PLANNING BY REWRITING

by

José Luis Ambite

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)

August 1999

Abstract

Domain-independent planning is a hard combinatorial problem. Taking into account plan quality makes the task even more difficult. This thesis introduces Planning by Rewriting (PbR), a new paradigm for efficient high-quality domain-independent planning. PbR exploits declarative plan rewriting rules and efficient local search techniques to transform an easy-to-generate, but possibly suboptimal, initial plan into a high-quality plan. In addition to addressing the issues of planning efficiency and plan quality, this framework offers a new anytime planning algorithm. We have implemented this planner and applied it to several existing domains, with special emphasis on query planning in distributed and heterogeneous environments. Our PbR-based query planner is flexible, scalable, and yields a novel combination of traditional cost-based query optimization and source selection. The experimental results show that the PbR approach provides significant savings in planning effort while generating high-quality plans.

Acknowledgements

I would like to thank all of the people from whom I have learned.

My advisor, Craig Knoblock, has been a excellent source of insight and support. He has offered the best advice in the most critical moments. I have always been amazed of his ability to provide constructive criticism with great patience and good humor. I also want to thank the other members of my thesis committee: Steve Minton, Dennis McLeod, Daniel O'Leary, and Aristides Requicha. They have provided me with valuable comments. Especially, Steve contributed greatly to the clarity of the thesis and helped me to keep things simple.

I have learned much from the members of the SIMS and Ariadne groups and enjoyed working with them. Yigal Arens leadership has been an enormous asset. I hope to be as successful in generating good research, and funding for it (!), as he is. Yigal, Craig, Steve, WeiMin Shen, and Weixiong Zhang provided, along with their considerable knowledge, interesting research challenges and the encouragement to pursue our own, Craig, Ion Muslea, and Andrew Philpot collaborated in the design of the axiom compilation algorithm that was used in the application of PbR to query planning. Andrew implemented most of that algorithm and taught me some good Lisp programming. I am also benefiting from the programming skills of Maria Muslea for the ongoing port of the PbR query planner to C++. Chunnan Hsu and Naveen Ashish have been excellent office mates. They have expanded both my scientific and cultural horizons.

The Intelligent Systems Division at ISI is a remarkable research environment. Not only it is comprised by a large number of very talented individuals, but there is an exceptional atmosphere of collaboration. From the most senior professor to the newly arrived student, I always found all doors open to exchange ideas. Sunsets over the marina do not hurt either. Many people contributed to my research by offering good advice, commenting on my papers, and improving my presentations. Apart from the members of the SIMS group, Yolanda Gil, Tom Russ, Ramesh Patil, Marcelo Tallis, Gal Kaminka, and Irene Langkilde were specially helpful. Both social and scientifically, I particularly enjoyed our daily lunch discussions. Over the years these covered most of human endeavor, from knowledge representation to proportional representation, passing through infamous uses of the Mercator projection. Tom and Ramesh deserve special recognition for starting many of the topics and providing some of the most lucid contributions.

At the beginning of my graduate school, Paul Rosenbloom showed me what it means to take a scientific position rigorously and to its ultimate consequences. Jihie Kim and Ben Smith were great to have in the Soar lab.

The Ministerio of Educación y Ciencia of Spain, through a Fulbright scholarship, generously supported me during my first years of graduate school. I hope to be able to repay this moral debt.

To my family I am thankful for their support and encouragement to pursue my goals and ideas. I regret the time I have not spend with my parents, my brother Emilio, my aunt Maribel, and my grandmother Lola. Friends made this time in L.A. much more enjoyable: Sylvain, Jean-Pierre, Ana, Tracey, Kumar, Spyros, Eric, Thanks.

To Maria Luisa for her love.

This work was supported in part by a Fulbright/Ministerio of Educación y Ciencia of Spain scholarship, in part by the United States Air Force under contract number F49620-98-1-0046, in part by the National Science Foundation under grant number IRI-9313993, in part by the Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency (DARPA) under contract numbers F30602-94-C-0210, F30602-97-2-0352, F30602-97-2-0238, in part by the Integrated Media Systems Center, a NSF Engineering Research Center, and in part by a research grant from General Dynamics Information Systems. The views and conclusions contained in this thesis are the author's and should not be interpreted as representing the official opinion or policy of any of the above organizations or any person connected with them.

Contents

List Of Figures

Chapter 1

Introduction

1.1 Problem Statement

Planning is the process of generating a network of actions, a plan, that achieves a desired goal from an initial state of the world. Many problems of practical importance can be cast as planning problems. Instead of crafting an individual planner to solve each specific problem, a long line of research has focused on constructing domain-independent planning algorithms. Domain-independent planning accepts as input, not only descriptions of the initial state and the goal for each particular problem instance, but also a declarative domain specification, that is, the set of actions that change the properties on the state. Domain-independent planning makes the development of planning algorithms more efficient, allows for software and domain reuse, and facilitates the principled extension of the capabilities of the planner. Unfortunately, domain-independent planning (like most planning problems) is computationally hard [Bylander, 1994, Erol *et al.*, 1995, Bäckström and Nebel, 1995]. Given the complexity limitations, most of the previous work on domain-independent planning has focused on finding *any* solution plan without careful consideration of plan quality. Usually very simple cost functions, such as the length of the plan, have been used. However, for many practical problems plan quality is crucial. In this thesis we present a new planning paradigm, Planning by Rewriting (PbR), that addresses both planning efficiency and plan quality while maintaining the benefits of domain independence. We also show the application of PbR to several planning problems with an special emphasis on query planning in mediator systems.

1.2 Solution Approach

Two observations guided the present work. First, there are two sources of complexity in planning:

- Satisfiability: the difficulty of finding any solution to the planning problem (regardless of the quality of the solution).
- Optimization: the difficulty of finding the optimal solution according to the given cost metric.

For a given domain, each of these facets may contribute differently to the complexity of planning. In particular, there are many domains in which the satisfiability problem is easy and their complexity is dominated by the optimization problem. For example, there may be many plans that would solve the problem, so that finding one is easy (that is, in polynomial time), but the cost of each solution varies greatly, thus finding the optimal one is hard. We will refer to these domains as optimization domains. Some optimization domains of great practical interest are query optimization and manufacturing process planning.[1]

Second, planning problems have a great deal of structure. Plans are a type of graphs with strong semantics, determined both by the general properties of planning, and each particular domain specification. This structure should and can be exploited to improve the efficiency of the planning process.

Prompted by the previous observations, we developed a novel approach for efficient planning in optimization domains: Planning by Rewriting (PbR). The framework works in two phases:

1. Generate an initial solution plan. Recall, that in optimization domains this is easy. However the quality of this initial plan may be far from optimal.

2. Iteratively rewrite the current solution plan improving its quality using a set of declarative plan rewriting rules, until either an acceptable solution is found, or a resource limit is reached.

As motivation, consider two domains: distributed query processing and manufacturing process planning.[2] Distributed query processing [Yu and Chang, 1984] involves generating a plan that efficiently computes a user query from data that resides at different nodes in a network. This query plan is composed of data retrieval actions at diverse information sources and operations on this data (such as those of the relational algebra: join, selection, etc). Some systems use a general-purpose planner to solve this problem [Knoblock, 1996]. In this domain it is easy to construct an initial plan (any parse of the query suffices) and then transform it using a gradient-descent search to reduce its cost. The plan transformations exploit the commutative and associative properties of the (relational algebra) operators and facts such as that when a group of operators can be executed together at a remote information source it is generally more efficient to do so. Figure 1.1 shows some sample transformations.

In manufacturing, the problem is to find an economical plan of machining operations that implement the desired features of a design. In a feature-based approach [Nau *et al.*, 1995], it is possible to enumerate the actions involved in building a piece by analyzing its CAD model. It is more difficult to find an ordering of the operations and the setups that optimize the machining cost. However, similar to query planning, it is possible to incrementally transform a (possibly inefficient) initial plan. Often, the order of actions does not affect the design goal, only the quality of the plan,

[1] Interestingly, one of the most widely studied planning domains, the Blocks World, also has this property.

[2] The more complex domain of distributed and heterogeneous query planning is the topic of Chapter 4. A graphical example of the rewriting process in query planning appears in Figure 4.14. The process planning domain of [Minton, 1988b] is analyzed in detail in Chapter 5. Figure 5.7 shows an example of plan rewriting in this domain. The reader may want to consult those figures even if not all details can be explained at this point.

Simple-Join-Swap:

$retrieve(Q1, Source1) \bowtie [retrieve(Q2, Source2) \bowtie retrieve(Q3, Source3)] \Leftrightarrow$
$retrieve(Q2, Source2) \bowtie [retrieve(Q1, Source1) \bowtie retrieve(Q3, Source3)]$

Remote-Join-Eval:

$(retrieve(Q1, Source) \bowtie retrieve(Q2, Source)) \wedge capability(Source, join)$
$\Rightarrow retrieve(Q1 \bowtie Q2, Source)$

Figure 1.1: Transformations in Query Planning

thus many actions can commute. Also, it is important to minimize the number of setups because fixing a piece on a machine is a rather time consuming operation. Interestingly, such grouping of machining operations on a setup is analogous to evaluating a subquery at a remote information source.

As suggested by these examples, there are many problems that combine the characteristics of traditional planning satisfiability with quality optimization. For these domains there often exists natural transformations that may be used to efficiently obtain high-quality plans by iterative rewriting. Planning by Rewriting provides a domain-independent framework that allows plan transformations to be conveniently specified as declarative plan rewriting rules and facilitates the exploration of efficient (local) search techniques.

1.3 Advantages of Planning by Rewriting

There are several advantages to the planning style that PbR introduces. First, PbR is a declarative domain-independent framework. This facilitates the specification of planning domains, their evolution, and the principled extension of the planner with new capabilities. Second, the declarative rewriting rule language provides a natural and convenient mechanism to specify complex plan transformations.

Third, PbR accepts sophisticated quality measures because it operates on complete plans. In general, a complete plan is required in order to accurately assess quality. Most previous planning approaches are based on plan refinement [Kambhampati *et al.*, 1995]. This means that only partial plans are available during the planning process. A partial plan cannot offer enough information to evaluate a complex cost metric. Therefore traditional planning systems either have not addressed quality issues or have very simple quality measures such as the number of steps in the plan.

Fourth, PbR can use local search methods that have been remarkably successful in scaling to large problems [Aarts and Lenstra, 1997].[3] By using local search techniques high-quality plans can be efficiently generated. Fifth, the search occurs in the space of solution plans, which is in many cases much smaller than the space of partial plans explored by planners based on refinement search.

[3] Although the space of rewritings can be explored exhaustively, we believe that our approach is more attuned to the local search techniques typical of combinatorial optimization algorithms.

Sixth, our framework yields an anytime planning algorithm [Dean and Boddy, 1988]. The planner always has a solution to offer at any point in its computation (modulo the initial plan generation that should be fast). This is a clear advantage over traditional planning approaches, which must run to completion before producing a solution. Thus, our system allows the possibility of trading off planning effort and plan quality. For example, in query planning the quality of a plan is its execution time and it may not make sense to keep planning if the cost of the current plan is small enough, even if a cheaper one could be found. Further discussion and concrete examples of these advantages are given throughout the following chapters.

1.4 Query Planning in Mediators

Query Planning is a problem of considerable practical importance. It is central to traditional database and mediator systems. Using our Planing by Rewriting approach we have developed a novel query processing algorithm for mediator systems.

Mediators provide access, in a given application domain, to information that resides in distributed and heterogeneous sources. These systems shield the user from the complexity of accessing and combining this information. The user interacts with the mediator using a single language with agreed upon semantics for the application domain, as if it were a centralized system, without worrying about the location or languages of the sources. In order to accomplish that, mediators must provide mechanisms to resolve the semantic heterogeneity among the different sources. This is critical for selecting which information sources are relevant for a user query. A common approach consists in specifying a global model of the application domain and defining the contents of the sources in terms in this global model [Arens *et al.*, 1996, Levy *et al.*, 1996a, Kwok and Weld, 1996, Duschka and Genesereth, 1997].

