the application domain.

6.2.2 Justification of Induction

Robustness of knowledge in its general sense provides a partial solution to the justification of induction raised in [Goodman, 1946]. The problem is to evaluate the two induced rules:

1. All emeralds are green.
2. All emeralds are *grue*, which means green if observed before tomorrow, and blue if observed then after.

Evaluating these statements is difficult because both of them are consistent with the current state of the world. The situation is similar to our learning problem where the learned rules are consistent with a current state of the world, but it is unknown whether the rules will remain consistent in the future. Robustness of knowledge in its general sense can be defined as $1 - \Pr(a)$, where a represents the event that an action is performed to change the world state into a new state in which the given knowledge is invalid.

We apply this general definition of the robustness to evaluate these two statements. Consider the probability of an action that invalidates statement 1, which will become invalid when some action is performed to change the color of all emeralds in the world. Since to change the color of an emerald would require some difficult chemical reaction, it is very unlikely that the color of all emeralds will be changed and statement 1 is thus very robust. On the other hand, an action that invalidate statement 2 is to make sure that nothing happens to one of the emeralds in the world tomorrow to change its color to blue. This is very likely, and statement 2 is thus not robust. Therefore, we should induce statement 1. Certainly, to draw this conclusion we will need to accumulate enough knowledge about actions that change the color of an emerald.

Previously, [Russell, 1989] provided a survey of solutions to the problem and proposes a solution of his own using *determinations*. He argues that since there is a determination between the type of an object and its color, we should prefer statement 1. However, he confesses that the determination may not provide a good justification when we replace emeralds with chameleons in both statements. In contrast, robustness applies to the chameleon case equally well. Since we know that the color of a chameleon can be changed easily if the chameleon moves to a place with a background of different color, when taking this probability into account, the robustness can still provide a good evaluation.

Though Russell's solution may not be as general as ours, unlike early philosophical work that seeks an objective justification, he points out that the justification of an induction should rest on the knowledge of a learner. Using robustness requires a learner to possess the knowledge about actions that change world states. It will be interesting to examine how general robustness can server as the justification of induction.

6.3 Further Inquiries

This research leaves many questions unanswered that deserve further investigation. Some of the questions are given as follows:

- What are the characteristics of queries that can be optimized effectively by semantic query optimization? How often is an input query suitable to be optimized effectively by SQO?
- What are the characteristics of a database state where a query can be optimized effectively by semantic query optimization?
- Is there a bound on the number of rules for an optimizer to achieve certain level of performance? Is there any bound on the number of training queries to learn sufficient semantic rules? Should a semantic query optimizer stop learning?
- How many transactions in a transaction log is sufficient to estimate robustness accurately?

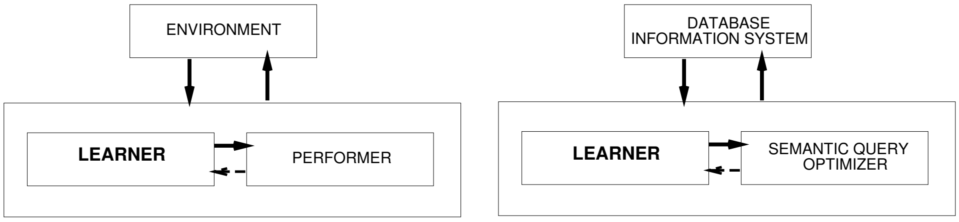

Figure 6.1: Learning for SQO as an instance of general autonomous learning

6.4 Conclusion

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 dissertation provides a solution to the problem with an inductive learning approach that applies robustness estimation to allow the learner to learn effective and robust semantic rules.

In a sense, the problem of learning for SQO fits well in the general framework of autonomous learning (or learning from dynamic environment) [Russell, 1986, Shen, 1989, Laird and Rosenbloom, 1990, Sutton, 1990, Gil, 1992, desJardins, 1992, Kaelbling, 1993, Parr and Russell, 1995], where an agent learns to improve its performance by interacting with a dynamic world where it is situated. Figure 6.1 illustrates the instantiation of learning for SQO in this framework. In our case, the task for the agent is semantic query optimization, and the world around the agent is a software environment of databases and information systems. Compared with previous work in autonomous learning, the learning agent in our problem must deal with a more complex database environment. Though database environments are man-made environments, the learner for SQO must deal with combinatorially many possible database states (on the order of billions or more for a relatively small database) and a large number of different queries. Database environments are beyond the control of the machine learning researchers. It is difficult to fully predict their dynamics. Researchers can not freely assume what type of information or knowledge would be available to the learner in a database environment. Consequently, a solution to the knowledge bottleneck problem of semantic query optimization represents a significant step toward autonomous learning in a software environment.

Appendix A

Terminology

This appendix chapter defines the basic terminology used throughout this dissertation, including databases, queries and semantic knowledge.

A.1 Databases

In this dissertation, we consider databases organized in the relational data model because it is well-defined and widely used in practice. A *database* consists of a set of relations. A *relation* is then a set of instances (or tuples). 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. Table A.1 shows the schema of two example databases. In database `Assets`, the relation `ship_class` stores information about ship classes, and `ship` contains data about individual ships. The other database `Geo` is built for geographic location information. Here we list the schema of two relations on seaports and wharves related to our examples.

Assets database:
```
ship_class(class_name,ship_type,max_draft,length,container_cap),
ship(ship_name,ship_class,status,fleet,year_built).
```
Geo database:
```
geoloc(name,glc_cd,country,latitude,longitude),
seaport(name,glc_code,storage,rail,road,anch_offshore),
wharf(wharf_id,glc_code,depth,length,crane_qty).
```

Table A.1: Schema of example databases

```
1 assets(?ship_class,?draft):-
2    ship_class(?ship_class,_,?draft,_,"Y"),
3    ship(_,?ship_class,"Active",_,_),
4    ?draft < 50.
```

Table A.2: An example query

A.2 Queries

The queries considered in this dissertation are conjunctive Datalog queries, which corresponds to the `select-from-where` subset of SQL.[1] A query begins with a predicate name as the site that the data will be retrieved, followed by a list of parameters. For example, the query in Figure A.2 begins with the site name `assets` followed by the parameters `?ship_class`, `?draft` and `?length`. Variables always start with a `?` mark. We refer to literals on a database relation as *database literals*. For example, the literals listed from line 1 to line 3 in the example query are all database literals. We refer to literals on built-in relations such as `>` and `member`, between a single variable and a set of constants as *built-in literals* (e.g., `?crane > 0`). Literals on built-in relations between two or more variables are *comparisons* (e.g., `?depth > ?draft`). We allow negations for built-in literals and comparisons. In addition to arithmetic comparisons, we also allow predicates for data manipulation operations including disjunctions, set operators (e.g., `intersection` and `union`) and `group-by`s and aggregate operators.

A.3 Semantic Knowledge

Our approach uses two forms of semantic knowledge: *semantic rules* and *range facts*. Semantic rules, represented in terms of Horn-clause rules, express the regularity of data in an individual database. We adopt standard Prolog terminology and semantics as defined in [Lloyd, 1987] in our discussion of rules. Semantic knowledge is interpreted under the closed-world assumption. That is, a database literal is true with regard to a database if and only if in the database there exists an instance that satisfies the literal. Semantic knowledge should be consistent with the database. To distinguish a rule from a query, we show queries using Datalog syntax and semantic rules in a standard logic notation. Table A.3 shows some example semantic rules.

[1]In our implementation, however, queries are expressed in the LOOM knowledge representation language [MacGregor, 1990]. It is also used as the representation language to build an integrated model of databases. For the simplicity of presentation, we choose Datalog to express queries because modeling is not the subject of this dissertation and Datalog is well-known to both the database and AI research communities.

Rules:

```
R1:  If the maximum draft of a ship is less than 50 then its status is active.
 ship_class(?class,_,?draft,_,_) ∧ ship(_,?class,?status,_,_) ∧ ?draft < 50
    ⇒ ?status = "Active"
R2:  If a ship class has container capability, then there must exist some
  ships that belong to that ship class in the database.
 ship_class(?class,_,_,_,"Y") ⇒ ship(_,?class,_,_,_)
R3:  If two seaports share the same geographic location code, than their names are also
  identical.
 seaport(?name1,?code,_,_,_,_) ∧ seaport(?name2,?code,_,_,_,_)
    ⇒ ?name1 = ?name2
R4:  For all the geographic location codes of wharves, there is a seaport with
  the same code.
 wharf(_,?code,_,_,_) ⇒ seaport(_,?code,_,_,_,_)
```

Table A.3: Example semantic rules

We make distinction between two types of rules. The first type, referred to as a *range rule*, are rules with their consequent a positive built-in literal (e.g., R1). The second type consists of rules with their consequent a database literal (e.g., R2), referred to as a *relational rule*. Range rules are useful for inferring ranges of attribute variables during the global optimization of multidatabase queries. They are also useful for deriving redundant built-in literals for deletions and introducing cost-reducing built-in literals for insertions. Relational rules are used to derive redundant retrievals and joins to a database relation. They can also be used for introducing cost-reducing relations.

Horn-clause rules can express a variety of database integrity constraints. R4 is an example of a referential integrity constraint, a special case of relational rule. A functional dependency can also be expressed in Horn-clause such as R3. Functional dependencies are useful in inferring a precise range of a variable so as to delete or insert a beneficial literal to a query.

Generally speaking, a rule is *applicable* to a query if the antecedent part of the rule is a logical consequence of the literals of Q. This can be determined by applying SLD-refutation [Lloyd, 1987] to a target rule and query, where the goal comprises the antecedents of R and the body of the query corresponds to the program clauses. Unlike general theorem proving, the program clauses only contains query literals. They will never induce new subgoals and the refutation process is decidable and can be computed efficiently. We note that the use of SLD-refutation is the reason why our approach can apply any Horn-clause rules in optimization, an advantage over previous work.

Range facts state the range of the values of a given database attribute. For numeral attributes, their range facts show the minimal value and the maximal value. For string-typed attributes, their range facts enumerate the possible values. In our

Range Facts:

```
F1: 12 ≤ ship_class.max_draft ≤ 72.
F2: 325 ≤ ship_class.length ≤ 950.
F3: ship_class.container_cap ∈ {"Y","N"}.
F4: ship.status ∈ {"Active","Inactive","Resigned"}.
```

Table A.4: Range facts state the range of attribute values

implementation, range facts of a string-typed attribute will not be used if there are more than 20 possible values. Table A.4 gives some examples.

A.4 Database States and Transactions

A *database state* at a given time `t` is the collection of the instances present in the database at the time `t`. We use the *closed-world assumption* (CWA) to interpret the semantics of a database state. That is, information not explicitly present in the database is taken to be false. A rule is said to be *consistent* with a database state if all variable instantiations that satisfy the antecedents of the rule also satisfy the consequent of the rule. A straightforward approach to identifying an inconsistent rule is to transform the negation of a rule $C \leftarrow A$ into a query $\neg C \wedge A$ and send it to the database system. If the query returns an answer that is not empty, then the rule is inconsistent.

A database can be changed by *transactions*. A transaction can be considered as a mapping from a database state to a new database state. There are three kinds of *primitive transactions* — inserting a new tuple into a relation, deleting an existing tuple from a relation, and updating an existing tuple in a relation.

Appendix B

Templates of Robustness Estimates

This appendix describes the complete templates for two classes of Horn-clause rules: range rules and relational rules. For each class, a set of transaction templates and templates of estimates for the probability of those transactions are presented.

Before presenting the templates, we first explain the terminology and notation. We use a notation to represent the repeatedly used parameters in the templates. Table B.1 gives this notation. Parameters of the form $N(\ldots)$ can be obtained by counting data in a given database. To evaluate the parameters of the form $\tau(\ldots)$ needs to access transaction log information. When no transaction log available, their default values are zero. Also, if a literal is of the form $A(\ldots, ?x, \ldots)$, then $A.x$ is used to denote the attribute of A where $?x$ values are instantiated.

B.1 Range Rules

Consider a range rule of the form

$$\theta(?x) \Longleftarrow \bigwedge_{1 \le i \le I} A_i \wedge \bigwedge_{1 \le j \le J} B_j \wedge \bigwedge_{1 \le k \le K} L_k,$$

where θ is a predicate composed of a built-in predicate and constants (e.g., we can define $\theta(?x) \equiv ?x \geq 100$), A_i's are database literals where $?x$ occurs, and L_k's are built-in literals. Three mutually exclusive classes of invalidating transactions and the templates to estimate their probabilities are given as follows.