Query planning in mediators involves generating a plan that efficiently computes a user query from the relevant information sources. This plan is composed of data retrieval actions at diverse information sources and data manipulation operations, such as those of the relational algebra: join, selection, union, etc. For an efficient execution, the plan has to specify both from which sources each different piece of information should be obtained and which data processing operations are going to be needed and in what order. The first problem, source selection, is characteristic of distributed and heterogeneous systems. The second problem has been the focus of traditional cost-based query optimization in databases.

Query planning in mediators is a challenging domain for planning technology. First, finding a good quality plan is computationally hard. The complexity arises both from choosing the relevant sources among a large number of available sources and from selecting the data processing operations and their ordering. Second, query planning must be done efficiently. Because in many cases the quality of a plan is the query execution time, the planner must find a plan quickly. Finally, mediators need to be flexible and extensible. A mediator should easily incorporate new sources with different capabilities and new data processing techniques. The PbR approach was developed

to meet this type of challenges. A PbR-based query planner offers a good compromise among the conflicting requirements of planning efficiency, high quality plans, flexibility, and extensibility.

We developed a PbR-based query planner for the SIMS and Ariadne mediator systems [Arens *et al.*, 1996, Knoblock and Ambite, 1997, Knoblock *et al.*, 1998] that addresses both source selection and traditional cost-based optimization. Previous approaches to query processing in mediators do not consider cost-based optimization or do it in a two-phase process. In the first phase, they find all possible translations to the user query in source terms (source selection). In the second phase, each of those source queries is sent to a cost-based optimizer. Finally, the best source query plan is selected for execution. This two-phase approach does not scale when there are many alternative information sources such as is the case in the Web. PbR performs both aspects of query planning in mediators in a single search process. By using local search techniques the combined optimization is scalable and supports an anytime behavior.

1.5 Contributions

The main contribution of this thesis is the development of Planning by Rewriting, a novel domain-independent paradigm for efficient high-quality planning. First, we define a language of declarative plan rewriting rules and present the algorithms for domain-independent plan rewriting. Our techniques for plan rewriting generalize traditional graph rewriting. Graph rewriting rules need to specify in the rule consequent the complete embedding of the replacement subplan. We introduce the novel class of partially-specified plan-rewriting rules that relax that restriction. By taking advantage of the semantics of planning, this embedding can be automatically computed. A single partially-specified rule can concisely represent a great number of fully-specified rules. These rules are also easier to write and understand than their fully-specified counterparts. Second, we adapt local search techniques, such as gradient descent and simulated annealing, to efficiently explore the space of plan rewritings. Finally, we demonstrate the usefulness of the PbR approach in several planning domains, with an special emphasis in query planning in mediator systems.

A second contribution of this thesis is a novel query planning algorithm for mediator systems that combines source selection and traditional cost-based optimization. The design and implementation of this query planner was facilitated by following the general PbR framework. Our PbR-based query planner is flexible, more scalable than previous approaches, and supports anytime behavior.

An outline of the contributions of the Planning by Rewriting framework are:

1. New paradigm for efficient high quality planning: Planning by Rewriting
   (a) Declarative Domain-independent,
      i. Flexible, Extensible, Reusable
      ii. Rewriting rules provide a natural and convenient mechanism to specify complex plan transformations

(b) Supports sophisticated quality measures

(c) Scalable:

i. Uniform framework that combines efficient local search techniques and domain-specific knowledge of useful plan transformations

ii. Explores a smaller search space (solution space) than traditional planners (partial-plan space)

(d) Anytime algorithm: allows to trade-off planning efficiency and plan quality

2. Application to several domains, with special emphasis in query access planning in mediators:

(a) Flexible and scalable query planner

(b) Novel combination of traditional cost-based query optimization and source selection

1.6 Outline

The remainder of this thesis is structured as follows. Chapter 2 provides background on planning, rewriting, and local search, some of the fields upon which PbR builds. Chapter 3 presents the basic framework of Planning by Rewriting as a domain independent approach to local search. This chapter describes in detail plan rewriting and our declarative rewriting rule language. Chapter 4 describes the challenging problem of query planning in distributed and heterogeneous domains and how a PbR-based query planner offers a novel and efficient solution. Chapter 5 describes other domains in which the PbR approach proved useful. We include experimental results for each of these planning domains. Chapter 6 reviews related work. Finally, Chapter 7 summarizes the main contributions of the thesis and discusses future work.

Chapter 2

Preliminaries: Planning, Rewriting, and Local Search

The framework of Planning by Rewriting arises as the confluence of several areas of research, namely, artificial intelligence planning algorithms, graph rewriting, and local search techniques. In this chapter we give some background on these areas and explain how they relate to PbR.

2.1 AI Planning

We assume that the reader is familiar with AI planning, but in this section we will highlight the main concepts and relate them to the PbR framework. An excellent introduction to AI Planning, including many pointers to the planning literature, can be found in [Russell and Norvig, 1995].

PbR follows the classical AI planning representation of actions that transform a state. The state is a set of ground propositions understood as a conjunctive formula. Propositions not mentioned in the initial state are considered false. In general, AI planners follow the *Closed World Assumption*, that is, if a proposition is not explicitly mentioned in the state it can be assumed to be false, similarly to the *negation as failure* semantics of logic programming. The propositions on the state are modified, asserted or negated, by the actions in the domain. The actions of a domain are specified by operator schemas.

An operator schema consists of two logical formulas: the precondition, which defines the conditions under which the operator may be applied, and the postcondition, which specifies the changes on the state effected by the operator. Propositions not mentioned in the postcondition are assumed not to change during the application of the operator. This type of representation was initially introduced in the STRIPS system [Fikes and Nilsson, 1971]. The language for the operators in PbR is the same as in Sage [Knoblock, 1995, Knoblock, 1994b], which is an extension of UCPOP [Penberthy and Weld, 1992]. The operator description language in PbR accepts arbitrary function-free first-order formulas in the preconditions of the operators, and conditional and universally quantified effects (but no disjunctive effects). In addition, the operators can specify the resources they use. Sage and PbR address unit non-consumable resources. These resources are fully acquired by an operator until the completion of its action and then released to be reused.

A sample operator schema specification for a simple Blocks World domain,[1] in the representation accepted by PbR, is given in Figure 2.1. This domain has two actions: `stack` that puts one block on top of another, and, `unstack` that places a block on the table.[2] The state is described by two predicates: `(on ?x ?y)`[3] that denotes that a block, `?x`, is on top of another block (or on the Table), `?y`, and `(clear ?x)` that denotes that a block does not have any other block on top of it.

```
(define (operator STACK)
  :parameters (?X ?Y ?Z)
  :precondition (:and (on ?X ?Z) (clear ?X) (clear ?Y)
                      (:neq ?Y ?Z) (:neq ?X ?Z) (:neq ?X ?Y)
                      (:neq ?X Table) (:neq ?Y Table))
  :effect (:and (on ?X ?Y)
                (:not (on ?X ?Z))
                (clear ?Z)
                (:not (clear ?Y))))

(define (operator UNSTACK)
  :parameters (?X ?Y)
  :precondition (:and (on ?X ?Y) (clear ?X)
                      (:neq ?X ?Y) (:neq ?X Table) (:neq ?Y Table))
  :effect (:and (on ?X Table)
                (:not (on ?X ?Y))
                (clear ?Y)))
```

Figure 2.1: Blocks World Operators

An example of a more complex operator from a process manufacturing domain is shown in Figure 2.2. This operator describes the behavior of a punch, which is a machine used to make holes in parts. The punch operation requires that there is an available clamp at the machine and that the orientation and width of the hole is appropriate for using the punch. After executing the operation the part will have the desired hole but it will also have a rough surface.[4] Note the specification on the resources slot. Declaring `(machine punch)` as a resource enforces that no other operator can use the punch concurrently. Similarly, declaring the part, `(is-object ?x)`, as a resource means that only one operation at a time can be performed on the object. Further examples of operator specifications appear in Figures 4.3, 5.1, 5.2, and 5.3.

A plan in PbR is represented by a graph, in the spirit of partial-order causal-link planners, such as UCPOP [Penberthy and Weld, 1992]. The nodes are plan steps, that is, instantiated domain operators. The edges specify a temporal ordering relation among steps imposed by causal links and ordering constraints. A causal link is a record of how a proposition is established in a plan.

[1] To illustrate the basic concepts in planning, we will use examples from a simple Blocks World domain, the reader will find a "real-world" application of planning techniques, query planning, in Chapter 4.

[2] `(stack ?x ?y ?z)` can be read as stack the block `?x` on top of block `?y` lifting from `?z`. `(unstack ?x ?y)` can be read as lift block `?x` from the top of block `?y` and put it on the Table.

[3] By convention, variables are preceded by a question mark symbol (?), as in `?x`.

[4] This operator uses an idiom combining universal quantification and negated conditional effects to specify that all previous values of the multi-valued attribute `surface-condition` are deleted.

```
(define (operator PUNCH)
  :parameters (?x ?width ?orientation)
  :resources ((machine PUNCH) (is-object ?x))
  :precondition (:and (is-object ?x)
                      (is-punchable ?x ?width ?orientation)
                      (has-clamp PUNCH))
  :effect (:and (:forall (?cond)(:when (:neq ?cond ROUGH)
                                       (:not (surface-condition ?x ?cond))))
                (surface-condition ?x ROUGH)
                (has-hole ?x ?width ?orientation)))
```

Figure 2.2: Manufacturing Operator

This record contains the proposition (sometimes also called a condition), a producer step, and a consumer step. The producer is a step in the plan that asserts the proposition, that is, the proposition is one of its effects. The consumer is a step that needs that proposition, that is, the proposition is one of its preconditions. By causality, the producer must precede the consumer.

The ordering constraints are needed to ensure that the plan is consistent. They arise from resolving operator threats and resource conflicts. An operator threat occurs when a step that negates the condition of a causal link can be ordered between the producer and the consumer steps of the causal link. To prevent this situation, which possibly makes the plan inconsistent, partial-order causal-link planners order the threatening step either before the producer (demotion) or after the consumer (promotion) by posting the appropriate ordering constraints. For the unit non-consumable resources we considered, steps requiring the same resource have to be sequentially ordered, and such a chain of ordering constraints will appear in the plan.

An example of a plan in the Blocks World using this graph representation is given in Figure 2.3. This plan transforms an initial state, consisting of two towers: C on A, A on the Table, B on D, and D on the Table; to the final state consisting of one tower: A on B, B on C, C on D, and D on the Table. The initial state is represented as step `0` with no preconditions and all the propositions of the initial state as postconditions. Similarly, the goal state is represented as a step `goal` with no postconditions and the goal formula as precondition. The plan achieves the goal by using two `unstack` steps to destroy the towers and then using three `stack` steps to build the desired tower. The causal links are shown as solid arrows and the ordering constraints as dashed arrows. The additional effects of a step that are not used in causal links, sometimes called side effects, are shown after each step pointed by thin dashed arrows. Negated propositions are preceded by $\neg$. Note the need of the ordering link between the steps 2, `stack(B C Table)`, and 3, `stack(A B Table)`. If step 3 could be ordered concurrently or before step 2, it would negate the precondition `clear(B)` of step 2, making the plan inconsistent. A similar situation occurs between steps 1 and 2 where another ordering link is introduced.

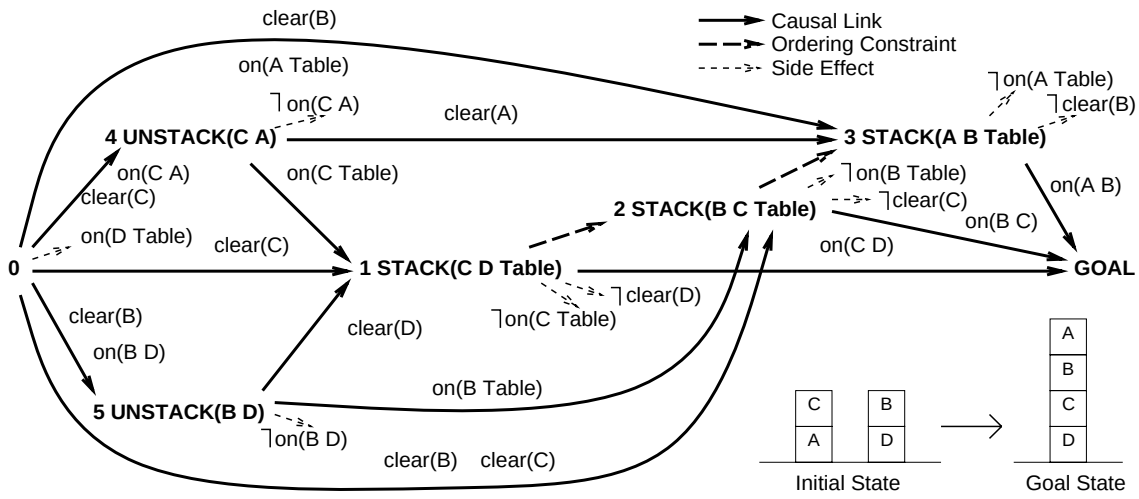


Figure 2.3: Sample Plan in the Blocks World Domain

PbR plans are always complete, consistent, and fully instantiated. All propositions in a PbR plan are ground, that is all variables are bound to constants.[5]

2.2 Rewriting

Plan rewriting in PbR is related to term and graph rewriting. Term rewriting originated in the context of equational theories and reduction to normal forms as an effective way to perform deduction — see [Avenhaus and Madlener, 1990] for a short introduction or [Baader and Nipkow, 1998] for an in-depth treatment. A rewrite system is specified as a set of rules. Each rule corresponds to a preferred direction of an equivalence theorem. The main issue in term rewriting systems is convergence, that is, if two arbitrary terms can be rewritten in a finite number of steps into a unique normal form. In PbR plans are considered "equivalent", in the term rewriting sense, if they are solutions to the same problem, although they may differ on their cost (that is they are "equivalent" with respect to "satisfiability" as introduced above). However, we are not interested in using the rewriting rules to prove that two arbitrary solution plans for a given problem are indeed equivalent, our framework uses the rewriting rules to explore the space of solution plans.

Graph rewriting, akin to term rewriting, refers to the process of replacing a subgraph of a given graph, when some conditions are satisfied, by another subgraph. Graph rewriting has found broad applications, such as very high-level programming languages, database data description and query languages, etc. See [Schürr, 1996a] for a survey. The main drawback of general graph rewriting is their complexity. Because graph matching can be reduced to (sub)graph isomorphism the problem

[5] As we will show in the following chapter, this results from the requirement that PbR starts with a complete initial plan and all rewritings being valid plans.

is NP-complete. Nevertheless, under some restrictions graph rewriting can be performed efficiently [Dorr, 1995].

Planning by Rewriting adapts general graph rewriting to the semantics of partial-order planning with a STRIPS-like operator representation. A plan rewriting rule in PbR specifies the replacement, under certain conditions, of a subplan by another subplan. However in our formalism the rule does not need to specify the completely detailed embedding of the consequent as in graph rewriting systems. All the consistent embeddings of the consequent, with the generation of edges if necessary, are automatically computed according to the semantics of partial-order planning. Our algorithm ensures that the rewritten plans always remain sound and complete.[6] The plan rewriting rules are intended to explore the space of solution plans and lead to high-quality plans.

2.3 Local Search in Combinatorial Optimization

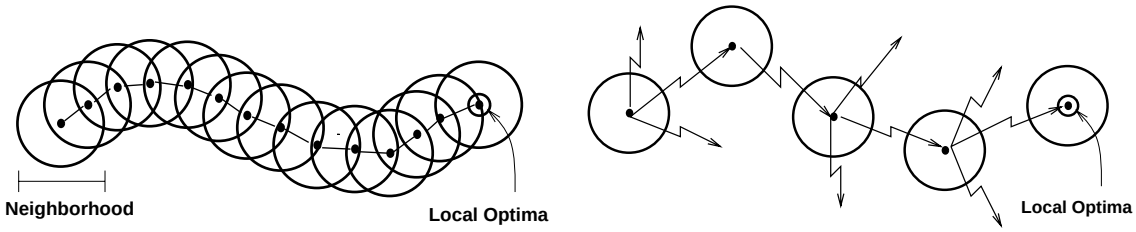
PbR is inspired by the local search techniques used in combinatorial optimization. An instance of a combinatorial optimization problem consists of a set of feasible solutions and a cost function over the solutions. The problem consists in finding a solution with the optimal cost among all feasible solutions. Generally the problems addressed are computationally intractable, thus approximation algorithms have to be used. One class of approximation algorithms that have been surprisingly successful in spite of their simplicity are local search methods [Aarts and Lenstra, 1997, Papadimitriou and Steiglitz, 1982].

Local search is based on the concept of a neighborhood. A neighborhood of a solution p is a set of solutions that are in some sense close to p, for example because they can be easily computed from p or because they share a significant amount of structure with p. The neighborhood generating function may, or may not, be able to generate the optimal solution. When the neighborhood function can generate the global optima, starting from some initial point, it is called exact.

Local search can be seen as a walk on a directed graph whose vertices are solutions points and whose arcs connect each point with all its neighbors. The neighborhood generating function determines the properties of this graph. In particular, whether it is connected or disconnected, how densely, and more importantly if it contains the global optimum (that is, whether the neighborhood is exact). In PbR the points are solution plans and the neighbors of a plan are the plans generated by the application of a set of declarative plan rewriting rules.

The basic version of local search is *iterative improvement*. Iterative improvement starts with an initial solution and searches a neighborhood of the solution for a lower cost solution. If such solution is found, it replaces the current solution and the search continues. Otherwise the algorithm returns a locally optimal solution. Figure 2.4 (a) shows a graphical depiction of basic iterative improvement. There are several variations of this basic algorithm. *First improvement* generates the neighborhood incrementally and selects the first solution of better cost than the current one. *Best improvement* generates the complete neighborhood and selects the best solution within this neighborhood.

[6] Section 3.2.2 describes the rewriting process using partially-specified rules in detail.



(a) Basic Iterative Improvement (b) Variable-Depth Search

Figure 2.4: Local Search

Basic iterative improvement obtains local optima not necessarily the global optimum. One way to improve the quality of the solution is to restart the search from several initial points and choose the best of the local optima reached from them. More advanced algorithms, such as variable depth rewriting, simulated annealing and tabu search, attempt to minimize the probability of being stuck in a low-quality local optimum.

Variable-depth search is based on applying a sequence of steps as opposed only one step at each iteration. Moreover the length of the sequence may change from iteration to iteration. In this way the system overcomes small cost increases if eventually they lead to strong cost reductions. Figure 2.4 (b) shows a graphical depiction of variable-depth search.

Simulated annealing selects the next point randomly. If a lower cost solution is chosen, it is selected. If a solution of a higher cost is chosen, it is still selected with some probability. This probability is decreased as the algorithm progresses (analogously to the temperature in the physical process of annealing). The function that governs the behavior of the acceptance probability is called cooling schedule. It can be proven that simulated annealing converges asymptotically to the optimal solution. Unfortunately such convergence requires exponential time. So, in practice, simulated annealing is used with faster cooling schedules (not guaranteed to converge to the optimal) and thus it behaves like an approximation algorithm.

Tabu search can also accept cost-increasing neighbors. The next solution is a randomly chosen legal neighbor even if its cost is worse than the current solution. A neighbor is legal if it is not in a tabu list. The dynamically updated tabu list prevents some solution points from being considered for some period of time. The intuition is that if we decide to consider a solution of a higher cost at least it should lie in an unexplored part of the space. This mechanism forces the exploration of the solution space out of local minima.

Finally, we should stress that the appeal of local search relies on its simplicity and good average-case behavior. As it could be expected, there are a number of negative worst-case results. For example, in the traveling salesman problem it is known that exact neighborhoods, that do not depend on the problem instance, must have exponential size [Weiner *et al.*, 1973, Savage *et al.*, 1976]. Moreover, an improving move in these neighborhoods cannot be found in polynomial time unless P = NP [Papadimitriou and Steiglitz, 1977]. Nevertheless, the best approximation algorithm for the traveling salesman problem is a local search algorithm [Johnson, 1990].

Chapter 3

Planning by Rewriting as Local Search

Planning by Rewriting can be viewed as a domain-independent framework for local search. PbR accepts arbitrary domain specifications, declarative plan rewriting rules that generate the neighborhood of a plan, and arbitrary (local) search methods. Therefore, assuming that a given combinatorial problem can be encoded as a planning problem, PbR can take it as input and experiment with different neighborhoods and search methods.

We will describe the main issues in Planning by Rewriting as an instantiation of the local search idea typical of combinatorial optimization algorithms:

- *Selection of an initial feasible point*: In PbR this phase consists in efficiently generating an initial solution plan.
- *Generation of a local neighborhood*: In PbR the neighborhood of a plan are the plans obtained from the application of a set of declarative plan rewriting rules.
- *Cost function to minimize*: This is the measure of plan quality that the planner is optimizing. The plan quality function can range from a simple domain independent cost metric, such as the number of steps, to more complex domain specific ones, such as the query evaluation cost or the total manufacturing time for a set of parts.
- *Selection of the next point*: In PbR, this consists in deciding which solution plan to consider next. This choice determines how the global space will be explored and has a significant impact on the efficiency of planning. A variety of local search strategies can be used in PbR, such as steepest descent, simulated annealing, etc. Which search method yields the best results may be domain or problem specific.

In the following sections we expand on these issues. First, we discuss the initial plan generation. Second, we show how the local neighborhood is generated by the rewriting rules. We present the syntax and semantics of the rules, the conditions for valid rewriting, the rewriting algorithm, a complexity analysis of plan rewriting, and a rule taxonomy. Third, we discuss the measures of plan quality. Finally, we address the selection of the next plan and the associated search techniques.

3.1 Initial Plan Generation

Fast initial plan generation is domain-specific in nature. It requires the user to specify an efficient mechanism to compute the initial solution plan. In general, generating an initial plan may be as hard as generating the optimal plan. However, the crucial intuition behind planning algorithms is that most practical problems are quasi-decomposable [Simon, 1969], that is, that the interactions among parts of the problems are limited. If interactions in a problem are pervasive, such as in the 8-puzzle, the operator-based representation and the algorithms of classical planning are of little use, they would behave as any other search based problem solver. Fortunately, many practical problems are indeed quasi-decomposable. This same intuition also suggests that finding initial plan generators for planning problems may not be as hard as it appears, because the system can solve the subproblems independently, and then combine them in the simplest way, for example, concatenating the solutions sequentially. Moreover, in many circumstances the problems may be easily transformed into a state that minimizes the interactions and solving the problem from this state is much easier. For example, in the Blocks World the state in which all blocks are on the table minimizes the interactions. It is simple to design an algorithm that solves any Blocks World problem passing through such intermediate state. Using these methods an initial plan generator may produce suboptimal initial plans but at a reasonable planning cost.

These ideas for constructing initial plan generators can be embodied in two general ways, which are both implemented in our system. The first one is to bootstrap on the results of a general purpose planning with a strong search control bias. The second one is to provide the user convenient high-level facilities in which to describe plan construction algorithms programmatically.

3.1.1 Biased Generative Planners

There are a variety of ways in which to control the search of a generic planner. Some planners accept search control rules, others accept heuristic functions, and some have built-in search control. We present examples of these techniques.

A very general way of efficiently constructing plans is to use a domain-independent generative planner that accepts search control rules. For example, Prodigy [Carbonell *et al.*, 1991], UCPOP [Penberthy and Weld, 1992] and Sage [Knoblock, 1995] are such planners. By setting the type of search and providing a strong bias by means of the search control rules, the planner can quickly generate a valid, although possibly suboptimal, initial plan. For example, in the manufacturing domain of [Minton, 1988a], analyzed in Section 5.1, depth-first search and a goal selection heuristic based on abstraction hierarchies [Knoblock, 1994a] quickly generates a feasible plan, but often the quality of this plan, which is defined as the time required to manufacture all objects, is suboptimal.

TLPlan [Bacchus and Kabanza, 1995] is an efficient forward-chaining planner that uses search control expressed in temporal logic. Because in forward chaining the complete state is available, much more refined domain control knowledge can be specified. The preferred search strategy used by TLPlan is depth-first search, so although it finds plans efficiently, the plans may be of low

quality. Note that because it is a generative planner that explores partial sequences of steps, it cannot use sophisticated quality measures.