- **T1:** Update a tuple of A_i or B_j covered by the rule so that a new $?x$ value satisfies the antecedent but does not satisfy $\theta(?x)$.

$$\Pr(T1) = \sum_{1 \le i \le I} u(A_i, A_i.x) + \sum_{1 \le j \le J} \sum_{y \in \mathrm{Attr}(B_j)} u(B_j, y)$$

where for a relation A and one of its attribute $A.z$,

$$u(A, A.z) = u_1 \cdot u_2 \cdot u_3 \cdot u_4 \cdot u_5,$$

Symbol	Meaning
$\mathcal{A}$	Antecedents of the rule
$\mathcal{R}$	Number of relations
$\mathrm{Attr}(A)$	The set of attributes of a relation A
$N(A)$	Number of tuples in a relation A
$N(A\|\varphi)$	Number of tuples in a relation A, satisfying a set of literals φ
$\Pr(A \in \varphi)$	Probability that an A tuple satisfies φ
$\Pr(A.a \in \varphi)$	Probability that the value of an attribute $A.a$ satisfies φ
τ	Number of all transactions
$\tau(t)$	Number of a certain type of transactions (e.g., updates)
$\tau(t, A)$	Number of a certain type of transactions on a relation A
$\tau(t, A\|\varphi)$	Number of a certain type of transactions on A satisfying φ
$\tau(t, A.a)$	Number of a certain type of transactions on some attribute $A.a$

Table B.1: Notation

$$u_1 = \frac{\tau(update) + 1}{\tau + 3}$$
$$u_2 = \frac{\tau(update, A) + 1}{\tau(update) + \mathcal{R}}$$
$$u_3 = \frac{\tau(update, A|\mathcal{A}) + 1}{\tau(update, A) + \frac{N(A)}{N(A|\mathcal{A})}}$$
$$u_4 = \frac{\tau(update, A.z) + 1}{\tau(update, A) + |\mathrm{Attr}(A)|}$$
$$u_5 = \Pr(A.z \in \neg\theta(?x) \wedge \mathcal{A}).$$

- **T2:** Insert a new tuple to a relation A_i or B_j so that the tuple satisfies all the antecedents but not $\theta(?x)$.

$$\Pr(T2) = \sum_{1 \le i \le I} s(A_i) + \sum_{1 \le j \le J} s(B_j),$$

where for a relation A,

$$s(A) = s_1 \cdot s_2 \cdot s_3,$$
$$s_1 = \frac{\tau(insert) + 1}{t + 3}$$
$$s_2 = \frac{\tau(insert, A) + 1}{\tau(insert) + \mathcal{R}}$$
$$s_3 = \Pr(A \in \neg\theta(?x) \wedge \mathcal{A}).$$

- **T3:** Update one tuple of a relation A_i or B_j not covered by the rule so that the resulting tuple satisfies all the antecedents but not $\theta(?x)$.

$$\Pr(T3) = \sum_{1 \leq i \leq I} \sum_{w \in \mathrm{Attr}(A_i) - \{A.x\}} v(A_i, w) + \sum_{1 \leq j \leq J} \sum_{y \in \mathrm{Attr}(B_j)} v(B_j, y),$$

where for a relation A and its attribute $A.z$,

$$v(A, A.z) = v_1 \cdot v_2 \cdot v_3 \cdot v_4 \cdot v_5,$$

$$v_1 = \frac{\tau(update) + 1}{\tau + 3}$$

$$v_2 = \frac{\tau(update, A) + 1}{\tau(update) + \mathcal{R}}$$

$$v_3 = 1 - \frac{\tau(update, A|\mathcal{A} \wedge \theta(?x)) + 1}{\tau(update, A) + \frac{N(A)}{N(A|\theta(?x) \wedge \mathcal{A})}}$$

$$v_4 = \frac{\tau(update, A.z) + 1}{\tau(update, A) + |\mathrm{Attr}(A)|}$$

$$v_5 = \Pr(A.z \in \neg\theta(?x) \wedge \mathcal{A}).$$

B.2 Relational rules

Consider a relational rule of the form

$$C(?x) \Longleftarrow \bigwedge_{1 \leq i \leq I} A_i \wedge \bigwedge_{1 \leq j \leq J} B_j \wedge \bigwedge_{1 \leq k \leq K} L_k,$$

where $C(?x)$ is the abbreviation of $C(\ldots, ?x, \ldots)$ for a relation C in the database, A_i and B_j's are database literals, and L_k's are built-in literals. Five mutually exclusive classes of invalidating transactions and the templates to estimate their probabilities are given as follows.

- **T1:** Update an attribute of a tuple of A_i or B_j covered by the rule so that the new value allows a new $?x$ value satisfies the antecedents but not the consequent.

$$\Pr(T1) = \sum_{1 \leq i \leq I} u(A_i, A_i.x) + \sum_{1 \leq j \leq J} \sum_{y \in \mathrm{Attr}(B_j)} u(B_j, y)$$

where for a relation A and one of its attribute $A.z$,

$$u(A, A.z) = u_1 \cdot u_2 \cdot u_3 \cdot u_4 \cdot u_5,$$

$$u_1 = \frac{\tau(update)+1}{\tau+3}$$
$$u_2 = \frac{\tau(update, A)+1}{\tau(update)+\mathcal{R}}$$
$$u_3 = \frac{\tau(update, A|\mathcal{A})+1}{\tau(update, A)+\frac{N(A)}{N(A|\mathcal{A})}}$$
$$u_4 = \frac{\tau(update, A.z)+1}{\tau(update, A)+|\mathrm{Attr}(A)|}$$
$$u_5 = \Pr(A.z \in \neg C(?x) \wedge \mathcal{A}).$$

- **T2:** Insert a new tuple to A_i or B_j so that the new tuple allows a new $?x$ value satisfies the antecedents but not the consequent.

$$\Pr(T2) = \sum_{1\leq i\leq I} s(A_i) + \sum_{1\leq j\leq J} s(B_j),$$

where for a relation A,

$$s(A) = s_1 \cdot s_2 \cdot s_3,$$
$$s_1 = \frac{\tau(insert)+1}{t+3}$$
$$s_2 = \frac{\tau(insert, A)+1}{\tau(insert)+\mathcal{R}}$$
$$s_3 = \Pr(A \in \neg C(?x) \wedge \mathcal{A}).$$

- **T3:** Update an attribute of a tuple of A_i or B_j not covered by the rule so that the new value allows a new $?x$ value satisfies the antecedents but not the consequent.

$$\Pr(T3) = \sum_{1\leq i\leq I} \sum_{w\in \mathrm{Attr}(A_i)-\{A.x\}} v(A_i, w) + \sum_{1\leq j\leq J} \sum_{y\in \mathrm{Attr}(B_j)} v(B_j, y),$$

where for a relation A and its attribute $A.z$,

$$v(A, A.z) = v_1 \cdot v_2 \cdot v_3 \cdot v_4 \cdot v_5,$$
$$v_1 = \frac{\tau(update)+1}{\tau+3}$$
$$v_2 = \frac{\tau(update, A)+1}{\tau(update)+\mathcal{R}}$$

$$v_3 = 1 - \frac{\tau(update, A|\mathcal{A} \wedge C(?x)) + 1}{\tau(update, A) + \frac{N(A)}{N(A|C(?x) \wedge \mathcal{A})}}$$

$$v_4 = \frac{\tau(update, A.z) + 1}{\tau(update, A) + |\mathrm{Attr}(A)|}$$

$$v_5 = \Pr(A.z \in \neg C(?x) \wedge \mathcal{A}).$$

- **T4:** Update $C.x$ of all C tuples that share a certain $C.x$ value that satisfies the antecedents to a new value that does not satisfies the antecedents.

$$\Pr(T4) = \sum_{X \in \mathcal{I}_x} p(X)^{N(C|C.x=X)},$$

where $\mathcal{I}_x$ is the set of distinct values of $C.x$, and $p(X)$ is the probability that the update is applied to C tuples whose $C.x = X$. $\Pr(T4)$ can be approximated by

$$\Pr(T4) \le n_2 \cdot p(Y)^{n_1},$$

where n_1 is the minimal number of C tuples that is grouped by the same $C.x$ value denoted as Y, n_2 is the number of all distinct $C.x$ values, and

$$p(Y) = p_1 \cdot p_2 \cdot p_3 \cdot p_4 \cdot p_5,$$

where

$$p_1 = \frac{\tau(update) + 1}{\tau + 3}$$

$$p_2 = \frac{\tau(update, C) + 1}{\tau(update) + \mathcal{R}}$$

$$p_3 = \frac{\tau(update, C|C.x = Y) + 1}{\tau(update, C) + \frac{N(C)}{N(C|C.x=Y)}}$$

$$p_4 = \frac{\tau(update, C.x) + 1}{\tau(update, C) + |\mathrm{Attr}(C)|}$$

$$p_5 = \Pr(C.x \in \neg\mathcal{A}).$$

- **T5:** Delete all C tuples that share a certain $C.x$ value that satisfies the antecedents of the rule.

$$\Pr(T5) = \sum_{X \in \mathcal{I}_x} d(X)^{N(C|C.x=X)},$$

where $\mathcal{I}_x$ is the set of distinct values of $C.x$, and $d(X)$ is the probability that the deletion is applied to C tuples whose $C.x = X$. $\Pr(T4)$ can be approximated by

$$\Pr(T5) \le n_2 \cdot d(Y)^{n_1},$$

where n_1 is the minimal number of C tuples that is grouped by the same $C.x$ value denoted as Y, n_2 is the number of all distinct $C.x$ values, and

$$d(Y) = d_1 \cdot d_2 \cdot d_3,$$

where

$$d_1 = \frac{\tau(delete) + 1}{\tau + 3}$$

$$d_2 = \frac{\tau(delete, C) + 1}{\tau(delete) + \mathcal{R}}$$

$$d_3 = \frac{\tau(delete, C|C.x = Y) + 1}{\tau(delete, C) + \frac{N(C)}{N(C|C.x=Y)}}.$$

Appendix C

Optimization and Learning of Sample Query

This appendix gives in detail the output produced by the optimization and learning for a sample multidatabase query. The data provided here include an original input query, subqueries generated by the SIMS query planner, subqueries optimized by PESTO, a trace of rule and range fact application in the optimization, and semantic rules and their estimated robustness learned by BASIL using one of the subqueries as the training trigger query. The queries and rules are expressed in the LOOM knowledge representation language [MacGregor, 1990]. Please refer to the SIMS manual [Ambite *et al.*, 1995] for a formal definition of the syntax of the LOOM language.

```
;;;;; Input query:

(214 "List all wharves at Long Beach with RORO ramps, by pier name and berth
  ID where METEOR ships can dock"
  (sims-retrieve (?PNAME ?BERTHID)
       (:AND  (SHIP_CLASS ?SHIP)
              (SHIP_CLASS.SH_CLASS ?SHIP "METEOR")
              (SHIP_CLASS.LENGTH ?SHIP ?SH-LENGTH)
              (SHIP_CLASS.BEAM ?SHIP ?BEAM)
              (SHIP_CLASS.MAX_DRAFT ?SHIP ?DRAFT)
              (SHIP_CLASS.HEIGHT ?SHIP ?SH-HEIGHT)
              (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#75578
                        ?CH-NAME)
              (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#75578
                        ?CODE75550)
              (SEAPORTS ?SPORT)
              (SEAPORTS.GLC_CD ?SPORT ?TEMP75623)
              (SEAPORTS.PORT_NM ?SPORT "Long Beach")
              (CHANNELS ?CHANNEL)
              (CHANNELS.GLC_CD ?CHANNEL ?TEMP75624)
              (CHANNELS.CH_NM ?CHANNEL ?TEMP75625)
              (CHANNELS.CH_WIDTH_FT ?CHANNEL ?CH-WIDTH)
              (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
              (CHANNELS.CH_OVERHEAD_CLEARANCE ?CHANNEL ?CH-HEIGHT)
              (PIERS ?PIER)
              (PIERS.GLC_CD ?PIER ?TEMP75626)
              (PIERS.PIER_NM ?PIER ?PNAME)
              (WHARVES ?WHARF)
              (WHARVES.GLC_CD ?WHARF ?TEMP75627)
              (WHARVES.STERN_RAMP_FLAG ?WHARF "Y")
              (WHARVES.WHARF_BERTH_ID ?WHARF ?BERTHID)
              (WHARVES.WHARF_LENGTH_FT ?WHARF ?WH-LENGTH)
              (WHARVES.WHARF_WATER_DEPTH_FT ?WHARF ?WH-DEPTH)
              (= ?TEMP75623 ?CODE75550)
              (= ?TEMP75625 ?CH-NAME)
              (= ?TEMP75624 ?CODE75550)
              (= ?TEMP75626 ?CODE75550)
              (= ?TEMP75627 ?CODE75550)
              (= ?TEMP75624 ?TEMP75623)
              (= ?TEMP75626 ?TEMP75623)
              (= ?TEMP75627 ?TEMP75623)
              (= ?TEMP75626 ?TEMP75624)
              (= ?TEMP75627 ?TEMP75624)
```

```
                (= ?TEMP75627 ?TEMP75626)))


;;;;; Subqueries extracted by SIMS planner:

(#plan<S=12; O=0; U=0>
 (11  ;; step id of subquery
  (RETRIEVE (?SH-HEIGHT ?BEAM ?DRAFT ?SH-LENGTH)
   (:AND (SHIP_CLASS ?SHIP)
        (SHIP_CLASS.SH_CLASS ?SHIP "METEOR")
        (SHIP_CLASS.LENGTH ?SHIP ?SH-LENGTH)
        (SHIP_CLASS.BEAM ?SHIP ?BEAM)
        (SHIP_CLASS.MAX_DRAFT ?SHIP ?DRAFT)
        (SHIP_CLASS.HEIGHT ?SHIP ?SH-HEIGHT))))
 (9
  (RETRIEVE
   (?PNAME ?BERTHID
         ?CH-HEIGHT
         ?CH-DEPTH
         ?CH-WIDTH
         ?WH-DEPTH
         ?WH-LENGTH)
   (:AND (CHANNEL_HARBORS ?CHANNEL-HARBOR#75578)
       (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#75578
                         ?CH-NAME)
       (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#75578
                       ?CODE75550)
       (SEAPORTS ?SPORT)
       (SEAPORTS.GLC_CD ?SPORT ?TEMP75623)
       (SEAPORTS.PORT_NM ?SPORT "Long Beach")
       (CHANNELS ?CHANNEL)
       (CHANNELS.GLC_CD ?CHANNEL ?TEMP75624)
       (CHANNELS.CH_NM ?CHANNEL ?TEMP75625)
       (CHANNELS.CH_WIDTH_FT ?CHANNEL ?CH-WIDTH)
       (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
       (CHANNELS.CH_OVERHEAD_CLEARANCE ?CHANNEL ?CH-HEIGHT)
       (PIERS ?PIER)
       (PIERS.GLC_CD ?PIER ?TEMP75626)
       (PIERS.PIER_NM ?PIER ?PNAME)
       (WHARVES ?WHARF)
       (WHARVES.GLC_CD ?WHARF ?TEMP75627)
       (WHARVES.STERN_RAMP_FLAG ?WHARF "Y")
       (WHARVES.WHARF_BERTH_ID ?WHARF ?BERTHID)
       (WHARVES.WHARF_LENGTH_FT ?WHARF ?WH-LENGTH)
       (WHARVES.WHARF_WATER_DEPTH_FT ?WHARF ?WH-DEPTH)
       (= ?TEMP75623 ?CODE75550)
       (= ?TEMP75625 ?CH-NAME)
       (= ?TEMP75624 ?CODE75550)
       (= ?TEMP75626 ?CODE75550)
       (= ?TEMP75627 ?CODE75550)
       (= ?TEMP75624 ?TEMP75623)
       (= ?TEMP75626 ?TEMP75623)
       (= ?TEMP75627 ?TEMP75623)
       (= ?TEMP75626 ?TEMP75624)
       (= ?TEMP75627 ?TEMP75624)
       (= ?TEMP75627 ?TEMP75626))))
 (7
  ((< ?SH-HEIGHT ?CH-HEIGHT)
   (< ?DRAFT ?CH-DEPTH)
   (< ?BEAM ?CH-WIDTH)
   (< ?DRAFT ?WH-DEPTH)
   (<= ?SH-LENGTH ?WH-LENGTH))))



;;;;; Subqueries optimized by PESTO query optimizer:

(#plan<S=12; O=0; U=0>
  (11
   (RETRIEVE (?SH-HEIGHT ?DRAFT)
          (:AND (SHIP_CLASS ?SHIP)
               (SHIP_CLASS.SH_CLASS ?SHIP "METEOR")
               (SHIP_CLASS.HEIGHT ?SHIP ?SH-HEIGHT)
               (SHIP_CLASS.MAX_DRAFT ?SHIP ?DRAFT)
               (< ?DRAFT 37))))  ;; additional built-in constraint
  (9
   ;; note the literals and joins being removed
   (RETRIEVE (?PNAME ?BERTHID ?CH-HEIGHT ?WH-DEPTH)
          (:AND (CHANNELS ?CHANNEL)
               (CHANNELS.GLC_CD ?CHANNEL ?TEMP75624)
               (CHANNELS.CH_OVERHEAD_CLEARANCE ?CHANNEL ?CH-HEIGHT)
               (PIERS ?PIER)
               (PIERS.GLC_CD ?PIER ?TEMP75626)
               (PIERS.PIER_NM ?PIER ?PNAME)
               (WHARVES ?WHARF)
               (WHARVES.GLC_CD ?WHARF ?TEMP75627)
               (WHARVES.STERN_RAMP_FLAG ?WHARF "Y")
```

```
            (WHARVES.WHARF_BERTH_ID ?WHARF ?BERTHID)
            (WHARVES.PIER_NM ?WHARF ?R16)
            (WHARVES.WHARF_WATER_DEPTH_FT ?WHARF ?WH-DEPTH)
            (= ?TEMP75627 ?TEMP75626)
            (= ?TEMP75626 ?TEMP75624)
            (MEMBER ?R16 ("B" "C" "D")) ;; additional built-in constraint
            (<= ?WH-DEPTH 37))))        ;; additional built-in constraint
 (7
  ((< ?SH-HEIGHT ?CH-HEIGHT) (< ?DRAFT ?WH-DEPTH))))


;;;;; rule and range fact application trace of PESTO's optimization:

"==>
(:AND (>= ?SH-HEIGHT 102)
     (<= ?SH-HEIGHT 213))
 is INFERRED from range information:
(:IN SHIP_CLASS.HEIGHT (:THROUGH 102 213))
==>
(:AND (>= ?DRAFT 12) (<= ?DRAFT 40))
 is INFERRED from range information:
(:IN SHIP_CLASS.MAX_DRAFT (:THROUGH 12 40))
==>
(:AND (>= ?BEAM 56) (<= ?BEAM 106))
 is INFERRED from range information:
(:IN SHIP_CLASS.BEAM (:THROUGH 56 106))
==>
(:AND (>= ?SH-LENGTH 273)
     (<= ?SH-LENGTH 947))
 is INFERRED from range information:
(:IN SHIP_CLASS.LENGTH (:THROUGH 273 947))
==>
A NEW cost-reducing concept and relational join
(:AND (GEOLOC ?GEOGRAPHIC-LOCATION#7549)
     (GEOLOC.GLC_CD ?GEOGRAPHIC-LOCATION#7549 ?CODE7544)
     (= ?CODE7544 ?TEMP1733))
 are DERIVED from:
23100006:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP7555))
(:THEN (GEOLOC ?GEOGRAPHIC-LOCATION#7549)
      (GEOLOC.GLC_CD ?GEOGRAPHIC-LOCATION#7549 ?CODE7544)
      (= ?CODE7544 ?TEMP7555))
==>
A NEW cost-reducing clause
(:AND (>= ?CH-WIDTH 260)
     (<= ?CH-WIDTH 1800))
 is DERIVED from:
21300000:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP16189)
    (CHANNELS.CH_NM ?CHANNEL ?CHANNEL-NAME)
    (CHANNELS.CH_WIDTH_FT ?CHANNEL ?CHANNEL-WIDTH))
(:THEN (>= ?CHANNEL-WIDTH 260)
      (<= ?CHANNEL-WIDTH 1800))
==>
A NEW cost-reducing clause
(:AND (>= ?CH-HEIGHT 155)
     (<= ?CH-HEIGHT 999))
 is DERIVED from:
21300001:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP16189)
    (CHANNELS.CH_NM ?CHANNEL ?CHANNEL-NAME)
    (CHANNELS.CH_OVERHEAD_CLEARANCE ?CHANNEL
                          ?CHANNEL-LEAST-OVERHEAD-CLEAR-HEIGHT))
(:THEN (>= ?CHANNEL-LEAST-OVERHEAD-CLEAR-HEIGHT
         155)
      (<= ?CHANNEL-LEAST-OVERHEAD-CLEAR-HEIGHT 999))
==>
A NEW cost-reducing clause
((= ?TEMP1733 "NPTU"))
 is DERIVED from:
23101007:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP7553))
(:THEN (= ?TEMP7553 "NPTU"))
==>
((= ?TEMP1733 "NPTU"))
 is REDUNDANT from range information:
((= ?TEMP1733 "NPTU"))
==>
((= ?CODE1659 "NPTU"))
 is INFERRED from range information:
(:IN CHANNEL_HARBORS.GLC_CD (:ONE-OF "NPTU"))
==>
```

```
((MEMBER ?CH-NAME
        ("Back" "Long Beach" "Queens Gate")))
 is INFERRED from range information:
(:IN CHANNEL_HARBORS.LEAD_CH_NM (:ONE-OF "Back" "Long Beach" "Queens Gate"))
==>
(:AND (>= ?CH-WIDTH 260)
     (<= ?CH-WIDTH 1800))
 is INFERRED from range information:
(:IN CHANNELS.CH_WIDTH_FT (:THROUGH 260 1800))
==>
(:AND (>= ?CH-HEIGHT 155)
     (<= ?CH-HEIGHT 999))
 is INFERRED from range information:
(:IN CHANNELS.CH_OVERHEAD_CLEARANCE (:THROUGH 155 999))
==>
((= ?TEMP1733 "NPTU"))
 is INFERRED from range information:
(:IN CHANNELS.GLC_CD (:ONE-OF "NPTU"))
==>
(:AND (>= ?CH-DEPTH 41)
     (<= ?CH-DEPTH 60))
 is INFERRED from range information:
(:IN CHANNELS.CH_DEPTH_FT (:THROUGH 41 60))
==>
((MEMBER ?TEMP1734
        ("Back" "Cerritos"
                "Long Beach"
                "Queens Gate")))
 is INFERRED from range information:
(:IN CHANNELS.CH_NM (:ONE-OF "Back" "Cerritos" "Long Beach" "Queens Gate"))
==>
((MEMBER ?PNAME
        ("1" "2" "A" "B" "C" "D" "F" "G" "J")))
 is INFERRED from range information:
(:IN PIERS.PIER_NM (:ONE-OF "1" "2" "A" "B" "C" "D" "F" "G" "J"))
==>
((= ?TEMP1735 "NPTU"))
 is INFERRED from range information:
(:IN PIERS.GLC_CD (:ONE-OF "NPTU"))
==>
(:AND (>= ?WH-DEPTH 30)
     (<= ?WH-DEPTH 50))
 is INFERRED from range information:
(:IN WHARVES.WHARF_WATER_DEPTH_FT (:THROUGH 30 50))
==>
(:AND (>= ?WH-LENGTH 580)
     (<= ?WH-LENGTH 2700))
 is INFERRED from range information:
(:IN WHARVES.WHARF_LENGTH_FT (:THROUGH 580 2700))
==>
((MEMBER ?TEMP1856 ("N" "Y")))
 is INFERRED from range information:
(:IN WHARVES.STERN_RAMP_FLAG (:ONE-OF "N" "Y"))
==>
((= ?TEMP1736 "NPTU"))
 is INFERRED from range information:
(:IN WHARVES.GLC_CD (:ONE-OF "NPTU"))
==>
(> ?CH-WIDTH ?BEAM)
 is DELETED because
(:AND (>= ?CH-WIDTH 260)
     (<= ?CH-WIDTH 1800))
(:AND (>= ?BEAM 56) (<= ?BEAM 106))
==>
(> ?CH-DEPTH ?DRAFT)
 is DELETED because
(:AND (>= ?CH-DEPTH 41)
     (<= ?CH-DEPTH 60))
(:AND (>= ?DRAFT 12) (<= ?DRAFT 40))
==>
A NEW cost-reducing concept and relational join
(:AND (GEOLOC ?GEOGRAPHIC-LOCATION#7549)
     (GEOLOC.GLC_CD ?GEOGRAPHIC-LOCATION#7549 ?CODE7544)
     (= ?CODE7544 ?TEMP1733))
 are DERIVED from:
23100006:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP7555))
(:THEN (GEOLOC ?GEOGRAPHIC-LOCATION#7549)
      (GEOLOC.GLC_CD ?GEOGRAPHIC-LOCATION#7549 ?CODE7544)
      (= ?CODE7544 ?TEMP7555))
==>
A NEW cost-reducing clause
(:AND (>= ?CH-WIDTH 260)
     (<= ?CH-WIDTH 1800))
 is DERIVED from:
21300000:
```

```
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP16189)
    (CHANNELS.CH_NM ?CHANNEL ?CHANNEL-NAME)
    (CHANNELS.CH_WIDTH_FT ?CHANNEL ?CHANNEL-WIDTH))
(:THEN (>= ?CHANNEL-WIDTH 260)
      (<= ?CHANNEL-WIDTH 1800))
==>
A NEW cost-reducing clause
(:AND (>= ?CH-HEIGHT 155)
     (<= ?CH-HEIGHT 999))
 is DERIVED from:
21300001:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP16189)
    (CHANNELS.CH_NM ?CHANNEL ?CHANNEL-NAME)
    (CHANNELS.CH_OVERHEAD_CLEARANCE ?CHANNEL
                              ?CHANNEL-LEAST-OVERHEAD-CLEAR-HEIGHT))
(:THEN (>= ?CHANNEL-LEAST-OVERHEAD-CLEAR-HEIGHT
          155)
      (<= ?CHANNEL-LEAST-OVERHEAD-CLEAR-HEIGHT 999))
==>
A NEW cost-reducing clause
((= ?TEMP1733 ”NPTU”))
 is DERIVED from:
23101007:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP7553))
(:THEN (= ?TEMP7553 ”NPTU”))
==>
((= ?CODE1659 ”NPTU”))
 is INFERRED from range information:
(:IN CHANNEL_HARBORS.GLC_CD (:ONE-OF ”NPTU”))
==>
((MEMBER ?CH-NAME
        (”Back” ”Long Beach” ”Queens Gate”)))
 is INFERRED from range information:
(:IN CHANNEL_HARBORS.LEAD_CH_NM (:ONE-OF ”Back” ”Long Beach” ”Queens Gate”))
==>
(:AND (>= ?CH-HEIGHT 155)
     (<= ?CH-HEIGHT 999))
 is INFERRED from range information:
(:IN CHANNELS.CH_OVERHEAD_CLEARANCE (:THROUGH 155 999))
==>
(:AND (>= ?CH-DEPTH 41)
     (<= ?CH-DEPTH 60))
 is INFERRED from range information:
(:IN CHANNELS.CH_DEPTH_FT (:THROUGH 41 60))
==>
(:AND (>= ?CH-WIDTH 260)
     (<= ?CH-WIDTH 1800))
 is INFERRED from range information:
(:IN CHANNELS.CH_WIDTH_FT (:THROUGH 260 1800))
==>
((MEMBER ?TEMP1734
        (”Back” ”Cerritos”
               ”Long Beach”
               ”Queens Gate”)))
 is INFERRED from range information:
(:IN CHANNELS.CH_NM (:ONE-OF ”Back” ”Cerritos” ”Long Beach” ”Queens Gate”))
==>
((= ?TEMP1733 ”NPTU”))
 is INFERRED from range information:
(:IN CHANNELS.GLC_CD (:ONE-OF ”NPTU”))
==>
((MEMBER ?PNAME
        (”1” ”2” ”A” ”B” ”C” ”D” ”F” ”G” ”J”)))
 is INFERRED from range information:
(:IN PIERS.PIER_NM (:ONE-OF ”1” ”2” ”A” ”B” ”C” ”D” ”F” ”G” ”J”))
==>
((= ?TEMP1735 ”NPTU”))
 is INFERRED from range information:
(:IN PIERS.GLC_CD (:ONE-OF ”NPTU”))
==>
(:AND (>= ?WH-DEPTH 30)
     (<= ?WH-DEPTH 50))
 is INFERRED from range information:
(:IN WHARVES.WHARF_WATER_DEPTH_FT (:THROUGH 30 50))
==>
(:AND (>= ?WH-LENGTH 580)
     (<= ?WH-LENGTH 2700))
 is INFERRED from range information:
(:IN WHARVES.WHARF_LENGTH_FT (:THROUGH 580 2700))
==>
((MEMBER ?TEMP1856 (”N” ”Y”)))
 is INFERRED from range information:
(:IN WHARVES.STERN_RAMP_FLAG (:ONE-OF ”N” ”Y”))
==>
```

```
((= ?TEMP1736 "NPTU"))
 is INFERRED from range information:
(:IN WHARVES.GLC_CD (:ONE-OF "NPTU"))
==>
(:AND (>= ?SH-HEIGHT 102)
     (<= ?SH-HEIGHT 213))
 is INFERRED from range information:
(:IN SHIP_CLASS.HEIGHT (:THROUGH 102 213))
==>
(:AND (>= ?DRAFT 12) (<= ?DRAFT 40))
 is INFERRED from range information:
(:IN SHIP_CLASS.MAX_DRAFT (:THROUGH 12 40))
==>
(:AND (>= ?BEAM 56) (<= ?BEAM 106))
 is INFERRED from range information:
(:IN SHIP_CLASS.BEAM (:THROUGH 56 106))
==>
(:AND (>= ?SH-LENGTH 273)
     (<= ?SH-LENGTH 947))
 is INFERRED from range information:
(:IN SHIP_CLASS.LENGTH (:THROUGH 273 947))
"
>


;;;;; Semantic rules learned by BASIL using query 214 as training query

21400000:
(:IF (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (= ?TEMP3920 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (>= ?CH-DEPTH 45)
      (<= ?CH-DEPTH 60))
(robustness = 0.9966760129361756)
21400001:
(:IF (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923)
    (WHARVES.STERN_RAMP_FLAG ?WHARF ?TEMP4043)
    (= ?TEMP3923 ?TEMP3920)
    (= ?TEMP3920 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (>= ?CH-DEPTH 45)
      (<= ?CH-DEPTH 60))
(robustness = 0.9960218766038714)
21400002:
(:IF (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (PIERS ?PIER)
    (PIERS.GLC_CD ?PIER ?TEMP3922)
    (= ?TEMP3922 ?TEMP3920)
    (= ?TEMP3920 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (>= ?CH-DEPTH 45)
      (<= ?CH-DEPTH 60))
(robustness = 0.9960983269085316)
21400003:
(:IF (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923)
    (WHARVES.STERN_RAMP_FLAG ?WHARF ?TEMP4043)
    (= ?TEMP4043 "Y")
    (= ?TEMP3923 ?TEMP3920)
    (= ?TEMP3920 ?CODE3846)
```

```
    (= ?TEMP3921 ?CH-NAME))
(:THEN (>= ?CH-DEPTH 45)
      (<= ?CH-DEPTH 60))
(robustness = 0.9960238466742531)
21400004:
(:IF (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (PIERS ?PIER)
    (PIERS.GLC_CD ?PIER ?TEMP3922)
    (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923)
    (WHARVES.STERN_RAMP_FLAG ?WHARF ?TEMP4043)
    (= ?TEMP3923 ?TEMP3922)
    (= ?TEMP3922 ?TEMP3920)
    (= ?TEMP3920 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (>= ?CH-DEPTH 45)
      (<= ?CH-DEPTH 60))
(robustness = 0.9954441905762275)
21400005:
(:IF (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (PIERS ?PIER)
    (PIERS.GLC_CD ?PIER ?TEMP3922)
    (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923)
    (WHARVES.STERN_RAMP_FLAG ?WHARF ?TEMP4043)
    (= ?TEMP4043 "Y")
    (= ?TEMP3923 ?TEMP3922)
    (= ?TEMP3922 ?TEMP3920)
    (= ?TEMP3920 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (>= ?CH-DEPTH 45)
      (<= ?CH-DEPTH 60))
(robustness = 0.9954441905762275)
21400006:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45))
(:THEN (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
      (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
      (= ?CH-NAME ?TEMP3921))
(robustness = 0.9251198590629485)
21400007:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
    (PIERS ?PIER)
    (PIERS.GLC_CD ?PIER ?TEMP3922)
    (= ?TEMP3922 ?TEMP3920))
(:THEN (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
      (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
      (= ?CH-NAME ?TEMP3921))
(robustness = 0.9245421730353045)
21400008:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
    (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923)
    (= ?TEMP3923 ?TEMP3920))
(:THEN (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
      (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
      (= ?CH-NAME ?TEMP3921))
(robustness = 0.9244657227306444)
21400009:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
```

```
    (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923)
    (PIERS ?PIER)
    (PIERS.GLC_CD ?PIER ?TEMP3922)
    (= ?TEMP3923 ?TEMP3922)
    (= ?TEMP3922 ?TEMP3920))
(:THEN (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
      (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
      (= ?CH-NAME ?TEMP3921))
(robustness = 0.9238880367030005)
21400010:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
      (= ?TEMP3923 ?TEMP3920))
(robustness = 0.9007692097122991)
21400011:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
      (= ?TEMP3923 ?TEMP3920))
(robustness = 0.9008421451878443)
21400012:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
      (= ?TEMP3923 ?TEMP3920))
(robustness = 0.8994024432908254)
21400013:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
      (= ?TEMP3923 ?TEMP3920))
(robustness = 0.8994024432908254)
21400014:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (PIERS ?PIER)
    (PIERS.GLC_CD ?PIER ?TEMP3922)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3922 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
      (= ?TEMP3923 ?TEMP3920))
(robustness = 0.8988585181349269)
21400015:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
    (PIERS ?PIER)
    (PIERS.GLC_CD ?PIER ?TEMP3922)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3922 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
```

```
      (= ?TEMP3923 ?TEMP3920))
(robustness = 0.8988585181349269)
21400016:
(:IF (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
      (= ?TEMP3923 ?CODE3846))
(robustness = 0.72188082919790244)
21400017:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
      (= ?TEMP3923 ?CODE3846))
(robustness = 0.8994024379998201)
21400018:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
      (= ?TEMP3923 ?CODE3846))
(robustness = 0.8994024379998201)
21400019:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (PIERS ?PIER)
    (PIERS.GLC_CD ?PIER ?TEMP3922)
    (= ?TEMP3922 ?TEMP3920)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
      (= ?TEMP3923 ?CODE3846))
(robustness = 0.8988585128439216)
21400020:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (PIERS ?PIER)
    (PIERS.GLC_CD ?PIER ?TEMP3922)
    (= ?TEMP3922 ?TEMP3920)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (WHARVES ?WHARF)
      (WHARVES.GLC_CD ?WHARF ?TEMP3923)
      (= ?TEMP3923 ?CODE3846))
(robustness = 0.8988585128439216)
21400021:
(:IF (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923))
(:THEN (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
      (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
      (= ?CODE3846 ?TEMP3923))
(robustness = 0.9263016551779618)
21400022:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH))
(:THEN (PIERS ?PIER)
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?TEMP3920))
(robustness = 0.5598601188032083)
21400023:
```

```
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45))
(:THEN (PIERS ?PIER)
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?TEMP3920))
(robustness = 0.5598601188032083)
21400024:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (PIERS ?PIER)
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?TEMP3920))
(robustness = 0.5583643246244873)
21400025:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (PIERS ?PIER)
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?TEMP3920))
(robustness = 0.5583643246244873)
21400026:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3923 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (PIERS ?PIER)
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?TEMP3920))
(robustness = 0.5577101882921831)
21400027:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
    (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3923 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (PIERS ?PIER)
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?TEMP3920))
(robustness = 0.5577101882921831)
21400028:
(:IF (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846))
(:THEN (PIERS ?PIER)
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?CODE3846))
(robustness = 0.5601925175095908)
21400029:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (PIERS ?PIER)
```

```
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?CODE3846))
(robustness = 0.5583643246244873)
21400030:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (PIERS ?PIER)
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?CODE3846))
(robustness = 0.5583643246244873)
21400031:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923)
    (= ?TEMP3923 ?TEMP3920)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (PIERS ?PIER)
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?CODE3846))
(robustness = 0.5577101882921831)
21400032:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45)
    (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
    (CHANNEL_HARBORS.LEAD_CH_NM ?CHANNEL-HARBOR#3874 ?CH-NAME)
    (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
    (WHARVES ?WHARF)
    (WHARVES.GLC_CD ?WHARF ?TEMP3923)
    (= ?TEMP3923 ?TEMP3920)
    (= ?TEMP3921 ?CH-NAME))
(:THEN (PIERS ?PIER)
      (PIERS.GLC_CD ?PIER ?TEMP3922)
      (= ?TEMP3922 ?CODE3846))
(robustness = 0.5577101882921831)
21400033:
(:IF (PIERS ?PIER)
    (PIERS.GLC_CD ?PIER ?TEMP3922))
(:THEN (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
      (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
      (= ?CODE3846 ?TEMP3922))
(robustness = 0.926370365920408)
21400034:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH))
(:THEN (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
      (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
      (= ?CODE3846 ?TEMP3920))
(robustness = 0.9251198590629485)
21400035:
(:IF (CHANNELS ?CHANNEL)
    (CHANNELS.GLC_CD ?CHANNEL ?TEMP3920)
    (CHANNELS.CH_NM ?CHANNEL ?TEMP3921)
    (CHANNELS.CH_DEPTH_FT ?CHANNEL ?CH-DEPTH)
    (>= ?CH-DEPTH 45))
(:THEN (CHANNEL_HARBORS ?CHANNEL-HARBOR#3874)
      (CHANNEL_HARBORS.GLC_CD ?CHANNEL-HARBOR#3874 ?CODE3846)
      (= ?CODE3846 ?TEMP3920))
(robustness = 0.9251198590629485)
```