HSP [Bonet *et al.*, 1997] is a forward search planner that performs a variation of heuristic search applied to classical AI planning. The built-in heuristic function is a relaxed version of the planning problem. It computes the number of required steps to reach the goal disregarding negated effects in the operators. Such metric can be computed efficiently. Despite its simplicity and that the heuristic is not admissible, it scales surprisingly well for many domains. Because the plans are generated according to the fixed heuristic function, the planner cannot incorporate a quality metric.

These types of planners are quite efficient in practice although produce suboptimal plans. They are excellent candidates to produce the initial plans that will be subsequently optimized by PbR.

3.1.2 Facilitating Algorithmic Plan Construction

For many domains, simple domain-dependent approximation algorithms will provide good initial plans. For example, in the query planning domain, the system can easily generate initial query evaluation plans by randomly (or greedily) parsing the given query. In the Blocks World it is also straightforward to generate a solution in linear time using the naive algorithm: put all blocks on the table and build the desired towers from the bottom up. Note that this algorithm produces plans of length no worse than twice the optimal, which makes it already a good approximation algorithm. However, the interest in the Blocks World has traditionally been on optimal solutions and this problem is known to be NP-hard [Gupta and Nau, 1992].

Our system facilitates the creation of these initial plans by freeing the user from specifying the detailed graph structure of a plan. The user only needs to specify an algorithm that produces a sequence of instantiated actions, that is, action names and the ground parameters that each action takes.[1] For example, Figure 3.1 shows the sequence of steps that the (user-defined) naive algorithm for the Blocks World domain described above would need to produce for the problem in Figure 2.3. Then, the system automatically converts this sequence of actions into a fully detailed partial-order plan using the operator specification of the domain. The resulting plan conforms to the internal data structures that PbR uses. This process includes creating nodes that are fully detailed operators with preconditions and effects, and adding edges that represent all the necessary causal links and ordering constraints. In our Blocks World example the resulting plan is that of Figure 2.3.

The algorithm that transforms the user-defined sequence of instantiated steps into a partial-order plan is presented in Figure 3.2. The algorithm first constructs the causal structure of the plan (statements 2 to 6) and then adds the necessary ordering links to avoid threats (statements 7 to 10). Note that the user only needs to specify action names and the corresponding instantiated action parameters as in Figure 3.1. Our algorithm consults the operator specification to find the

[1] The algorithm also accepts extra ordering constraints in addition to the sequence if they are available from the initial plan generator

```
1. unstack(C A)
2. unstack(B D)
3. stack(C D Table)
4. stack(B C Table)
5. stack(A B Table)
```

Figure 3.1: Sample High-level Plan in the Blocks World

preconditions and effects, instantiate them, construct the causal links, and check for operator threats. Operator threats all always resolved in favor of the ordering given by the user in the input plan. The reason is that the input plan may be overconstrained by the total order, but it is assumed valid. Therefore, by processing each step last to first, only the orderings that indeed avoid threats are included in the partial-order plan.

procedure TO2PO
Input: a valid total-order plan $(a_1, ..., a_n)$
Output: an equivalent partial-order plan
1. **for** $i := n$ to 1
2. **for** $p \in Preconditions(a_i)$
3. **choose** $k < i$ such that
4. 1. $p \in PositiveEffects(a_k) \wedge$
5. 2. $\nexists\ l$ such that $k < l < i \wedge p \in NegativeEffects(a_l)$
6. add order $a_k \prec a_i$
7. **for** $p \in NegativeEffects(a_i)$
8. **for** $j := (i \Leftrightarrow 1)$ to 1
9. **if** $p \in Preconditions(a_j)$
10. **then** add order $a_j \prec a_i$
11. **return** $((a_1, ..., a_n), \prec)$

Figure 3.2: Algorithm for Converting Total-order to Partial-order Plans

Our algorithm is an extension of the greedy algorithm presented in [Veloso *et al.*, 1990]. Our algorithm explores non-deterministically all the producers of a proposition (statement 3), as opposed to taking the latest producer in the sequence as in Veloso's algorithm.[2] That is, if our algorithm is explored exhaustively, it produces all partially-ordered causal structures consistent with the input sequence. Our generalization stems from the criticism of [Bäckström, 1994b] to the algorithm in [Veloso *et al.*, 1990] and our desire of being able to produce alternative initial plans.

The problem of transforming a sequence of steps into an least constrained plan is analyzed in [Bäckström, 1994b] under several natural definitions of optimality. Under his definitions of least-constrained plan and shortest parallel execution the problem is NP-hard. Bäckström shows that Veloso's algorithm, although polynomial, does not conform to any of these natural definitions.

[2] To implement their algorithm it is enough to replace statement 3 in Figure 3.2 with:
find max $k < i$ such that

Because our algorithm is not greedy, it does not suffer from the drawbacks pointed by Bäckström. Moreover, for our purposes we do not need these optimal initial plans. The space of partial orders will be explored during the rewriting process.

Initial plan generators that are able to provide multiple initial plans are preferable. The different initial plans are used in conjunction with multiple restart search techniques in order to escape from low-quality local minima.

3.2 Local Neighborhood Generation: Plan Rewriting Rules

The neighborhood of a solution plan is generated by the application of a set of declarative plan rewriting rules. These rules embody the domain-specific knowledge about what transformations of a solution plan are likely to result in higher-quality solutions and what part of the solution space is generated. The application a given rule may produce one or several rewritten plans or fail to produce a plan. The rewritten plans are guaranteed to be valid solutions.

In this section, first we describe the syntax and the semantics of the rules, including the conditions for valid rewriting. Second we discuss the rewriting algorithm and the complexity of plan rewriting. Finally, we present a taxonomy of plan rewriting rules.

3.2.1 Plan Rewriting Rules: Syntax and Semantics

First, we introduce the rule syntax and semantics through some examples. Then, we provide a formal description. A sample rule for the Blocks World domain introduced in Figure 2.1 is given in Figure 3.3. Intuitively, the rule says that, whenever possible, it is better to stack a block on top of another directly, rather than first moving it to the table.[3] Note that this situation occurs in plans generated by the simple algorithm that first puts all blocks on the table and then build the desired towers, such as the plan in Figure 2.3.

```
(define-rule :name avoid-move-twice
  :if (:operators ((?n1 (unstack ?b1 ?b2))
                   (?n2 (stack ?b1 ?b3 Table)))
       :links (?n1 (on ?b1 Table) ?n2)
       :constraints ((possibly-adjacent ?n1 ?n2)
                     (:neq ?b2 ?b3)))
  :replace (:operators (?n1 ?n2))
  :with (:operators (?n3 (stack ?b1 ?b3 ?b2))))
```

Figure 3.3: Blocks World Rewriting Rule

[3] The interpreted predicate `possibly-adjacent` ensures that the operators are consecutive in some linearization of the plan. Note that the link expression in the antecedent is actually redundant and could be removed form the rule specification. Unstack puts the block `?b1` on the table from where it is picked up by the stack operator, thus the causal link `(?n1 (on ?b1 Table) ?n2)` is already implied by the `:operators` and `:constraints` specification.

A rule for the manufacturing domain of [Minton, 1988b] is shown in Figure 3.4. This domain and other rewriting rules are described in detail in Chapter 5. The rule states that if a plan includes two consecutive punching operations in order to make holes in two different objects, but another machine, a drill-press, is also available, the plan quality may be improved by replacing one of the punch operations with the drill-press. In this domain the plan quality is the time to manufacture all parts. This rule helps to parallelize the plan and thus improve the plan quality.

```
(define-rule :name punch-by-drill-press
  :if (:operators ((?n1 (punch ?o1 ?width1 ?orientation1))
                   (?n2 (punch ?o2 ?width2 ?orientation2)))
       :links (?n1 ?n2)
       :constraints ((:neq ?o1 ?o2)
                     (possibly-adjacent ?n1 ?n2)))
  :replace (:operators (?n1))
  :with (:operators (?n3 (drill-press ?o1 ?width1 ?orientation1))))
```

Figure 3.4: Process Planning Rewriting Rule

The plan rewriting rule syntax is described by the BNF specification given in Figure 3.5. This BNF generates rules that follow the template shown in Figure 3.6. Essentially, the rules have three parts. The antecedent, the `:if` field, specifies a subplan to be matched. The consequent is divided in two fields: `:replace` and `:with`. The `:replace` field identifies the subplan that is going to be removed, a subset of steps and links of the antecedent. The `:with` field specifies the replacement subplan.

```
<rule> ::= (define-rule :name <name>
                        :if (<graph-spec-with-constraints>)
                        :replace (<graph-spec>)
                        :with (<graph-spec>))
<graph-spec-with-constraints> ::= <graph-spec>
                                  :constraints (<constraints>)
<graph-spec> ::= :operators (<nodes>)
                 :links (<edges>)
<nodes> ::= <node> | <node> <nodes>
<edges> ::= <edge> | <edge> <edges>
<constraints> ::= <constraint> | <constraint> <constraints>
<node> ::= (<node-var> <node-predicate> {:resource})
<edge> ::= (<node-var> <node-var>) |
           (<node-var> <edge-predicate> <node-var>) |
           (<node-var> :threat <node-var>)
<constraint> ::= <interpreted-predicate> |
                 (:neq <pred-var> <pred-var>)
<node-var> ∩ <pred-var> = ∅
 = optional,  | = alternative
```

Figure 3.5: BNF for the Rewriting Rules

```
(define-rule :name <rule-name>
  :if (:operators ((<nv> <np> {:resource}) ...)
       :links ((<nv> {<lp>|:threat} <nv>) ...)
       :constraints (<ip> ...))
  :replace (:operators (<nv> ...)
            :links ((<nv> {<lp>|:threat} <nv>) ...))
  :with (:operators ((<nv> <np> {:resource}) ...)
         :links ((<nv> {<lp>} <nv>) ...)))

 <nv> = node variable, <np> = node predicate,
 <lp> = causal link predicate, {} = optional
 <ip> = interpreted predicate,  | = alternative
```

Figure 3.6: Rewriting Rule Template

The semantics of the rules is as follows. The antecedent, the `:if` field, identifies a subplan, within the current plan, that satisfies some conditions. The antecedent has three parts. The `:operators` field specifies a set of nodes. The `:links` field specifies a set of edges. Finally, the `constraints` field specifies a set of constraints that the nodes and edges must satisfy.

The `:operators` field consists of a list of node variable and node predicate pairs. The step number of those steps in the plan that match the given node predicate would be correspondingly bound to the node variable. The node predicate can be interpreted in two ways: as an step action, or as a resource used by an step. The node specification `(?n2 (stack ?b1 ?b3 Table))` from Figure 3.3 is an example of an node predicate that denotes an step action. This node specification will collect tuples, composed of step number `?n2` and blocks `?b1` and `?b3`, of steps whose action is a `stack` of a block, `?b1`, that is on the `Table`, and it is moved on top of another block, `?b3`. This node specification applied to the plan in Figure 2.3 would result in three matches: (1 C D), (2 B C), and (3 A B), for the variables `(?n2 ?b1 ?b3)` respectively. If the optional keyword `:resource` is present, the node predicate is interpreted as one of the resources used by a plan step, as opposed to describing the step action. An example of a rule that matches against the resources of an operator is given in Figure 3.7. In this rule, the node specification `(?n1 (machine ?x) :resource)` will match all steps that use a resource of type `machine` and collect pairs of step number `?n1` and machine object `?x`.

```
(define-rule :name resource-swap
  :if (:operators ((?n1 (machine ?x) :resource)
                   (?n2 (machine ?x) :resource))
       :links ((?n1 :threat ?n2)))
  :replace (:links (?n1 ?n2))
  :with (:links (?n2 ?n1)))
```

Figure 3.7: Resource-Swap Rewriting Rule

The `:links` field consists of a list of link specifications. Our language admits link specifications of three types. The first type is specified as a pair of node variables. For example `(?n1 ?n2)` in Figure 3.4. This specification matches any temporal ordering link in the plan, regardless if it was imposed by causal links or by the resolution of threats. This only requires that node `?n1` is indeed ordered before `?n2` in the plan (not necessarily immediately before).

The second type of link specification matches causal links. Causal links are specified as triples composed of a producer step node variable, an edge predicate, and a consumer step node variable. The semantics of a causal link is that the producer step asserts in its effects the predicate, which in turn is needed in the preconditions of the consumer step. For example, the link specification `(?n1 (on ?b1 Table) ?n2)` in Figure 3.3 matches steps `?n1` that put a block `?b1` on the `Table` and steps `?n2` that subsequently pick up this block. That link specification applied to the plan in Figure 2.3 would result in the matches: (4 C 1) and (5 B 2), for the variables `(?n1 ?b1 ?n2)`.

The third type of link specification matches ordering links originating from the resolution of threats (coming either from resource conflicts or from operator conflicts). These links are selected by using the keyword `:threat` in the place of a condition. For example, the `resource-swap` rule in Figure 3.7 uses the link specification `(?n1 :threat ?n2)` to ensure that only steps that are ordered because they are involved in a threat situation are matched. This helps to identify which are the "critical" steps that do not have any other reasons (i.e. causal links) to be in such order, and therefore this rule may attempt to reorder them. This is useful when the plan quality depends on the degree of parallelism in the plan as a different ordering may help to parallelize the plan. Recall that threats can be solved either by promotion or demotion, so the reverse ordering may also produce a valid plan, which is often the case when the conflict is among resources as in the rule in Figure 3.7.

Interpreted predicates, built-in and user-defined, can be specified in the `:constraints` field. These predicates are implemented programmatically are opposed to being obtained by matching against components from the plan. The built-in predicates currently implemented are inequality[4] (`:neq`), comparison (`< <= > >=`), and arithmetic (`+ - * /`) predicates. The user can also add arbitrary predicates and their corresponding programmatic implementations. The interpreted predicates may act as filters on the previous variables or introduce new variables (and compute new values for them). For example, the user-defined predicate `possibly-adjacent` in the rule in Figure 3.3 ensures that the steps are consecutive in some linearization of the plan. For the plan in Figure 2.3 the extension of the `possibly-adjacent` predicate is: (0 4), (0 5), (4 5), (5 4), (4 1), (5 1), (1 2), (2 3), and (3 Goal).

The user can easily add interpreted predicates in our system. The user only needs to include a function definition that implements the interpreted predicate. During rule matching our algorithm passes arguments and calls such functions when appropriate. The current plan is a default first argument to the interpreted predicates to provide a context for the computation of the predicate (the user may ignore it). Figure 3.8 show an skeleton for the (Lisp) implementation of

[4] Equality is denoted by sharing variables in the rule specification

the `possibly-adjacent` and `less-than` interpreted predicates. Several examples of interpreted predicates that introduce new variables are shown in the rules in Chapter 4.

```
(defun possibly-adjacent (plan node1 node2)
       <function body>)

(defun less-than (plan n1 n2)
  (declare (ignore plan))
  (when (and (numberp n1) (numberp n2))
    (if (< n1 n2)
        '(nil) ;; true
      nil))) ;; false
```

Figure 3.8: Sample Implementation of Interpreted Predicates

The consequent is divided in two fields: `:replace` and `:with`. The `:replace` field identifies the subplan that is going to be removed from the plan, which is a subset of steps and links identified in the antecedent. If a step is removed, all the links that refer to the step are also removed. The `:with` field specifies the replacement subplan. The replacement subplan does not need to be completely specified. As we will see in Sections 3.2.2 and 3.2.3, the system generates all valid embeddings of the replacement subplan into the original plan (once the subplan in the `:replace` field has been removed). Thus, a single rule instantiation may produce several rewritten plans. For example, the `:with` field of the `avoid-move-twice` rule in Figure 3.3 only specifies the addition of a `stack` step but not how this step is embedded into the plan. The links to the rest of the steps of the plan are automatically computed during the rewriting process. The plan resulting from the application of this rule to the plan in Figure 2.3 is shown in Figure 3.9. Note how steps 4, `unstack(C A)`, and 1, `stack(C D Table)`, have been removed (along with all the edges incoming and emanating from them), and the new step 6, `stack(C D A)`, has replaced both of them. Step 6 now produces both `clear(A)`, that was previously produced by step 4, and `on(C D)`, that was previously produced by step 1.

The rewriting rules must be *safe*. Safety is a syntactic restriction on the consequents of the rules analogous to range-restriction and safety in database query languages [Abiteboul *et al.*, 1995]. Safety for our rewriting rules has two conditions. First, the node variables in the `:with` field must appear in the antecedent. Otherwise, the plan to be replaced would be undefined. Second, all variables in node and edge predicates in the `:replace` field must also be bound in the antecedent. This ensures that the operators and links of the replacement subplan are fully instantiated. The node variables of the replacement subplan are the only variables not appearing in the antecedent. Their values, the identifiers of the new steps, cannot be known in advance and are assigned automatically by the rewriting algorithm.

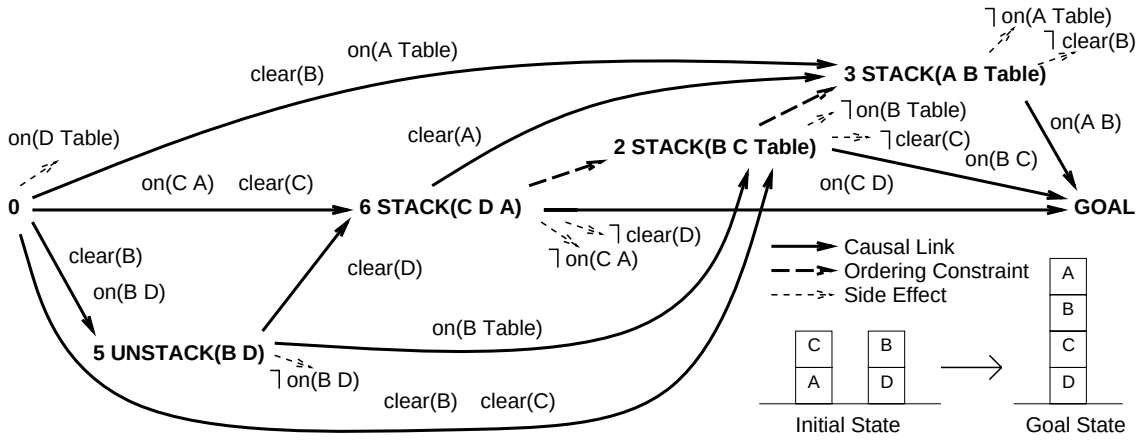


Figure 3.9: Rewritten Plan in the Blocks World Domain

3.2.2 Valid Rewriting

Before describing what the conditions for valid rewriting are, we need to introduce some definitions about (sub)plans:

Preconditions: of a (sub)plan P are the union of the preconditions of all the steps in P.

Effects: of a (sub)plan P are the union of the effects of all the steps in P.

Net Preconditions of a subplan S, embedded in a plan P, are those predicates present in causal links whose consumer is in S and whose producer is in $P \Leftrightarrow S$.

Useful Effects: of a subplan S, embedded in a plan P, are those predicates present in causal links whose producer is in S and whose consumer is in $P \Leftrightarrow S$.

Note that in the definitions of preconditions and effects of a (sub)plan the resulting set of logical formulas may not be consistent. In particular, a proposition and its negation may both appear in these sets. This does not present a problem given that planning is a form of non-monotonic reasoning. The temporal ordering in the plan ensures that a proposition and its negation cannot occur concurrently. As an example of these definitions, consider the subplan formed by steps 4 and 1 of the plan in Figure 2.3, the preconditions of this subplan are: `clear(C)`, `on(C A)`, `clear(D)`, and `on(C Table)`;[5] the net preconditions are: `clear(C)`, `on(C A)`, and `clear(D)`; the effects are: `clear(A)`, ¬`on(C A)`, `on(C Table)`, `on(C D)`, ¬`on(C Table)`, and ¬`clear(D)`; and the useful effects are: `clear(A)` and `on(C D)`.

[5] Excluding the built-in inequality predicates.

Using these definitions we can state the formal conditions for valid rewriting:

Valid Rewriting A subplan $S1$, embedded in a plan P, can be replaced by a subplan $S2$, resulting in plan P', iff:

1. UsefulEffects$(S1, P) \subseteq$ Effects$((P \Leftrightarrow S1) \cup S2)$, and
2. Preconditions$(S2) \subseteq$ Effects$((P \Leftrightarrow S1) \cup S2)$, and
3. there exists an acyclic ordering O for P' such that all causal links in P' are safe.

The first condition means that all the effects that the replaced subplan, $S1$, was producing, can still be produced once $S1$ is removed and the replacement plan, $S2$, introduced. The needed effects may come either from $S2$ or from other steps in the remainder of the plan, $P \Leftrightarrow S1$. Typically all the effects are produced by the replacement subplan. The second condition means that the preconditions of the replacement subplan, $S2$, can indeed be satisfied either by effects of the rest of the plan, $P \Leftrightarrow S1$, or internally within $S2$. The preconditions provided by $P \Leftrightarrow S1$ to $S2$ are the NetPreconditions$(S2, P')$. The remaining preconditions of steps in $S2$ must be provided by (other) steps within $S2$. These two conditions ensure that the causal links that embed $S2$ into $P \Leftrightarrow S1$ can be established. The third condition states that the new plan, P', can be made internally consistent. That is, that both the new causal links introduced by embedding $S2$ and those previously in $P \Leftrightarrow S1$ are not threatened. In order to ensure this property some additional ordering constraints may be needed.[6] These conditions are a generalization of plan merging [Foulser *et al.*, 1992].

As an example, consider how these conditions are satisfied in the plan of Figure 3.9. This plan results from applying the `avoid-move-twice` rule of Figure 3.3 to the plan of Figure 2.3. The rule antecedent identifies the subplan formed by steps 1 and 4 in Figure 2.3, which is also the subplan to be replaced. The useful-effects of this subplan are `clear(A)` and `on(C D)`. This effects are provided by the replacement subplan formed by step 6 in Figure 3.9, so our first condition is satisfied. The preconditions of the replacement subplan are `on(C A)`, `clear(C)`, and `clear(D)`. This propositions are still found in the remainder of the plan after steps 1 and 4 are removed. In particular steps `0` provides `on(C A)` and `clear(C)`, and step 5 provides `clear(D)`. Thus, the second condition is also

[6] The conditions presented here are more general than those that appear in [Ambite and Knoblock, 1997], reprinted here for convenience:

Valid Rewriting A subplan $S1$, embedded in a plan P, can be replaced by a subplan $S2$, resulting in plan P', iff there exists an ordering O, such that

1. UsefulEffects$(S1, P) \subseteq$ UsefulEffects$(S2, P')$.
2. NetPreconditions$(S2, P') \subseteq$ NetPreconditions$(S1, P)$, and
3. $P' = (P \Leftrightarrow S1) \cup S2 \cup O$ is a consistent plan, and

In particular, condition 1 requires the replacement subplan, $S2$, to produce all the effects that the replaced subplan, $S1$, was producing. Although this is generally the case, it is possible that other steps in the remainder of the plan, $P \Leftrightarrow S1$, can also provide some of the necessary effects and the rewritten plan, P', can be made complete even if $S2$ does not produce by itself all the necessary effects. Condition 2 requires that the replacement subplan takes only preconditions that the replaced plan was using. This is unnecessarily restrictive, other precondition of preceding steps can be used regardless if the replaced subplan was using them or not. The new condition 3 simply emphasizes the requirements for a consistent ordering.

satisfied. Finally, it suffices to add an ordering link between step 5 and step 6 to ensure that the plan is consistent and satisfy the third condition.

The conditions for valid rewriting are very general and admit a variety of implementations. One way of ensuring the satisfaction of these conditions, that we call the *full-specification* approach, is to rely on the user, or an automated rule defining procedure, to fully specify all steps and links involved in a rewriting, both in the antecedent and consequent of the rule, in such a way that the correctness of the rewriting is guaranteed. These kind of rules, that fully specify the embedding of the replacement subplan, are the ones typically used in graph rewriting systems. However, we can take advantage of the semantics of domain independent planning to relax the specification of the rules. In this alternative, that we call the *partial-specification* approach, only the most significant aspects of the replacement subplan are specified and the system is responsible for filling in the details. That is, both the rule antecedent and consequent may state only a subset of the conditions necessary for a valid rewriting. During the rewriting phase the system computes all the embeddings consistent with such partial specification. Moreover, the user is free to specify rules that may not necessarily be able to compute a rewriting for a plan that matches the antecedent because some necessary condition was not checked in the antecedent. This may seem undesirable, but in many cases a rule can be more naturally specified in this general form. The rule may only fail for rarely occurring plans, so that the effort in defining and matching the complete specification may not be worthwhile. In the partial-specification case, the system always ensures that the rewritten plans are valid.