Appendix D

Performance Data

This appendix provides the detailed numeric data from the experiments discussed in Chapter 5. All execution time data are in seconds. Key: "id" = query id, "w/o opt." = execution time without optimization, "w/ opt." = execution time with optimization, "savings" = percentage time saved due to the optimization, "opt" = optimization time.

Performance of Hand-coded Rules

id	w/o opt.	w/ opt.	savings	opt.
1	0.54±0.03	0.54±0.03	0.0%	0.02±0.00
2	1.04±0.01	0.71±0.05	32.0%	0.04±0.04
3	0.47±0.00	0.48±0.03	-2.0%	0.02±0.00
4	0.78±0.02	0.73±0.04	6.0%	0.05±0.04
5	0.78±0.01	0.80±0.04	-3.0%	0.08±0.04
12	0.66±0.00	0.69±0.03	-5.0%	0.03±0.03
13	0.66±0.00	0.70±0.04	-6.0%	0.04±0.04
16	0.78±0.01	0.78±0.04	0.0%	0.05±0.03
18	0.70±0.04	0.74±0.05	-6.0%	0.06±0.05
32	0.66±0.03	0.61±0.04	8.0%	0.03±0.04
200	0.67±0.02	0.69±0.02	-3.0%	0.03±0.02
201	0.87±0.02	0.77±0.03	12.0%	0.03±0.00
202	0.65±0.01	0.64±0.05	2.0%	0.05±0.05
204	0.70±0.00	0.68±0.05	3.0%	0.04±0.04
205	1.02±0.05	1.00±0.04	2.0%	0.08±0.04
206	1.03±0.03	1.01±0.04	2.0%	0.07±0.03
208	0.60±0.00	0.63±0.00	-5.0%	0.01±0.00
209	0.38±0.00	0.39±0.00	-3.0%	0.00±0.00
213	3.50±0.06	2.12±0.05	39.0%	0.08±0.02
214	2.44±0.06	1.67±0.05	31.0%	0.10±0.04
215	8.53±0.07	2.46±0.06	71.0%	0.09±0.04
216	1.12±0.02	1.19±0.04	-6.0%	0.06±0.03
217	0.87±0.03	0.94±0.05	-8.0%	0.05±0.04
218	1.01±0.04	1.06±0.05	-5.0%	0.05±0.00
221	0.76±0.00	0.77±0.03	-2.0%	0.03±0.03
224	0.46±0.00	0.47±0.05	-2.0%	0.01±0.00
225	0.45±0.00	0.47±0.01	-4.0%	0.01±0.00
226	1.08±0.01	1.11±0.06	-3.0%	0.09±0.04
227	0.99±0.02	0.95±0.05	4.0%	0.07±0.05
231	2.68±0.04	2.68±0.06	0.0%	0.06±0.02
232	0.58±0.01	0.62±0.03	-7.0%	0.03±0.00
233	0.58±0.01	0.62±0.04	-7.0%	0.03±0.00
304	0.74±0.00	0.02±0.00	97.0%	0.02±0.00
308	0.99±0.10	0.58±0.05	41.0%	0.03±0.05
309	0.58±0.04	0.02±0.03	97.0%	0.02±0.03
310	0.59±0.00	0.02±0.00	97.0%	0.02±0.00
Total	40.89	30.33		1.58
Average	1.14	0.84	25.84%	0.04
s =	1.42	0.58		0.03
Rules?	112			