As an example of these two approaches to rule specification, consider the `avoid-move-twice-fs` rewriting rule in Figure 3.10 that shows a fully specified version of the rule `avoid-move-twice` in Figure 3.3. Note that the `avoid-move-twice` rule is simpler and more natural to specify than `avoid-move-twice-fs`. But, more importantly, `avoid-move-twice-fs` is making more commitments that `avoid-move-twice`. In particular, `avoid-move-twice-fs` fixes the producer of `(clear ?b1)` for `?n11` to be `?n1` when `?n7` is also known to be a valid candidate. Moreover, there may be steps in the plan, other than the ones matched by `?n1`, `?n8`, and `?n2`, that also produce the needed propositions for `?n11`. More rules would be needed to cover these additional cases. In general the number of rules may grow exponentially as all permutations of the embeddings are enumerated. However, by using the partial-specification approach we could express the general plan transformation by only one natural rule.

In summary, the main advantage of the full-specification rules is that the rewriting can be performed quite efficiently because the embedding of the consequent is already specified. The disadvantages are that the number of rules to represent a generic plan transformation may be very large and the resulting rules quite lengthy, both of these problems may decrease the performance of the match algorithm. Also, the rule specification is error-prone if written by the user. Conversely, the main advantage of the partial-specification rules is that a single rule can represent a complex plan transformation naturally and concisely. The rule can cover a large number of plan structures even if it may occasionally fail. Also, the partial specification rules are much easier to specify and

```
(define-rule :name avoid-move-twice-fs
  :if (:operators ((?n3 (unstack ?b1 ?b2))
                   (?n9 (stack ?b1 ?b3 Table)))
       :links ((?n1 (clear ?b1) ?n3)
               (?n2 (on ?b1 ?b2) ?n3)
               (?n3 (clear ?b2) ?n4)
               (?n3 (on ?b1 Table) ?n9)
               (?n7 (clear ?b1) ?n9)
               (?n8 (clear ?b3) ?n9)
               (?n9 (on ?b1 ?b3) ?n10))
       :constraints ((possibly-adjacent ?n3 ?n9)
                     (:neq ?b2 ?b3)))
  :replace (:operators (?n1 ?n2))
  :with (:operators ((?n11 (stack ?b1 ?b3 ?b2)))
         :links ((?n1 (clear ?b1) ?n11)
                 (?n8 (clear ?b3) ?n11)
                 (?n2 (on ?b1 ?b2) ?n11)
                 (?n11 (on ?b1 ?b3) ?n10))))
```

Figure 3.10: Blocks World Rewriting Rule with Full Embedding Specified

understand by the users of the system. As shown in this section, the PbR approach provides a high degree of flexibility for defining the plan rewriting rules.

3.2.3 Rewriting Algorithm

The design of a plan rewriting algorithm depends on several parameters. First, the language of the operators. Second, the language of the rewriting rules. Third, the choice of full-specification or partial-specification rewriting rules. Finally, the need for all rewritings or one rewriting, as required by the search method. In this section we analyze these issues in turn, and describe how the rewriting algorithm accommodates these choices. In the next section, we present a complexity analysis for plan rewriting.

The *language of the operators* affects the way in which the initial and rewritten plans are constructed. Our framework supports the expressive operator definition language described in Section 2.1. We provide support for this language by using standard techniques for causal link establishment and threat checking like those in Sage [Knoblock, 1995] and UCPOP [Penberthy and Weld, 1992].

The *language of the antecedents of the rewriting rules* affects the efficiency of matching. Our system implements the conjunctive query language that was described in Section 3.2.1. However, our system could easily accommodate a more expressive query language for the rule antecedent, such as a relationally complete language (non-recursive first-order language), or a recursive language such as datalog with stratified negation, without significantly increasing the computational complexity of the approach in an important way, as we discuss in Section 3.2.4.

The choice of *fully versus partially specified rewriting rules* affects the way in which the replacement plan is embedded into the current plan. If the rule is completely specified, the embedding is already specified in the rule consequent, and the replacement subplan is simply added to the current plan. If the rule is partially specified, our algorithm computes all the valid embeddings.

The choice of *one versus all rewritings* affects both the antecedent matching and the embedding of rule consequent. The rule matches can be computed either all at the same time, as in bottom-up evaluation of logic databases, or one-at-a-time as in Prolog, depending if the search strategy requires one or all rewritings. If the rule is fully-specified only one embedding per match is possible. But, if the rule is partially-specified multiple embeddings may result from a single match. If the search strategy only requires one rewriting, it must also provide a mechanism for choosing which rule is applied, which match is computed, and which embedding is generated. Different search strategies are discussed in Section 3.4.

The design of the rewriting algorithm depends on the choices described above, which are informed by the desired search strategy. Our rewriting algorithm has a modular implementation that supports all these combinations. The main modules are the antecedent match and consequent embedding routines. The rewriting algorithm is outlined in Figure 3.11.

---

1. For each plan rewriting rule, **Match** rule antecedent (`:if` field), returning a set of candidate rule instantiations:
   (a) Heuristic optimization of the antecedent query.
   (b) Evaluate the antecedent query against the relational view of the current plan.
   (c) Collect matches one-at-a-time or set-at-a-time.
2. For each antecedent instantiation:
   (a) **Remove** the subplan specified in the `:replace` field from the current plan. Add open preconditions for the net-effects of the replaced subplan.
   (b) **Generate the embedding(s)** of the replacement subplan specified in the `:with` field. This is achieved in three steps:
      i. Add replacement subplan to the remainder of the current plan.
      ii. Find new threats.
      iii. Apply a standard partial-order causal-link planning algorithm specialized to do only step reuse in order to resolve the threats and open conditions.
   (c) Collect one or all rewritings.

Figure 3.11: Outline of the Plan Rewriting Algorithm

The matching of the antecedent of the rewriting rules is implemented straightforwardly as conjunctive query evaluation. Our implementation keeps a relational representation of the steps and links in the current plan similar to the node and link specifications of the rewriting rules. The match process consists of converting the rule antecedent to a conjunctive query with interpreted predicates, and executing this query against the relational view of the plan structures. In our

current implementation we perform a simple heuristic optimization of each antecedent query for each rule, but we do not perform multiple query optimization across all rules, or a multiple rule evaluation as in Rete [Forgy, 1982]. For example, matching the antecedent of the rule in Figure 3.3 against the plan in Figure 2.3 identifies steps 1 and 4. Considering the antecedent as query, the result is the single tuple (4 C A 1 D) for the variables `(?n1 ?b1 ?b2 ?n2 ?b3)`. Note that all the variables in the rule consequent (except for the node identifiers) are present in the antecedent, thus satisfying our rule safety restriction.

For each antecedent match, the algorithm removes the subplan identified in the `:replace` field, which is a subset of the steps and links of antecedent. If a node is identified as part of the replaced plan, all the edges incoming and emanating from this node are also removed. Continuing with our example, the plan resulting of removing steps 1 and 4 from the plan in Figure 2.3 is shown in Figure 3.12. The net-effects of the replaced plan will have to be achieved but the replacement subplan. In order to facilitate this process, these effects are recorded as open conditions in the plan structure.

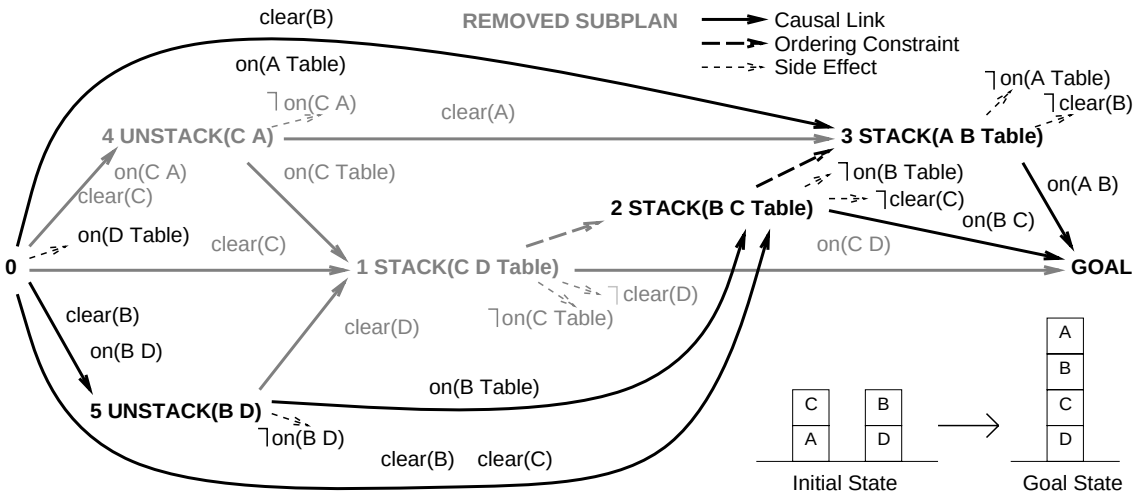


Figure 3.12: Application of a Rewriting Rule: After Removing Subplan

Finally, the algorithm embeds the replacement subplan (the `:with` field) into the remainder of the original plan. If the rule is completely specified, the algorithm simply adds the (already instantiated) steps and links of the replacement subplan to the plan, and no further work is necessary.

If the rule is partially specified, the system finds the embeddings of the replacement subplan into the remainder of the original plan. The computation of these embeddings can be seen as the application of an specialized partial-order causal-link planner that is restricted to only reuse steps. In our current implementation this is exactly the case. This allows us to support our expressive operator language and to have flexibility for computing one or all embeddings.

The embedding for the partially-specified rules is performed in three stages. First, the algorithm adds the instantiated steps and links of the replacement plan (`:with field`) into the remainder

of the original plan. Second, the algorithm computes the possible threats, both operator threats and resource conflicts, occurring in the plan. Figure 3.13 shows the state of our example after the new `stack` step (6) has been incorporated into the plan. At this point the threat situation on the `clear(C)` proposition between step 6 and 2 is identified. Finally, the system searches as in partial-order causal-link planning for all complete plans. In this search all valid ways of satisfying the open preconditions and resolving conflicts are explored, resulting in the set of plans that constitute all the valid embeddings. If only one rewriting is needed, the search can stop at the first valid plan. Otherwise, the search continues until exhausting all alternatives. In our running example, only one embedding is possible and the resulting plan is that of Figure 3.9.

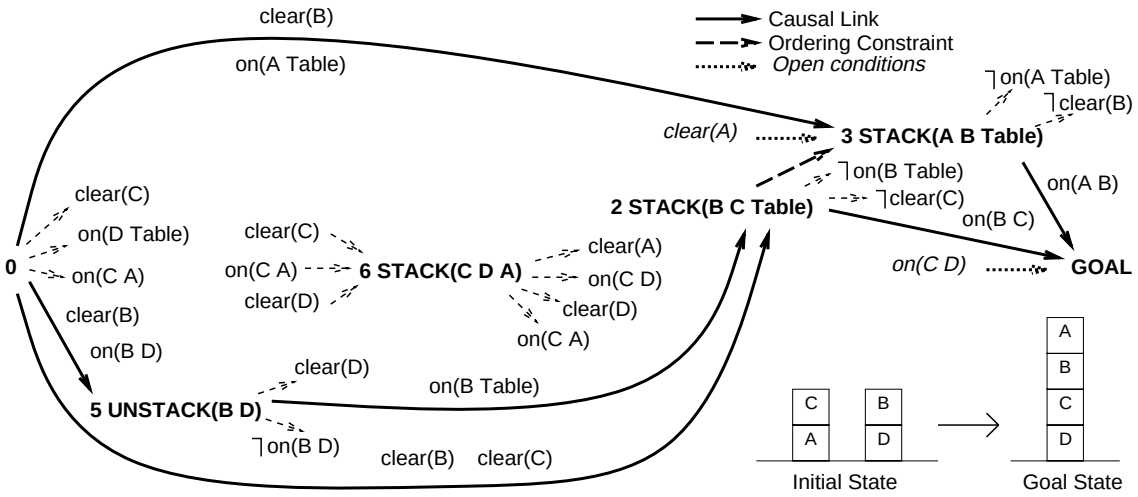


Figure 3.13: Application of a Rewriting Rule: After Adding Replacement Steps

3.2.4 Complexity of Plan Rewriting

The complexity of plan rewriting in PbR originates from two sources: matching the rule antecedent against the plan, and computing the embeddings of the replacement plan. In order to analyze the complexity of matching the plan rewriting rules we will introduce the following database-theoretic definitions of complexity [Abiteboul *et al.*, 1995]:

Data Complexity: is the complexity of evaluating a *fixed query* for variable database inputs.