Performance of Learned Rules

id	w/o opt.	w/ opt.	savings	opt.	id	w/o opt.	w/ opt.	savings	opt.
2	1.05±0.02	1.05±0.02	0.0%	0.01±0.00	5	0.79±0.01	0.75±0.03	5.0%	0.04±0.03
16	0.84±0.02	0.88±0.02	-5.0%	0.03±0.03	12	0.66±0.02	0.68±0.00	-5.0%	0.02±0.00
18	0.73±0.00	0.76±0.02	-4.0%	0.03±0.02	32	0.65±0.02	0.59±0.02	8.0%	0.02±0.00
32	0.66±0.00	0.69±0.03	-5.0%	0.02±0.00	202	0.65±0.00	0.60±0.03	8.0%	0.02±0.00
202	0.67±0.03	0.69±0.03	-3.0%	0.02±0.03	204	0.69±0.03	0.72±0.02	-4.0%	0.05±0.02
204	0.74±0.03	0.76±0.03	-3.0%	0.05±0.03	208	0.61±0.01	0.62±0.01	-2.0%	0.01±0.00
208	0.62±0.02	0.63±0.00	-2.0%	0.01±0.00	209	0.38±0.00	0.40±0.00	-5.0%	0.01±0.00
215	8.33±0.05	3.33±0.04	60.0%	0.15±0.04	215	8.39±0.05	3.55±0.14	58.0%	0.33±0.04
216	1.11±0.04	1.15±0.04	-4.0%	0.05±0.00	216	1.06±0.01	1.18±0.05	-11.0%	0.05±0.03
218	1.01±0.03	1.09±0.02	-8.0%	0.07±0.03	217	0.90±0.04	0.94±0.08	-5.0%	0.05±0.02
221	0.78±0.03	0.70±0.03	10.0%	0.02±0.03	218	1.01±0.03	1.07±0.04	-6.0%	0.05±0.00
227	0.98±0.01	1.07±0.03	-9.0%	0.06±0.03	221	0.75±0.00	0.79±0.03	-5.0%	0.03±0.03
231	2.64±0.01	2.73±0.03	-3.0%	0.05±0.02	225	0.46±0.00	0.47±0.01	-2.0%	0.01±0.00
232	0.58±0.00	0.61±0.00	-5.0%	0.03±0.00	226	1.07±0.03	1.06±0.03	1.0%	0.06±0.03
304	0.74±0.00	0.02±0.00	97.0%	0.02±0.00	227	0.98±0.00	0.93±0.03	5.0%	0.05±0.02
308	0.99±0.09	0.58±0.03	41.0%	0.03±0.03	304	0.73±0.02	0.02±0.00	97.0%	0.02±0.00
309	0.57±0.00	0.02±0.00	96.0%	0.02±0.00	308	1.00±0.09	0.56±0.02	44.0%	0.03±0.02
310	0.59±0.01	0.02±0.00	97.0%	0.02±0.00	309	0.57±0.01	0.03±0.04	95.0%	0.03±0.04
Total	23.60	16.76		0.69	Total	21.91	14.99		0.90
Average	1.31	0.93	29.00%	0.04	Average	1.15	0.79	31.60%	0.05
s =	1.81	0.84		0.03	s =	1.76	0.75		0.07
Rules?	101				Rules?	119			
3	0.47±0.01	0.48±0.01	-2.0%	0.01±0.02	1	0.66±0.02	0.62±0.02	7.0%	0.01±0.00
4	0.78±0.02	0.72±0.02	8.0%	0.03±0.00	13	0.78±0.06	0.82±0.06	-5.0%	0.04±0.05
32	0.66±0.00	0.61±0.04	8.0%	0.02±0.04	32	0.74±0.02	0.69±0.04	8.0%	0.02±0.02
201	0.86±0.00	0.91±0.03	-6.0%	0.02±0.02	200	0.83±0.01	0.75±0.01	10.0%	0.02±0.00
202	0.66±0.02	0.61±0.03	8.0%	0.02±0.03	202	0.73±0.02	0.68±0.01	7.0%	0.02±0.00
204	0.70±0.01	0.72±0.01	-3.0%	0.03±0.00	204	0.85±0.02	0.78±0.01	8.0%	0.05±0.00
205	1.02±0.03	0.97±0.03	5.0%	0.04±0.03	208	0.70±0.01	0.68±0.05	3.0%	0.01±0.00
206	1.01±0.02	0.99±0.03	2.0%	0.05±0.04	214	2.68±0.14	1.79±0.03	33.0%	0.12±0.00
208	0.62±0.00	0.62±0.00	0.0%	0.01±0.00	215	8.57±0.24	3.97±0.17	54.0%	0.37±0.03
213	3.43±0.03	2.13±0.04	38.0%	0.12±0.04	216	1.15±0.04	1.49±0.06	-30.0%	0.05±0.00
215	8.46±0.05	3.48±0.05	59.0%	0.11±0.04	218	1.24±0.09	1.34±0.06	-8.0%	0.08±0.00
216	1.09±0.01	1.18±0.04	-9.0%	0.06±0.00	221	0.85±0.04	0.91±0.02	-7.0%	0.03±0.00
218	0.97±0.03	1.05±0.04	-9.0%	0.08±0.03	224	0.48±0.01	0.54±0.08	-12.0%	0.01±0.03
221	0.75±0.01	0.69±0.03	8.0%	0.03±0.03	227	1.09±0.03	1.14±0.05	-5.0%	0.06±0.00
227	0.97±0.00	0.94±0.03	3.0%	0.04±0.03	233	0.67±0.02	0.77±0.03	-15.0%	0.04±0.00
304	0.73±0.03	0.02±0.02	97.0%	0.02±0.02	304	0.79±0.01	0.02±0.00	97.0%	0.02±0.00
308	0.99±0.10	0.57±0.01	43.0%	0.03±0.00	308	1.09±0.17	0.62±0.05	43.0%	0.03±0.02
309	0.58±0.00	0.03±0.02	95.0%	0.03±0.02	309	0.64±0.02	0.02±0.00	97.0%	0.02±0.00
310	0.59±0.00	0.03±0.00	95.0%	0.03±0.00	310	0.62±0.01	0.03±0.02	95.0%	0.03±0.02
Total	25.32	16.73		0.78	Total	25.13	17.63		1.03
Average	1.33	0.88	33.94%	0.04	Average	1.32	0.93	29.86%	0.05
s =	1.84	0.79		0.03	s =	1.82	0.87		0.08
Rules?	106				Rules?	118			

Performance of Very high Robust Rules

id	w/o opt.	w/ opt.	savings	opt.	id	w/o opt.	w/ opt.	savings	opt.
2	1.05±0.05	1.09±0.13	-4.0%	0.01±0.00	5	0.85±0.03	0.87±0.01	-2.0%	0.03±0.00
16	0.84±0.00	0.87±0.04	-4.0%	0.02±0.04	12	0.67±0.02	0.70±0.02	-5.0%	0.02±0.00
18	0.73±0.01	0.75±0.01	-3.0%	0.02±0.00	32	0.72±0.01	0.74±0.01	-3.0%	0.02±0.00
32	0.68±0.03	0.68±0.01	0.0%	0.01±0.00	202	0.72±0.06	0.71±0.01	1.0%	0.02±0.00
202	0.68±0.00	0.70±0.00	-3.0%	0.01±0.00	204	0.75±0.02	0.76±0.02	-1.0%	0.02±0.00
204	0.74±0.00	0.76±0.03	-2.0%	0.02±0.00	208	0.63±0.02	0.66±0.01	-5.0%	0.01±0.00
208	0.64±0.04	0.64±0.03	0.0%	0.01±0.00	209	0.39±0.01	0.41±0.02	-5.0%	0.00±0.00
215	8.64±0.75	7.49±0.07	13.0%	0.06±0.00	215	8.78±0.11	7.25±0.08	17.0%	0.08±0.00
216	1.13±0.05	1.15±0.05	-2.0%	0.03±0.03	216	1.14±0.23	1.25±0.17	-10.0%	0.04±0.00
218	1.03±0.03	1.06±0.01	-3.0%	0.03±0.00	217	0.92±0.05	1.37±0.06	-48.0%	0.26±0.05
221	0.78±0.00	0.80±0.01	-4.0%	0.01±0.00	218	1.06±0.03	1.11±0.08	-5.0%	0.04±0.00
227	0.99±0.05	1.05±0.04	-6.0%	0.03±0.00	221	0.83±0.05	0.87±0.04	-5.0%	0.02±0.00
231	2.75±0.35	2.83±0.02	-3.0%	0.03±0.00	225	0.52±0.02	0.52±0.02	0.0%	0.00±0.00
232	0.59±0.00	0.61±0.03	-3.0%	0.01±0.03	226	1.12±0.04	1.65±0.09	-48.0%	0.05±0.00
304	0.75±0.04	0.02±0.00	97.0%	0.02±0.00	227	1.00±0.04	1.15±0.05	-14.0%	0.05±0.00
308	1.02±0.11	0.57±0.04	43.0%	0.01±0.04	304	0.80±0.02	0.02±0.00	98.0%	0.02±0.00
309	0.58±0.01	0.01±0.00	98.0%	0.01±0.00	308	1.04±0.02	0.79±0.01	24.0%	0.02±0.00
310	0.60±0.02	0.01±0.00	98.0%	0.01±0.00	309	0.59±0.01	0.02±0.00	97.0%	0.02±0.00
Total	24.19	21.08		0.35	Total	23.13	20.84		0.73
Average	1.34	1.17	12.84%	0.02	Average	1.22	1.10	9.91%	0.04
s =	1.89	1.69		0.01	s =	1.84	1.55		0.06
Rules?	19				Rules?	33			
3	0.53±0.03	0.54±0.04	-2.0%	0.01±0.04	1	0.61±0.01	0.62±0.03	-1.0%	0.01±0.00
4	0.85±0.05	0.88±0.01	-4.0%	0.03±0.00	13	0.75±0.01	0.75±0.01	0.0%	0.02±0.00
32	0.73±0.00	0.75±0.05	-3.0%	0.02±0.00	32	0.72±0.01	0.74±0.04	-3.0%	0.01±0.04
201	0.95±0.00	0.97±0.05	-2.0%	0.02±0.00	200	0.74±0.00	0.75±0.01	-1.0%	0.01±0.00
202	0.73±0.01	0.75±0.00	-3.0%	0.02±0.00	202	0.73±0.02	0.74±0.05	-1.0%	0.02±0.00
204	0.76±0.01	0.79±0.00	-4.0%	0.03±0.00	204	0.76±0.02	0.79±0.01	-4.0%	0.03±0.00
205	1.16±0.02	1.21±0.02	-4.0%	0.04±0.00	208	0.67±0.01	0.68±0.01	-1.0%	0.01±0.00
206	1.16±0.01	1.23±0.04	-6.0%	0.04±0.00	214	2.63±0.08	2.37±0.08	10.0%	0.08±0.06
208	0.67±0.01	0.68±0.01	-1.0%	0.01±0.00	215	9.07±0.25	7.67±0.22	15.0%	0.08±0.00
213	3.67±0.06	3.44±0.07	6.0%	0.08±0.04	216	1.17±0.06	1.27±0.06	-8.0%	0.04±0.00
215	8.91±0.07	7.53±0.04	15.0%	0.08±0.00	218	1.11±0.05	1.16±0.04	-5.0%	0.04±0.04
216	1.24±0.03	1.24±0.05	0.0%	0.04±0.00	221	0.84±0.02	0.87±0.02	-4.0%	0.02±0.00
218	1.13±0.05	1.17±0.16	-4.0%	0.04±0.06	224	0.52±0.01	0.52±0.06	0.0%	0.00±0.00
221	0.83±0.01	0.88±0.04	-6.0%	0.02±0.04	227	1.10±0.04	1.13±0.05	-4.0%	0.04±0.00
227	1.14±0.03	1.18±0.04	-3.0%	0.04±0.00	233	0.61±0.08	0.64±0.05	-5.0%	0.02±0.00
304	0.85±0.03	0.02±0.00	98.0%	0.02±0.00	304	0.82±0.03	0.02±0.00	98.0%	0.02±0.00
308	1.11±0.04	0.78±0.00	30.0%	0.02±0.00	308	1.10±0.05	0.79±0.04	28.0%	0.02±0.04
309	0.61±0.01	0.02±0.00	97.0%	0.02±0.00	309	0.61±0.05	0.02±0.00	97.0%	0.02±0.00
310	0.64±0.01	0.01±0.00	98.0%	0.01±0.00	310	0.62±0.03	0.01±0.00	98.0%	0.01±0.00
Total	27.66	24.05		0.59	Total	25.16	21.53		0.50
Average	1.46	1.27	13.04%	0.03	Average	1.32	1.13	14.41%	0.03
s =	1.93	1.68		0.02	s =	1.93	1.67		0.02
Rules?	30				Rules?	32			

Performance of High Robust Rules

id	w/o opt.	w/ opt.	savings	opt.	id	w/o opt.	w/ opt.	savings	opt.
2	1.09±0.09	1.10±0.05	-1.0%	0.01±0.00	5	0.85±0.04	0.89±0.00	-5.0%	0.03±0.00
16	0.86±0.03	1.15±0.04	-33.0%	0.27±0.05	12	0.73±0.00	0.76±0.02	-4.0%	0.02±0.00
18	0.77±0.01	1.05±0.06	-36.0%	0.02±0.05	32	0.74±0.00	0.75±0.00	-1.0%	0.02±0.00
32	0.72±0.01	0.76±0.00	-6.0%	0.02±0.00	202	0.73±0.01	0.76±0.00	-4.0%	0.02±0.00
202	0.73±0.01	0.76±0.00	-4.0%	0.02±0.00	204	0.77±0.05	0.79±0.00	-2.0%	0.03±0.00
204	0.77±0.05	0.79±0.00	-2.0%	0.03±0.00	208	0.66±0.01	0.67±0.02	-2.0%	0.01±0.00
208	0.67±0.01	0.67±0.01	0.0%	0.01±0.00	209	0.42±0.01	0.41±0.00	2.0%	0.00±0.00
215	8.73±0.06	7.92±0.08	9.0%	0.32±0.04	215	8.77±0.08	7.25±0.05	17.0%	0.11±0.03
216	1.17±0.03	1.20±0.02	-3.0%	0.04±0.00	216	1.15±0.01	1.20±0.04	-4.0%	0.05±0.00
218	1.11±0.04	1.15±0.01	-4.0%	0.05±0.00	217	1.03±0.05	1.08±0.03	-5.0%	0.05±0.00
221	0.84±0.01	0.76±0.01	9.0%	0.02±0.00	218	1.11±0.07	1.16±0.16	-5.0%	0.05±0.08
227	1.10±0.01	1.17±0.03	-6.0%	0.04±0.03	221	0.83±0.01	0.86±0.05	-4.0%	0.02±0.05
231	2.89±0.03	2.89±0.01	0.0%	0.04±0.00	225	0.51±0.01	0.51±0.00	0.0%	0.00±0.00
232	0.60±0.05	0.63±0.01	-5.0%	0.02±0.00	226	1.20±0.05	1.26±0.05	-5.0%	0.06±0.05
304	0.85±0.01	0.02±0.00	98.0%	0.02±0.00	227	1.09±0.04	1.13±0.06	-5.0%	0.06±0.00
308	1.07±0.01	0.85±0.04	21.0%	0.02±0.06	304	0.80±0.01	0.02±0.00	98.0%	0.02±0.00
309	0.60±0.02	0.01±0.00	98.0%	0.01±0.00	308	1.06±0.05	0.59±0.05	44.0%	0.03±0.00
310	0.61±0.01	0.02±0.00	97.0%	0.02±0.00	309	0.59±0.00	0.03±0.00	95.0%	0.03±0.00
Total	25.16	22.87		0.98	Total	23.64	20.14		0.63
Average	1.40	1.27	9.12%	0.05	Average	1.24	1.06	14.82%	0.03
s =	1.90	1.78		0.09	s =	1.84	1.55		0.03
Rules?	35				Rules?	55			
3	0.53±0.01	0.54±0.00	-2.0%	0.01±0.00	1	0.61±0.04	0.62±0.05	-2.0%	0.01±0.05
4	0.86±0.05	0.88±0.05	-2.0%	0.03±0.00	13	0.73±0.00	0.77±0.04	-6.0%	0.03±0.00
32	0.73±0.01	0.75±0.04	-3.0%	0.02±0.04	32	0.73±0.00	0.74±0.01	-1.0%	0.02±0.00
201	0.95±0.05	0.96±0.05	-1.0%	0.02±0.05	200	0.73±0.01	0.76±0.04	-4.0%	0.02±0.04
202	0.73±0.03	0.75±0.06	-3.0%	0.02±0.04	202	0.72±0.02	0.75±0.00	-4.0%	0.02±0.00
204	0.75±0.00	0.78±0.00	-4.0%	0.03±0.00	204	0.75±0.00	0.79±0.00	-5.0%	0.03±0.00
205	1.15±0.01	1.19±0.06	-4.0%	0.04±0.00	208	0.67±0.01	0.68±0.00	-1.0%	0.01±0.00
206	1.12±0.01	1.21±0.05	-8.0%	0.05±0.00	214	2.52±0.06	2.44±0.05	3.0%	0.10±0.05
208	0.68±0.02	0.67±0.01	1.0%	0.01±0.00	215	8.99±0.06	7.40±0.09	18.0%	0.10±0.00
213	3.59±0.07	3.33±0.05	7.0%	0.10±0.04	216	1.18±0.05	1.22±0.02	-3.0%	0.04±0.00
215	8.93±0.05	7.49±0.06	16.0%	0.10±0.05	218	1.11±0.02	1.21±0.04	-9.0%	0.06±0.04
216	1.16±0.01	1.23±0.12	-6.0%	0.05±0.04	221	0.84±0.05	0.85±0.01	-1.0%	0.02±0.00
218	1.11±0.06	1.18±0.06	-6.0%	0.06±0.05	224	0.52±0.04	0.53±0.00	-2.0%	0.00±0.00
221	0.82±0.02	0.77±0.01	6.0%	0.03±0.00	227	1.10±0.01	1.17±0.06	-7.0%	0.05±0.05
227	1.11±0.03	1.17±0.04	-5.0%	0.04±0.05	233	0.61±0.01	0.67±0.01	-10.0%	0.03±0.00
304	0.82±0.01	0.02±0.00	98.0%	0.02±0.00	304	0.82±0.04	0.02±0.00	98.0%	0.02±0.00
308	1.07±0.07	0.60±0.01	45.0%	0.03±0.00	308	1.08±0.07	0.58±0.01	46.0%	0.02±0.00
309	0.60±0.01	0.03±0.00	95.0%	0.03±0.00	309	0.62±0.02	0.02±0.00	97.0%	0.02±0.00
310	0.60±0.01	0.02±0.00	97.0%	0.02±0.00	310	0.61±0.02	0.02±0.04	97.0%	0.02±0.04
Total	27.29	23.56		0.71	Total	24.92	21.23		0.62
Average	1.44	1.24	13.69%	0.04	Average	1.31	1.12	14.82%	0.03
s =	1.93	1.67		0.03	s =	1.91	1.61		0.03
Rules?	57				Rules?	54			

Performance of Low Robust Rules

id	w/o opt.	w/ opt.	savings	opt.	id	w/o opt.	w/ opt.	savings	opt.
2	1.09±0.03	1.11±0.00	-2.0%	0.02±0.00	5	0.86±0.02	0.91±0.04	-6.0%	0.04±0.00
16	0.85±0.04	0.90±0.00	-6.0%	0.03±0.00	12	0.73±0.01	0.76±0.00	-4.0%	0.03±0.00
18	0.77±0.05	0.81±0.00	-5.0%	0.03±0.00	32	0.73±0.02	0.76±0.01	-4.0%	0.02±0.00
32	0.73±0.00	0.76±0.01	-4.0%	0.02±0.00	202	0.73±0.02	0.76±0.03	-4.0%	0.02±0.04
202	0.73±0.01	0.76±0.05	-4.0%	0.02±0.00	204	0.77±0.00	0.78±0.05	-1.0%	0.04±0.00
204	0.76±0.00	0.78±0.05	-3.0%	0.04±0.05	208	0.67±0.04	0.68±0.01	-1.0%	0.01±0.00
208	0.66±0.01	0.67±0.01	-1.0%	0.01±0.00	209	0.42±0.01	0.43±0.01	-3.0%	0.00±0.00
215	8.94±0.03	7.64±0.04	15.0%	0.10±0.00	215	8.72±0.06	7.49±0.43	14.0%	0.10±0.05
216	1.17±0.02	1.22±0.04	-4.0%	0.05±0.04	216	1.17±0.01	1.20±0.01	-3.0%	0.05±0.00
218	1.10±0.01	1.15±0.05	-5.0%	0.06±0.00	217	1.03±0.05	1.10±0.08	-7.0%	0.05±0.04
221	0.84±0.01	0.77±0.05	8.0%	0.02±0.05	218	1.11±0.05	1.18±0.01	-7.0%	0.05±0.00
227	1.08±0.01	1.17±0.01	-8.0%	0.06±0.00	221	0.83±0.01	0.86±0.05	-4.0%	0.02±0.05
231	2.82±0.01	2.88±0.02	-2.0%	0.04±0.00	225	0.52±0.00	0.53±0.00	-2.0%	0.01±0.00
232	0.62±0.01	0.64±0.05	-3.0%	0.02±0.00	226	1.19±0.05	1.29±0.06	-9.0%	0.07±0.04
304	0.80±0.01	0.02±0.05	98.0%	0.02±0.05	227	1.11±0.00	1.18±0.06	-6.0%	0.06±0.00
308	1.07±0.01	0.59±0.04	45.0%	0.02±0.04	304	0.80±0.01	0.02±0.00	98.0%	0.02±0.00
309	0.61±0.05	0.02±0.00	97.0%	0.02±0.00	308	1.07±0.01	0.60±5.15	45.0%	0.03±0.00
310	0.61±0.01	0.02±0.00	97.0%	0.02±0.00	309	0.61±0.02	0.03±0.04	95.0%	0.03±0.04
Total	25.22	21.89		0.60	Total	23.69	20.56		0.67
Average	1.40	1.22	13.21%	0.03	Average	1.25	1.08	13.20%	0.04
s =	1.95	1.72		0.02	s =	1.82	1.60		0.02
Rules?	47				Rules?	64			
3	0.60±0.05	0.54±0.04	10.0%	0.01±0.04	1	0.61±0.05	0.61±0.00	0.0%	0.01±0.00
4	0.85±0.01	0.90±0.05	-6.0%	0.03±0.00	13	0.74±0.01	0.78±0.05	-6.0%	0.03±0.03
32	0.73±0.00	0.75±0.04	-3.0%	0.02±0.04	32	0.74±0.07	0.74±0.05	0.0%	0.02±0.05
201	0.94±0.01	0.97±0.00	-3.0%	0.02±0.00	200	0.75±0.01	0.76±0.04	-2.0%	0.02±0.00
202	0.74±0.05	0.76±0.01	-3.0%	0.03±0.00	202	0.73±0.00	0.75±0.00	-3.0%	0.02±0.00
204	0.75±0.00	0.80±0.05	-7.0%	0.03±0.05	204	0.77±0.05	0.78±0.01	-1.0%	0.04±0.00
205	1.13±0.05	1.21±0.05	-7.0%	0.05±0.04	208	0.67±0.02	0.67±0.01	0.0%	0.01±0.00
206	1.14±0.01	1.22±0.04	-7.0%	0.05±0.00	214	2.56±0.02	2.36±0.04	8.0%	0.09±0.03
208	0.66±0.01	0.67±0.05	-1.0%	0.01±0.05	215	8.76±0.08	7.44±0.08	15.0%	0.09±0.05
213	3.57±0.06	3.38±0.07	5.0%	0.11±0.05	216	1.18±0.01	1.22±0.04	-4.0%	0.05±0.00
215	8.87±0.10	7.42±0.07	16.0%	0.11±0.05	218	1.15±0.04	1.18±0.05	-3.0%	0.07±0.05
216	1.15±0.00	1.29±0.06	-12.0%	0.06±0.05	221	0.84±0.02	0.86±0.07	-3.0%	0.02±0.00
218	1.08±0.01	1.22±0.04	-13.0%	0.07±0.00	224	0.51±0.00	0.52±0.00	-2.0%	0.01±0.00
221	0.83±0.00	0.78±0.00	6.0%	0.03±0.00	227	1.09±0.06	1.16±0.05	-6.0%	0.06±0.00
227	1.11±0.06	1.20±0.04	-8.0%	0.05±0.00	233	0.63±0.05	0.68±0.02	-8.0%	0.03±0.00
304	0.82±0.12	0.02±0.06	98.0%	0.02±0.06	304	0.84±0.02	0.02±0.00	98.0%	0.02±0.00
308	1.11±0.04	0.60±0.02	47.0%	0.03±0.00	308	1.08±0.01	0.59±0.01	45.0%	0.03±0.00
309	0.60±0.00	0.03±0.04	95.0%	0.03±0.04	309	0.60±0.01	0.02±0.00	97.0%	0.02±0.00
310	0.63±0.01	0.03±0.00	95.0%	0.03±0.00	310	0.64±0.05	0.02±0.00	97.0%	0.02±0.00
Total	27.28	23.77		0.79	Total	24.87	21.16		0.66
Average	1.44	1.25	12.86%	0.04	Average	1.31	1.11	14.92%	0.03
s =	1.91	1.66		0.03	s =	1.86	1.62		0.03
Rules?	71				Rules?	70			