Expression Complexity: is the complexity of evaluating, for a *fixed database instance*, the queries specifiable in a given query language.

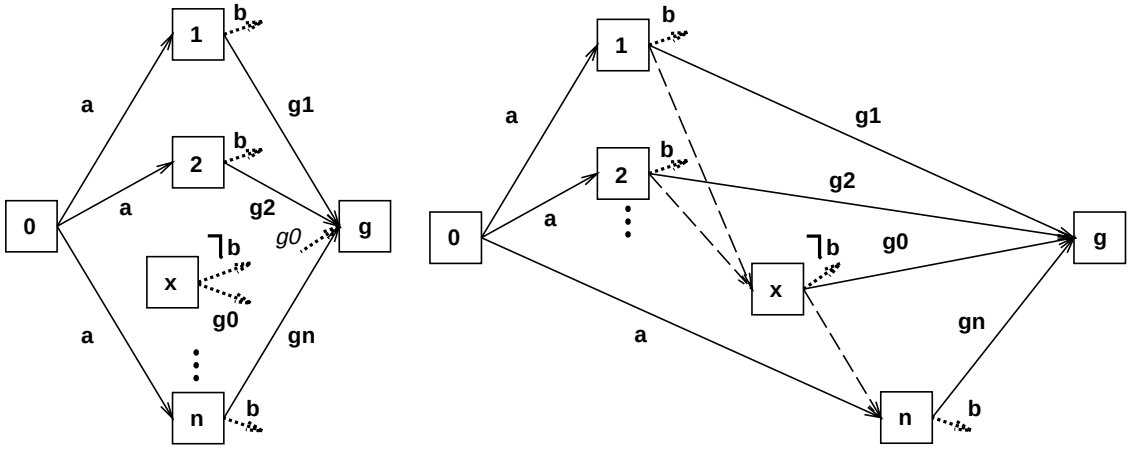
Data complexity measures the complexity with respect to the size of the database. Expression complexity measures the complexity with respect to the size of the queries (taken from a given language). In our case, the database is the steps and links of the plan and the queries are the antecedents of the plan rewriting rules.

Formally, the language of the rule antecedents that we have described consists of conjunctive queries with interpreted predicates. Unfortunately, the worst-case combined data and expression complexity of conjunctive queries is exponential [Abiteboul *et al.*, 1995]. That is, if the size of the query, the rule antecedent, and the size of the database, the plan, grow simultaneously, there is little hope of matching efficiently. Fortunately, relationally-complete languages have a data complexity contained in Logarithmic Space (which is, in turn, contained in Polynomial Time). Thus our conjunctive query language has at most this complexity. This is a very encouraging result that shows that the cost of evaluating a fixed query grows very slowly as the database size increases. For PbR this means that matching the antecedent of the rules is not strongly affected by the size of the plans. Moreover, in our experience useful rule antecedents are not very large. Besides, in practice there are many constant labels in the rule antecedent (at least, the node and edge predicate names) which help to reduce the size of the intermediate results. This result also indicates that we could extend the language of the antecedent to be relationally complete without affecting significantly the performance of the system. Another possible extension could be to datalog with stratified negation which also has polynomial time data complexity. Some graph-theoretic properties of our plans could be easily described in datalog. For example, the `possibly-adjacent` interpreted predicate of Figure 3.4 could be described declaratively as a datalog program instead of a piece of code. In summary, rule match for moderately sized rules, even for quite expressive languages and large plans, remains tractable and can be made efficient using the techniques in the production match and database literature.

The second source of complexity is computing the embeddings of the replacement plan given in the consequent of a plan rewriting rule. By the definition of full-specification rules, the embedding is completely specified in the rule itself. Given that the rules are required to be range-restricted all the variables of the consequent have been instantiated in the antecedent, thus it suffices simply to remove the undesired subplan and directly add the replacement subplan. This is linear in the size of the consequent.

For partial-specification rules, computing all the embeddings of the replacement subplan can be exponential in the size of the plan in the worst case. However this occurs only in pathological cases. For example, consider the plan in Figure 3.14 (a) in which we are going to compute the embeddings of step `x` into the remainder of the plan in order to satisfy the open precondition $g0$. Step `x` has no preconditions and has two effects ¬`b` and `g0`. Each step in the plan has proposition `b` as an effect. Therefore, the new step `x` conflicts with every step in the plan (`1` to `n`) and has to be ordered with respect to these steps. Unfortunately there are an exponential number of orderings. In effect the orderings imposed by adding the step `x` correspond to *all* the partitions of the set of steps (`1` to `n`) into two sets: one ordered before `x` and one after. Figure 3.14 (b) shows one of the possible orderings. If the subplan we were embedding contained several steps that contained similar conflicts the problem would be compounded. Even deciding if a single embedding exists can be NP-hard. For example, if we add two additional effects ¬`a` and ¬`g1` to operator `x`, there is no valid embedding. In the worst case (solving first the flaws induced by the conflicts on proposition `b`) we have to explore an exponential number of positions for step `x` in the plan, all of

which end up in failure. Nevertheless, given the quasi-decomposability of useful planning domains we expect the number of conflicts to be relatively small. Also most of the useful rewriting rules specify replacement subplans that are small compared with the plan they are embedding into. Our experience indicates that plan rewriting with partial-specification rules can be performed efficiently as shown in Chapters 4 and 5.



(a) Before embedding (b) One possible embedding

Figure 3.14: Exponential Embeddings

3.2.5 A Taxonomy of Plan Rewriting Rules

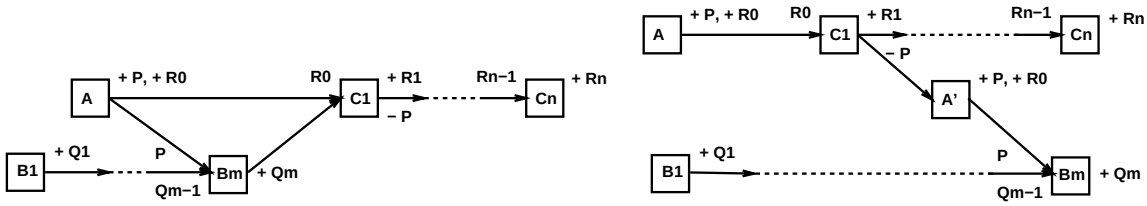
In order to guide the user in defining plan rewriting rules for a domain or to help in designing algorithms that may automatically deduce the rules from the domain specification (see Chapter 7), it is helpful to know what kinds of rules are useful. So far we have identified the following general types of transformation rules.

Reorder: These are rules based on algebraic properties of the operators, such as commutative, associative and distributive laws. For example, the commutative rule that reorders two operators that need the same resource in Figure 3.7, or the `join-swap` rule in Figure 4.10 that combines the commutative and associative properties of the relational algebra.

Collapse: These are rules that replace a subplan by a smaller subplan. For example, when several operators can be replaced by one, as in the `remote-join-eval` rule in Figure 4.8. This rule replaces two remote retrievals at the same information source and a local join operation by a single remote join operation, when the remote source has the capability of performing joins. An example of the application of this rule to a query plan is shown in Figure 4.14. Another example is the Blocks World rule in Figure 3.3 that replaces an `unstack` and a `stack` operators by an equivalent single `stack` operator.

Expand: These are rules that replace a subplan by a bigger subplan. Although this may appear counter-intuitive initially, it is easy to imagine a situation in which an expensive operator can be replaced by a set of operators that are cheaper as a whole. An interesting case is when some of these operators are already present in the plan and can be synergistically reused. We did not find this rule type in the domains analyzed so far, but [Bäckström, 1994a] presents a framework in which adding actions improves the quality of the plans. His quality metric is the plan execution time, similarly to the manufacturing domain of Section 5.1. Figure 3.15 shows an example of planning domain where adding actions improves quality (from [Bäckström, 1994a]). In this example removing the link between `Bm` and `C1` and inserting a new action `A'` shortens significantly the time to execute the plan.

Parallelize: These are rules that replace a subplan with an equivalent alternative subplan that requires fewer ordering constraints. A typical case is when there are redundant or alternative resources that the operators can use. For example, the rule `punch-by-drill-press` in Figure 3.4. Another example, is the rule that Figure 3.15 suggests that could be seen as a combination of expand and parallelize types.



(a) Low Quality Plan (b) High Quality Plan

Figure 3.15: Adding Actions Can Improve Quality

3.3 Plan Quality

In most practical planning domains the quality of the plans is crucial. This is one of the motivations for the Planning by Rewriting approach. In PbR the user defines the measure of plan quality most appropriate for the application domain. This quality metric could range from a simple domain-independent cost metric, such as the number of steps, to more complex domain-specific ones. For example, in a query planning domain the measure of plan quality may be the estimated query execution cost, or it may involve actual monetary costs if some of the information sources require payments. In the job-shop scheduling domain some simple cost functions are the schedule length (that is, the time to finish all pieces), or the sum of the times to finish each piece. A more sophisticated manufacturing domain may include a variety of concerns such as the cost, reliability, and precision of the each operator/process, the costs of resources and materials used by the operators, the utilization of the machines, etc. The reader will find more detailed examples

of quality metrics in Chapters 4 and 5. As seen in these examples, the quality function may be single objective or multi-objective. In a single objective cost function only one parameter, such as execution time, is optimized. In a multi-objective cost function, the plans are evaluated against several criteria and a combination of the objectives is the resulting cost function.

A significant advantage of PbR is that the complete plan is available to assess its quality. In generative planners the complete plan is not available until the search for a solution is completed, so usually only very simple plan quality metrics, such as the number of steps, have been used. Some work that incorporates quality concerns into generative planners is [Pérez, 1996]. Her system automatically learns quality-improving search control rules from examples of low and high quality plans. In PbR the rewriting rules can be seen as "post-facto" optimization search control. As opposed to guiding the search of a generative planner towards high-quality solutions based only in the information available in partial plans, PbR improves the quality of complete solution plans, on which quality can be better assessed. By exploiting domain-specific knowledge, conveniently expressed as plan rewriting rules, and the local search approach, PbR achieves both planning efficiency and high quality solutions.

High quality plans in some complex applications are better achieved by mixed-initiative planning. In mixed-initiative planning the user interacts with the planner in order to guide the search and define the criteria of quality. An interesting feature of complex domains is that they have multi-objective quality functions and the importance of different aspects of the quality metric may not be fully known in advance or may change over time. For example, in manufacturing the user may instruct the planner to find a solution with the cheapest monetary cost, but when the solution is returned, the user may want to find additional solution plans emphasizing other aspects such as time-to-finish, or introduce new quality criteria that may have been initially overlooked, such as avoiding the excessive use of a particular machine. An advantage of the PbR framework is that by working with complete plans the inefficiencies in the plan are more readily apparent. Sensibility analysis is also important for some applications. The user may need to test how robust are the solution plans with respect to changes in the quality criteria. Finally, note that in some domains the quality metric may be quite expensive to evaluate, thus user guidance is required. For example, in some manufacturing applications finding the quality of a plan may involve expensive geometrical computations, so the user (and the planner) needs to be careful on which plans are generated and evaluated if a high-quality plan is going to be obtained in a timely manner.

PbR is particularly well suited for mixed-initiative planning. First, the planning process is easier to understand by the user. The planner always has a complete plan to show to the user and the rewriting rules are natural perturbations of the plans which relate straightforwardly to the application domain. Second, the user can conveniently guide the system interactively. The user has the option to select which rewriting rules to apply at different stages during planning. Moreover, the user may change the quality function to emphasize different quality criteria during planning or perform sensibility analysis in order to obtain robust plans.

In summary, the Planning by Rewriting framework facilitates planning with complex quality criteria both in an automatic or mixed-initiative mode. Moreover the optimization of the plan quality is efficiently performed using local search techniques.