Performance of Range Rules

id	w/o opt.	w/ opt.	savings	opt.	id	w/o opt.	w/ opt.	savings	opt.
2	1.03±0.03	1.04±0.05	-1.0%	0.01±0.00	5	0.83±0.01	0.87±0.01	-5.0%	0.03±0.00
16	0.83±0.00	0.87±0.02	-5.0%	0.02±0.00	12	0.67±0.03	0.72±0.02	-6.0%	0.02±0.00
18	0.71±0.05	0.73±0.03	-3.0%	0.02±0.03	32	0.65±0.01	0.69±0.01	-5.0%	0.01±0.00
32	0.66±0.04	0.69±0.03	-5.0%	0.01±0.03	202	0.66±0.01	0.70±0.01	-6.0%	0.02±0.00
202	0.68±0.02	0.68±0.01	0.0%	0.01±0.00	204	0.72±0.01	0.90±0.05	-25.0%	0.03±0.00
204	0.72±0.01	0.89±0.04	-24.0%	0.03±0.03	208	0.61±0.00	0.64±0.01	-5.0%	0.01±0.00
208	0.64±0.02	0.64±0.03	0.0%	0.01±0.03	209	0.38±0.00	0.39±0.01	-3.0%	0.00±0.00
215	8.38±0.08	7.43±0.07	11.0%	0.08±0.00	215	8.64±0.04	7.30±0.09	16.0%	0.09±0.04
216	1.10±0.05	1.18±0.01	-7.0%	0.04±0.00	216	1.10±0.02	1.15±0.05	-5.0%	0.04±0.00
218	1.05±0.08	1.09±0.06	-4.0%	0.05±0.00	217	0.93±0.05	0.95±0.01	-2.0%	0.04±0.00
221	0.76±0.00	0.79±0.03	-4.0%	0.02±0.03	218	1.05±0.04	1.11±0.05	-6.0%	0.04±0.00
227	1.03±0.01	1.09±0.07	-6.0%	0.05±0.04	221	0.79±0.10	0.81±0.04	-3.0%	0.02±0.04
231	2.83±0.03	2.82±0.01	1.0%	0.03±0.00	225	0.47±0.00	0.47±0.01	0.0%	0.00±0.00
232	0.62±0.02	0.65±0.05	-5.0%	0.02±0.00	226	1.12±0.03	1.17±0.05	-5.0%	0.05±0.04
304	0.81±0.02	0.02±0.00	98.0%	0.02±0.00	227	1.04±0.02	1.06±0.02	-2.0%	0.05±0.00
308	1.03±0.28	0.56±0.01	45.0%	0.01±0.00	304	0.79±0.03	0.02±0.00	97.0%	0.02±0.00
309	0.60±0.01	0.01±0.00	98.0%	0.01±0.00	308	1.06±0.28	0.83±0.05	22.0%	0.02±0.00
310	0.61±0.01	0.02±0.00	97.0%	0.02±0.00	309	0.59±0.01	0.02±0.00	97.0%	0.02±0.00
Total	24.06	21.17		0.46	Total	22.72	19.81		0.53
Average	1.34	1.18	12.03%	0.03	Average	1.20	1.04	12.82%	0.03
s =	1.83	1.68		0.02	s =	1.82	1.56		0.02
Rules?	49				Rules?	56			
3	0.47±0.01	0.48±0.02	-2.0%	0.01±0.00	1	0.54±0.04	0.56±0.01	-4.0%	0.01±0.00
4	0.79±0.01	0.82±0.05	-4.0%	0.02±0.00	13	0.68±0.01	0.71±0.00	-4.0%	0.02±0.00
32	0.67±0.01	0.70±0.01	-4.0%	0.02±0.00	32	0.66±0.05	0.67±0.06	0.0%	0.01±0.00
201	0.89±0.02	0.91±0.02	-2.0%	0.02±0.00	200	0.70±0.05	0.69±0.04	1.0%	0.01±0.04
202	0.68±0.01	0.68±0.01	0.0%	0.02±0.00	202	0.66±0.01	0.67±0.01	-1.0%	0.01±0.00
204	0.71±0.01	0.74±0.04	-4.0%	0.03±0.04	204	0.73±0.05	0.90±0.04	-23.0%	0.03±0.00
205	1.05±0.05	1.10±0.08	-5.0%	0.03±0.00	208	0.63±0.01	0.63±0.00	0.0%	0.01±0.00
206	1.06±0.04	1.13±0.03	-6.0%	0.04±0.04	214	2.49±0.03	2.25±0.04	10.0%	0.08±0.00
208	0.62±0.01	0.62±0.05	0.0%	0.01±0.00	215	8.68±0.08	7.18±0.06	17.0%	0.08±0.00
213	3.58±0.06	3.18±0.09	11.0%	0.08±0.03	216	1.10±0.02	1.17±0.02	-6.0%	0.04±0.00
215	8.59±0.07	7.18±0.08	16.0%	0.08±0.03	218	0.99±0.03	1.09±0.07	-10.0%	0.05±0.00
216	1.07±0.02	1.26±0.05	-18.0%	0.05±0.03	221	0.78±0.01	0.81±0.04	-4.0%	0.02±0.04
218	1.03±0.01	1.12±0.05	-9.0%	0.05±0.04	224	0.46±0.07	0.46±0.00	0.0%	0.00±0.00
221	0.77±0.01	0.80±0.09	-4.0%	0.02±0.00	227	1.04±0.05	1.07±0.07	-3.0%	0.04±0.00
227	1.05±0.06	1.06±0.08	-1.0%	0.03±0.00	233	0.61±0.01	0.63±0.01	-3.0%	0.02±0.00
304	0.78±0.05	0.02±0.00	97.0%	0.02±0.00	304	0.78±0.02	0.02±0.00	97.0%	0.02±0.00
308	1.07±0.30	0.80±0.08	25.0%	0.02±0.00	308	1.03±0.32	0.78±0.08	24.0%	0.02±0.00
309	0.61±0.06	0.02±0.00	97.0%	0.02±0.00	309	0.60±0.05	0.01±0.04	98.0%	0.01±0.04
310	0.62±0.01	0.02±0.00	97.0%	0.02±0.00	310	0.67±0.02	0.02±0.00	97.0%	0.02±0.00
Total	26.10	22.62		0.59	Total	23.81	20.30		0.50
Average	1.37	1.19	13.31%	0.03	Average	1.25	1.07	14.74%	0.03
s =	1.87	1.60		0.02	s =	1.85	1.56		0.02
Rules?	60				Rules?	60			

Performance of Relational Rules

id	w/o opt.	w/ opt.	savings	opt.	id	w/o opt.	w/ opt.	savings	opt.
2	1.05±0.02	1.02±0.02	3.0%	0.01±0.00	5	0.80±0.02	0.81±0.01	-1.0%	0.03±0.00
16	0.86±0.01	0.87±0.03	-1.0%	0.02±0.03	12	0.65±0.01	0.70±0.04	-6.0%	0.02±0.04
18	0.71±0.05	0.75±0.04	-6.0%	0.02±0.00	32	0.65±0.07	0.67±0.01	-3.0%	0.01±0.00
32	0.67±0.01	0.68±0.03	-2.0%	0.01±0.00	202	0.65±0.01	0.69±0.06	-6.0%	0.02±0.00
202	0.67±0.01	0.69±0.02	-3.0%	0.01±0.00	204	0.68±0.01	0.72±0.03	-6.0%	0.04±0.03
204	0.71±0.01	0.75±0.03	-6.0%	0.04±0.03	208	0.61±0.02	0.62±0.01	-2.0%	0.01±0.00
208	0.61±0.06	0.64±0.00	-5.0%	0.01±0.00	209	0.39±0.04	0.38±0.00	3.0%	0.00±0.00
215	8.48±0.07	3.55±0.05	58.0%	0.12±0.04	215	8.47±0.09	3.50±0.06	59.0%	0.12±0.04
216	1.07±0.02	1.16±0.03	-9.0%	0.04±0.00	216	1.07±0.08	1.14±0.01	-6.0%	0.03±0.00
218	1.02±0.05	1.10±0.04	-7.0%	0.05±0.03	217	0.93±0.03	0.95±0.04	-2.0%	0.04±0.03
221	0.78±0.02	0.78±0.01	0.0%	0.01±0.00	218	1.02±0.01	1.11±0.05	-8.0%	0.04±0.00
227	1.00±0.01	1.06±0.03	-6.0%	0.03±0.04	221	0.76±0.01	0.78±0.02	-3.0%	0.02±0.00
231	2.78±0.01	2.86±0.04	-3.0%	0.03±0.00	225	0.46±0.05	0.46±0.04	0.0%	0.00±0.04
232	0.65±0.05	0.60±0.12	8.0%	0.02±0.00	226	1.11±0.01	1.15±0.04	-4.0%	0.04±0.03
304	0.77±0.02	0.02±0.00	97.0%	0.02±0.00	227	1.02±0.01	1.09±0.04	-7.0%	0.03±0.03
308	1.09±0.27	0.59±0.01	46.0%	0.02±0.00	304	0.87±0.05	0.02±0.00	98.0%	0.02±0.00
309	0.59±0.04	0.01±0.00	98.0%	0.01±0.00	308	1.05±0.27	0.59±0.01	44.0%	0.02±0.00
310	0.62±0.01	0.01±0.04	98.0%	0.01±0.04	309	0.58±0.01	0.02±0.00	97.0%	0.02±0.00
Total	24.12	17.11		0.48	Total	22.38	15.39		0.52
Average	1.34	0.95	29.05%	0.03	Average	1.18	0.81	31.24%	0.03
s =	1.85	0.90		0.03	s =	1.78	0.74		0.03
Rules?	52				Rules?	63			
3	0.48±0.01	0.48±0.01	0.0%	0.01±0.00	1	0.57±0.04	0.57±0.01	0.0%	0.01±0.00
4	0.80±0.01	0.79±0.03	1.0%	0.02±0.03	13	0.73±0.01	0.76±0.03	-4.0%	0.02±0.03
32	0.65±0.00	0.67±0.04	-3.0%	0.02±0.04	32	0.73±0.06	0.74±0.03	-1.0%	0.01±0.03
201	0.86±0.01	0.89±0.05	-4.0%	0.02±0.00	200	0.73±0.01	0.74±0.05	-1.0%	0.02±0.00
202	0.66±0.00	0.69±0.01	-5.0%	0.02±0.00	202	0.72±0.00	0.74±0.00	-3.0%	0.02±0.00
204	0.71±0.03	0.74±0.04	-4.0%	0.02±0.03	204	0.74±0.00	0.78±0.03	-5.0%	0.04±0.03
205	1.04±0.02	1.08±0.03	-4.0%	0.03±0.03	208	0.62±0.02	0.65±0.01	-5.0%	0.01±0.00
206	1.04±0.02	1.09±0.05	-5.0%	0.03±0.00	214	2.46±0.03	1.67±0.07	32.0%	0.10±0.00
208	0.61±0.01	0.62±0.01	-2.0%	0.01±0.00	215	8.49±0.08	3.60±0.05	58.0%	0.10±0.03
213	3.52±0.06	3.43±0.08	3.0%	0.10±0.04	216	1.13±0.07	1.19±0.08	-5.0%	0.04±0.00
215	8.58±0.05	7.49±0.05	13.0%	0.09±0.03	218	1.05±0.08	1.10±0.01	-5.0%	0.05±0.00
216	1.08±0.04	1.17±0.06	-8.0%	0.04±0.00	221	0.83±0.01	0.85±0.01	-2.0%	0.02±0.00
218	1.04±0.01	1.10±0.04	-5.0%	0.05±0.04	224	0.46±0.00	0.47±0.01	-2.0%	0.00±0.00
221	0.77±0.04	0.79±0.04	-3.0%	0.02±0.00	227	1.04±0.04	1.07±0.02	-3.0%	0.03±0.03
227	1.02±0.01	1.08±0.05	-6.0%	0.03±0.00	233	0.65±0.02	0.64±0.01	2.0%	0.02±0.00
304	0.79±0.03	0.02±0.00	97.0%	0.02±0.00	304	0.79±0.01	0.02±0.00	97.0%	0.02±0.00
308	1.02±0.29	0.56±0.01	45.0%	0.02±0.00	308	1.03±0.28	0.57±0.01	44.0%	0.02±0.00
309	0.63±0.03	0.02±0.00	97.0%	0.02±0.00	309	0.59±0.06	0.02±0.03	97.0%	0.02±0.03
310	0.60±0.04	0.01±0.00	98.0%	0.01±0.00	310	0.64±0.07	0.01±0.00	98.0%	0.01±0.00
Total	25.88	22.70		0.58	Total	23.98	16.18		0.56
Average	1.36	1.19	12.30%	0.03	Average	1.26	0.85	32.55%	0.03
s =	1.86	1.69		0.02	s =	1.80	0.78		0.03
Rules?	46				Rules?	58			

Learning Time Statistics

Trigger query id	Length of query	Number of learned rules	Primary relation	Elapsed time for learning	Database accesses
100	7	5	GEOLOC	153.23	22
200	9	5	COUNTRY_STATE	54.57	20
201	10	2	AUROIRTS	81.81	21
300	6	5	RUNWAYS	17.25	28
301	6	4	AIRPORTS	31.67	41
400	6	1	RUNWAYS	11.08	19
500	8	6	RUNWAYS	55.38	32
501	8	4	AIRPORTS	32.92	41
1200	7	7	SEAPORTS	61.33	46
1300	5	4	SEAPORTS	30.84	43
1600	12	37	PROT_BERTHS	137.08	84
1601	13	20	SEAPROTS	91.02	85
1800	10	9	AIRPORTS	53.34	53
1801	9	5	SEAPORTS	70.06	57
1803	10	6	GEOLOC	195.22	37
20000	8	4	SEAPORTS	64.47	53
20100	12	61	WHARVES	102.06	149
20101	12	3	PIERS	21.46	25
20500	7	0	RUNWAYS	5.97	9
20501	5	0	AIRPORTS	18.19	17
20503	4	2	AIRCRAFT_AIRFIELD_CHARS	44.06	70
20600	7	0	RUNWAYS	5.7	9
20601	5	0	RUNWAYS	17.99	17
20603	4	2	AIRCRAFT_AIRFIELD_CHARS	31.73	50
20800	7	0	AIRLIFT_PLANNING_FCTRS	24.14	36
20801	7	0	AIRCRAFT_GENERAL_CHARS	23.36	34
20900	3	2	AIRLIFT_PLANNING_FCTRS	21.05	35
21300	13	19	CHANNELS	74.46	84
21301	13	9	WHARVES	91.16	157
21303	13	4	PIERS	62.49	42
21400	18	36	CHANNELS	66.17	98
21401	18	36	WHARVES	112.45	156
21402	16	0	PIERS	35.94	42
21406	6	3	SHIP_CLASS	61.44	82
21700	7	0	SHIP_CLASS	123.64	155
21701	14	3	BERTHS	8.57	13
21703	11	7	PORT_BERTHS	45.55	33
21706	11	6	SEAPORTS	62.21	65
21800	6	0	SHIP_CLASS	33.07	42
21801	14	0	BERTHS	11.87	20
21802	11	0	PORT_BERTHS	88.25	85
21803	12	0	SEAPORTS	58.86	58
22400	4	1	AIRPORTS	23.77	30
22500	4	1	AIRPORTS	25.15	32
22600	8	3	RUNWAYS	14.89	26
22601	7	4	AIRPORTS	33.04	43
22602	3	0	AIRCRAFT_AIRFIELD_CHARS	38.42	60
22700	6	0	RUNWAYS	5.83	9
22701	5	0	AIRPORTS	18.21	17
22702	3	0	AIRCRAFT_AIRFIELD_CHARS	38.45	60
23100	14	7	CHANNELS	137.45	25
23101	14	16	SEAPORTS	378.15	80
23103	3	2	NOTIONAL_SHIP	51.38	70
23200	7	0	CHANNELS	21.11	17
23201	7	0	SEAPORTS	61.83	45
23203	3	2	NOTIONAL_SHIP	44.73	71
23300	7	0	CHANNELS	21.22	17
23301	7	0	SEAPORTS	62.02	45
23303	3	2	NOTIONAL_SHIP	50.16	65
Avg	8.39	6.02		57.94	50.46
s=	3.92	11.17		58.08	36.14
Max	18	61		378.15	157
Median	7	2		44.73	42
Min	3	0		5.7	9

Bibliography

[Agrawal *et al.*, 1993] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD*, pages 207–216, Washington, DC, 1993.

[Ambite *et al.*, 1995] Jose-Luis Ambite, Yigal Arens, Naveen Ashish, Chin Y. Chee, Chun-Nan Hsu, Craig A. Knoblock, Wei-Min Shen, and Sheila Tejada. The SIMS manual: Version 1.0. Technical Report ISI/TM-95-428, University of Southern California, Information Sciences Institute, 1995.

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

[Arens *et al.*, 1993] 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*, 2(2):127–159, 1993.

[Arens *et al.*, 1996] Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems, Special Issue on Intelligent Information Integration*, 1996.

[Bacchus *et al.*, 1992] Fahiem Bacchus, Adam Grove, Joseph Y. Halpern, and Daphne Koller. From statistics to beliefs. In *Proceedings of the Tenth National Conference on Artificial Intelligence(AAAI-92)*, pages 602–608, San Jose, CA, 1992.

[Bacchus *et al.*, 1993] Fahiem Bacchus, Adam Grove, Joseph Y. Halpern, and Daphne Koller. Statistical foundations for default reasoning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence(IJCAI-93)*, pages 563–569, Chambery, France, 1993.

[Bacchus *et al.*, 1994] Fahiem Bacchus, Adam Grove, Joseph Y. Halpern, and Daphne Koller. Forming beliefs about a changing world. In *Proceedings of the Twelfth National Conference on Artificial Intelligence(AAAI-94)*, pages 222–229, Seattle, WA, 1994.

[Bacchus, 1988] Fahiem Bacchus. *Representing and Reasoning with Probabilistic Knowledge*. PhD thesis, University of Alberta, Edmonton, Alta., Canada, 1988. Also available from MIT Press, 1990.

[Cestnik and Bratko, 1991] Bojan Cestnik and Ivan Bratko. On estimating probabilities in tree pruning. In *Machine Learning – EWSL-91, European Working Session on Learning*, pages 138–150. Springer-Verlag, Berlin, Germany, 1991.

[Chakravarthy *et al.*, 1990] 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.

[Chu and Chen, 1994] Wesley W. Chu and Qiming Chen. A pattern-based data and knowledge integration for intelligent query answering. *Journal of Integrated Computer-Aided Engineering*, 1(5), 1994.

[Clark and Boswell, 1991] Peter Clark and Robin Boswell. Rule induction with CN2: Some recent improvements. In *Machine Learning – EWSL-91, European Working Session on Learning*, pages 151–163. Springer-Verlag, Berlin, Germany, 1991.

[Clark and Niblett, 1989] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[Cohen, 1993] William W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence(IJCAI-93)*, Chambery, France, 1993.

[Cohen, 1995a] Paul R. Cohen. *Empirical methods for artificial intelligence*. The MIT Press, Cambridge, MA, 1995.

[Cohen, 1995b] William W. Cohen. Fast effective rule induction. In *Machine Learning, Proceedings of the 12th International Conference(ML-95)*, San Mateo, CA, 1995. Morgan Kaufmann.

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

[Cussens, 1993] James Cussens. Bayes and pesudo-Bayes estimates of conditional probabilities and their reliability. In *Machine Learning: ECML-93*, pages 136–152, Berlin, Germany, 1993. Springer-Verlag.

[DeJong and Mooney, 1986] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.

[desJardins, 1992] Marie E. desJardins. *PAGODA: A model for autonomous learning in probabilistic domains.* PhD thesis, University of California, Berkeley, Computer Science Division, 1992.

[Doorenbos *et al.*, 1992] 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(AAAI-92)*, pages 830–836, San Jose, CA, 1992.

[Etzioni, 1992] Oren Etzioni. An asymptotic analysis of speedup learning. In *Proceedings of the Ninth International Workshop on Machine Learning*, pages 129–136, San Mateo, CA, 1992. Morgan Kaufmann.

[Fayyad *et al.*, 1996] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI Press/MIT press, Cambridge, MA, 1996.

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

[Furnkranz and Widmer, 1994] Johannes Furnkranz and Gerhard Widmer. Incremental reduced error prunning. In *Machine Learning, Proceedings of the 11th International Conference(ML-94)*, San Mateo, CA, 1994. Morgan Kaufmann.

[Gil, 1992] Yolanda Gil. *Acquiring Domain Knowledge for Planning by Experimentation.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1992.

[Ginsberg, 1987] Matthew L. Ginsberg. *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann, San Mateo, CA, 1987.

[Goodman, 1946] Nielson Goodman. A query on confirmation. *Journal of Philosophy*, 43:383–385, 1946.

[Greiner and Likuski, 1989] Russel Greiner and Joseph Likuski. Incorporating redundant learned rules: A preliminary formal analysis of EBL. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 744–749, Detroit, MI, 1989.

[Halpern, 1990] Joseph Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3), December 1990.

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

[Hammer *et al.*, 1995] Joachim Hammer, Hector Garcia-Molina, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. Information translation, mediation, and mosaic-based browsing in the tsimmis system. In *Proceedings of the ACM SIGMOD Internation Conference on Management of Data*, San Jose, CA, 1995.

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

[Helmbold and Long, 1994] David P. Helmbold and Philip M. Long. Tracking drifting concepts by minimizing disagreement. *Machine Learning*, 14:27–45, 1994.

[Howson and Urbach, 1988] Colin Howson and Peter Urbach. *Scientific Reasoning: The Bayesian Approach.* Open Court, 1988.

[Hsu and Knoblock, 1993a] Chun-Nan Hsu and Craig A. Knoblock. Learning database abstractions for query reformulation. In Gregory Piatetsky-Shapiro, editor, *Proceedings of KDD-93, AAAI-93 Workshop on Knowledge Discovery in Databases*, Washington, D.C., 1993.

[Hsu and Knoblock, 1993b] Chun-Nan Hsu and Craig A. Knoblock. Reformulating query plans for multidatabase systems. In *Proceedings of the Second International Conference on Information and Knowledge Management(CIKM-93)*, Washington, D.C., 1993.

[Hsu and Knoblock, 1994] Chun-Nan Hsu and Craig A. Knoblock. Rule induction for semantic query optimization. In *Machine Learning, Proceedings of the 11th International Conference(ML-94)*, San Mateo, CA, 1994. Morgan Kaufmann.

[Hsu and Knoblock, 1995] Chun-Nan Hsu and Craig A. Knoblock. Estimating the robustness of discovered knowledge. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining(KDD-95)*, Menlo Park, CA, 1995. AAAI Press.

[Hsu and Knoblock, 1996a] Chun-Nan Hsu and Craig A. Knoblock. Discovering robust knowledge from dynamic closed-world data. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon, 1996. AAAI Press.

[Hsu and Knoblock, 1996b] Chun-Nan Hsu and Craig A. Knoblock. Using inductive learning to generate rules for semantic query optimization. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 17. AAAI Press/MIT Press, 1996.

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

[Kaelbling, 1993] Leslie. P. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning (ML-93)*, pages 167–173, Amherst, MA, 1993.

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

[Knoblock *et al.*, 1994] Craig A. Knoblock, Yigal Arens, and Chun-Nan Hsu. Cooperating agents for information retrieval. In *Proceedings of the Second International Conference on Intelligent and Cooperative Information Systems*, Toronto, Ontario, Canada, 1994.

[Laird and Rosenbloom, 1990] John E. Laird and Paul S. Rosenbloom. Integrating execution, planning, and learning in SOAR for external envirnoments. In *Proceedings of the Eighth National Conference on Artificial Intelligence(AAAI-90)*, pages 1022–1029, Boston, MA, 1990.

[Lavrač and Džeroski, 1994] Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[Levy *et al.*, 1994] Alon Y. Levy, Inderpal Singh Mumick, and Yehoshua Sagiv. Query optimization by predicate move-around. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994.

[Levy *et al.*, 1995] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval*, 5(2), 1995.

[Levy *et al.*, 1996] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query-answering algorithms for information agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence(AAAI-96)*, Portland, OR, 1996.

[Lloyd, 1987] John W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, Germany, 1987.

[MacGregor, 1990] Robert MacGregor. The evolving technology of classification-based knowledge representation systems. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, 1990.

[Mannila *et al.*, 1994] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Efficient algorithms for discovering association rules. In Usama M. Fayyad and

Ramasamy Uthurusamy, editors, *Proceedings of KDD-94, AAAI-94 Workshop on Knowledge Discovery in Databases*, Seattle, WA, 1994.

[Michalski, 1983] 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 *et al.*, 1989] 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.

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

[Mitchell *et al.*, 1983] Tom M. Mitchell, Paul E. Utgoff, and Ranan Banerji. Learning problem solving heuristics by experimentation. In *Machine Learning: An Artificial Intelligence Approach*, pages 163–190. Morgan Kaufmann, Palo Alto, CA, 1983.

[Mitchell *et al.*, 1986] Tom. M. Mitchell, Richard. M. Keller, and Smadar T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[Muggleton and Feng, 1990] Stephen Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the first conference on Algorithmic Learning Theory*, Tokyo, Japan, 1990. Morgan Kaufmann.

[Parr and Russell, 1995] Ronald Parr and Stuart J. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence(IJCAI-93)*, Montreal, Canada, 1995.

[Pawlak, 1991] Zdzislaw Pawlak. *Rough Sets: Theoretical aspects of Reasoning about Data.* Kluwer, Boston, MA, 1991.

[Pazzani and Kibler, 1992] Micheal J. Pazzani and Dennis Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9:57–94, 1992.

[Piatetsky-Shapiro and Frawley, 1991] Gregory Piatetsky-Shapiro and William J. Frawley. *Knowledge Discovery in Databases.* MIT Press, Cambridge, MA, 1991.

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

[Quinlan, 1990] J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Raedt and Bruynooghe, 1993] Luc De Raedt and Maurice Bruynooghe. A theory of clausal discovery. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence(IJCAI-93)*, Chambery, France, 1993.

[Ramsay, 1988] Allan Ramsay. *Formal Methods in Artificial Intelligence.* Cambridge University Press, Cambridge, U.K., 1988.

[Rosenbloom and Laird, 1986] Paul S. Rosenbloom and John E. Laird. Mapping explanation-based generalization onto SOAR. In *Proceedings of the Fifth National Conference on Artificial Intelligence(AAAI-86)*, Philadelphia, PA, 1986.

[Russell, 1986] Stuart J. Russell. Preliminary steps toward the automation of induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence(AAAI-86)*, Philadelphia, PA, 1986.

[Russell, 1989] Stuart J. Russell. *The Use of Knowledge in Analogy and Induction.* Morgan Kaufmann, San Mateo, CA, 1989.

[Shafer and Pearl, 1990] Glenn Shafer and Judea Pearl. *Readings in Uncertain Reasoning.* Morgan Kaufmann, San Mateo, CA, 1990.

[Shavlik and Dietterich, 1990] Jude Shavlik and Thomas A. Dietterich. *Readings in Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1990.

[Shekhar *et al.*, 1988] Shashi Shekhar, Jaideep Srivastava, and Soumitra Dutta. A formal model of trade-off between optimization and execution costs in semantic query optimization. In *Proceedings of the 14th VLDB Conference*, Los Angeles, CA, 1988.

[Shekhar *et al.*, 1993] Shashi Shekhar, Babak Hamidzadeh, Ashim Kohli, and Mark Coyle. Learning transformation rules for semantic query optimization: A data-driven approach. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):950–964, 1993.

[Shen, 1989] Wei-Min Shen. *Learning from the Environment Based on Percepts and Actions.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1989.

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

[Siegel, 1988] 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.

[Siegel, 1989] Michael D. Siegel. *Automatic Rule Derivation for Semantic Query Optimization.* PhD thesis, Department of Computer Science, Boston University, 1989.

[Silberschatz *et al.*, 1996] Avi Silberschatz, Mike Stonebraker, and Jeff Ullman. Database research: Achievements and opportunities into the 21st century. *SIGMOD Record*, March 1996.

[Sun and Yu, 1994] Wei Sun and Clement T. Yu. Semantic query optimization for tree and chain queries. *IEEE Trans. Knowledge and Data Engineering*, 6(1):136–151, 1994.

[Sutton, 1990] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning (ML-90)*, pages 216–224, Austin, TX, 1990.

[Tambe and Rosenbloom, 1993] Milind Tambe and Paul Rosenbloom. On masking effect. In *Proceedings of the Eleventh National Conference on Artificial Intelligence(AAAI-93)*, pages 526–533, Washington, D.C., 1993.

[Tambe, 1991] Milind Tambe. *Eliminating combinatorics from production match.* PhD thesis, Carnegie Mellon University, School of Computer Science, 1991.

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

[Widmer and Kubat, 1993] Gerhard Widmer and Miroslav Kubat. Effective learning in dynamic environments by explicit context tracking. In *Machine Learning: ECML-93*, Berlin, Germany, 1993. Springer-Verlag.

[Wiederhold, 1992] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, March 1992.

[Yu and Sun, 1989] Clement T. Yu and Wei Sun. Automatic knowledge acquisition and maintenance for semantic query optimization. *IEEE Trans. Knowledge and Data Engineering*, I(3):362–375, 1989.

[Ziarko, 1995] Wojciech Ziarko. The special issue on rough sets and knowledge discovery. *Computational Intelligence*, 11(2), 1995.