3.4 Selection of Next Plan: Search Strategies

Although the space of rewritings can be explored exhaustively, the Planning by Rewriting framework is better suited to the local search techniques typical of combinatorial optimization algorithms.

Many of the local search algorithms available in the literature can be adapted to work within the PbR framework. The characteristics of the planning domain, the initial plan generator, and the rewriting rules determine which algorithm performs best. First, we discuss how the initial plan generator affects the choice of local search methods. Second, we consider the impact of the rewriting rules. Third, we discuss the role of domain knowledge in the search process. Finally, we describe how several local search methods work in PbR.

An important difference between PbR and traditional combinatorial algorithms is the generation of feasible solutions. Usually, in combinatorial optimization problems it is assumed that there exists an effective procedure to generate *all* feasible solutions, for example, the permutations of an schedule. Thus, even if the local search graph is disconnected, by choosing an appropriate initial solution generator (for example random) we could fall in a component of the graph that contains the global optimum. In PbR we cannot assume such powerful initial plan generators. Even in optimization domains, in which an efficient initial plan generator is available, we may not have guarantees on the coverage of the solution space that it provides. Therefore, the optimal plan may not be reachable by the application of the rewriting rules starting from the initial plans available from the generator. Nevertheless, for many domains an initial plan generator that provides a good sample of the solution space is sufficient for multiple-restart search methods to escape from low-quality local minima and provide high-quality solutions.

The plan rewriting rules define the neighborhood function, which may be exact or not. For example, in the query planning domain we can define an exact set of rules that completely generate the space of solution plans. In other domains it may be hard to prove that we have an exact set of rules. However, in most cases a set of rules that provide access to low-cost local optima suffice.

Both the limitations on initial plan generation and the plan rewriting rules affect the possibility of theoretically reaching the global optimum. This is not surprising as it can be proven that many problems, regardless of whether they are cast as planning or in other formalisms, do not have converging local search algorithms. Nevertheless, in practice, good local optima can still be obtained for many domains.

PbR strives for a balance between domain knowledge and general local search techniques. PbR does not rely only in blindly applying local search to a problem, domain knowledge also takes an important role in structuring and directing the search process towards good solutions. Domain

knowledge is incorporated in PbR in two ways. First, PbR keeps the operator-based representation of a plan as a modular way of encoding planning knowledge. A STRIPS-like operator groups the propositions related to one action. In PbR, operators are treated as units, as opposed to approaches that translate a planning problem to another representation and apply randomized search with the new representation. For example, planning can be cast as a propositional satisfiability problem [Kautz and Selman, 1996], but there it is not clear how to exploit the connections among the propositions of an operator. We believe that some control of the search is lost during such translations and the knowledge engineering performed during the planning domain design in PbR helps in directing the search. Second, our plan rewriting rules encode useful domain knowledge. The rules can range from very general rewritings to highly domain or problem specific. So what rules are present and how the rules are expressed can be used to guide the search within a domain or even for each particular problem instance.

Many local search methods can be applied straightforwardly to PbR. In the remainder of the section we describe some of them and point to some of the adaptations needed to work within PbR. Concrete examples of these techniques applied to several domains appear in the following chapters.

First improvement generates the rewritings incrementally and selects the first plan of better cost than the current one. In order to implement this method efficiently we can use a tuple-at-a-time evaluation of the rule antecedent, similarly to the behavior of Prolog. Then, for that rule instantiation, generate one embedding, test the cost of the resulting plan, and if it is not better that the current plan, repeat. Note that we have the choice of generating another embedding of the same rule instantiation, generate another instantiation of the same rule, or generate a match of another rule.

Best improvement generates the complete set of rewritten plans and selects the best. This method requires computing all matches and all embeddings for each match. All the matches can be obtained by evaluating the rule antecedent as a traditional set-at-a-time database query. Despite that it may seem expensive to generate all matches for the rewriting rules, there are powerful database and production match techniques to optimize this evaluation.

Simulated annealing selects the next plan randomly. If a higher quality plan is chosen, it is selected. If a lower quality plan is chosen, it is still selected with some probability. This probability is decreased as planning progresses according to a cooling schedule. The requirements for its implementation are very similar to the first improvement method. The cooling schedule is another degree of freedom that the user may set to fine tune PbR for a particular domain.

Variable-depth rewriting is based on applying a sequence of rewritings atomically as opposed to only one rewriting at each iteration. This allows the planner to overcome initial cost increases that eventually would lead to strong cost reductions. This method is particularly interesting for PbR. Often it is easier to understand, specify, and apply a sequence of simple rules than a complex transformation. In some domains, such as query planning, sequences of rewritings are very natural. For example, a sequence of `Join-Swap` transformations may put two retrieve operators on the same database together in the query tree and then `Remote-Join-Eval` would collapse the explicit join

operator and the two retrieves into a single retrieval of a remote join. The complex transformation is more naturally expressed as a composition of the two simpler rules.

Tabu search is another method that facilitates escaping from local optima by allowing moves to lower-quality neighbors. However only those neighbors not in a tabu list can be selected. Often, the tabu list consists of the solutions recently considered. The problem of applying tabu search to PbR is comparing the rewritten plans to those in the tabu list. As plans are graphs this can be quite expensive. An interesting trick applicable when the plan cost is a real number is to differentiate solutions based on their cost as opposed to comparing the whole plans. Although less precise than comparing plans, this provides the desired behavior of exploring the space without the complexity of plan comparison.

In Planning by Rewriting the choice of the initial plan generator, the rewriting rules, and the search methods is intertwined. Once the initial plan generator is fixed, it determines the shape of the plans that would have to be modified by the rewriting rules, then according to this neighborhood, the most appropriate search mechanism can be chosen. Nevertheless, the design of PbR is modular. Different initial plan generators, sets of rewriting rules, and search strategies can be readily interchanged.

Chapter 4

Planning by Rewriting for Query Planning

In this chapter we present the application of the Planning by Rewriting (PbR) framework to query planning in mediators. Mediator systems integrate information from distributed and heterogeneous sources. These systems are becoming increasingly important in a world of interconnected information. The problem of query planning constitutes an excellent testbed for planning technology. Query planning is a practically important problem, planning efficiency and plan quality are critical, flexibility and extensibility are desirable, and there exists an interesting interplay between planning and execution.

Planning by Rewriting is able to address many of the challenges of query planning in mediators. The resulting PbR-based query planner is scalable, flexible, has anytime behavior, and yields a novel combination of traditional cost-based query optimization and heterogeneous information source selection.

The chapter is organized as follows. First, we introduce the challenging domain of query planning in mediators. Second, we present how this domain is encoded as a classical planning problem. Third, we describe in detail how query planning can be performed within the Planning by Rewriting framework. We describe the cost of a query plan, the initial plan generator, the rewriting rules, and the search strategies. Fourth, we show the results of several scalability experiments. Finally, we discuss the advantages of using the PbR approach for query planning.

4.1 Query Planning in Mediators

Mediators provide access, in a given application domain, to information that resides in distributed and heterogeneous sources. These systems shield the user from the complexity of accessing and combining this information. The user interacts with the mediator using a single language with agreed upon semantics for the application domain, as if it were a centralized system, without worrying about the location or languages of the sources.

As motivation, consider the example of a mediator that integrates restaurant information available on the Web as shown in Figure 4.1. Some sources are restaurant guides, such as Zagat and Fodor's. These provide information such as the type of cuisine, the cost, the rating, and a review of

the restaurants, in addition to basic information such as address, phone, etc. Other sources, such as the Tiger map server, allow the system to plot latitude and longitude coordinates on a map. Other sources, such as the ETAK geocoder, can translate from the street address of some location to the corresponding latitude and longitude coordinates. Finally, some cities publish health inspection information of food establishments. Having access to all these sources the mediator could answer queries such as: *Get the names, reviews, and map of all Chinese restaurants in Monterey Park with excellent food and health ratings.* The corresponding query plan involves accessing and combining information from the sources mentioned above: the restaurant sources in order to get the names, rating, reviews, and addresses of the restaurants; the health ratings source; a geocoder to translate the restaurant addresses to latitudes and longitudes; and, finally, a map server to obtain maps with the results displayed graphically.[1]

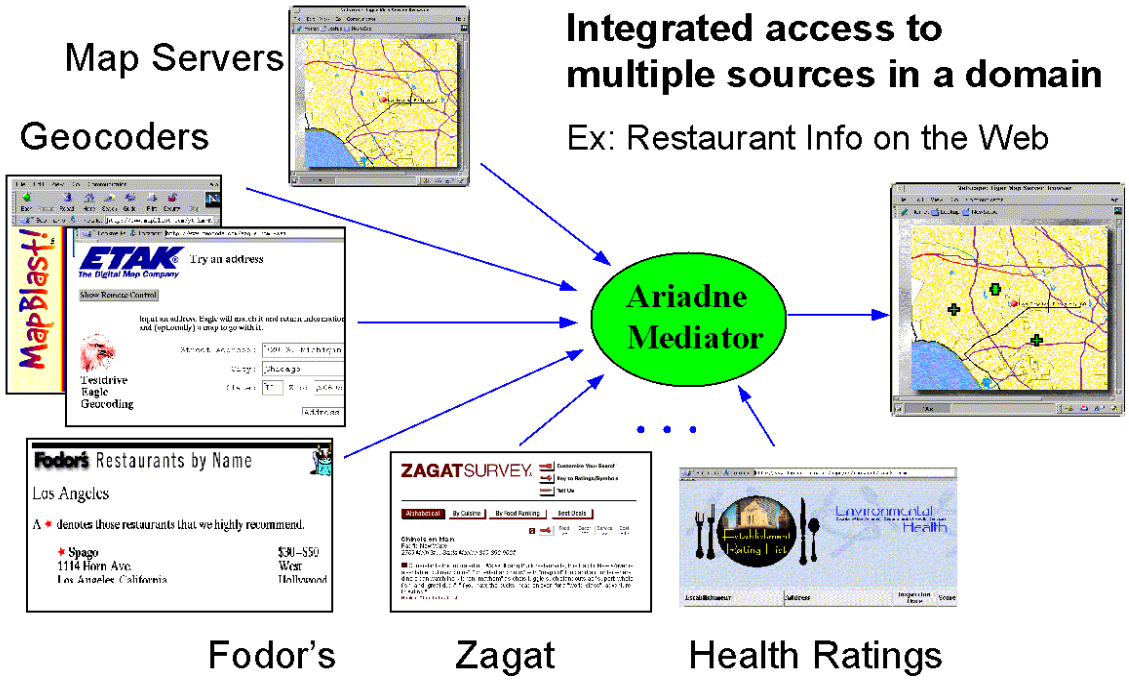


Figure 4.1: Example of Mediator in the Web

Mediators were initially developed to integrate structured information sources, such as databases. As this example suggests sources on the web provide only semistructured information. Nevertheless, we can apply the same mediator technology by wrapping the Web sources. A *wrapper* extracts the contents of a page according to its underlying conceptual schema. Wrappers can be programmed by hand or learned automatically [Kushmerick, 1997, Muslea *et al.*, 1998]. Also semantic mark-up

[1] Some of the sources in this example have a restriction on the type of queries they can answer known as binding pattern constraints. A binding pattern is a requirement that the values for some attributes must be given in order to access the values for the rest of the attributes. For example, the ETAK geocoder requires a street address as input to output its latitude and longitude. In this chapter we are not focusing on query planning in the presence of binding patterns. However, the planning techniques we present can be extended to handle binding patterns (see Section 7.2).

languages such as XML can facilitate considerably the extraction of information from Web sources. For the purposes of this chapter we consider the sources to have a well-defined schema (be it because the sources are databases or because they are appropriately wrapped sources).

Query planning in mediators involves generating a plan that efficiently computes a user query from the relevant information sources. This plan is composed of data retrieval actions at diverse information sources and data manipulation operations, such as those of the relational algebra: join, selection, union, etc. For an efficient execution, the plan has to specify both from which sources each different piece of information should be obtained and which data processing operations are going to be needed and in what order. The first problem, source selection, is characteristic of distributed and heterogeneous systems. The second problem has been the focus of traditional query optimization in databases. The highly combinatorial nature of query planning in mediators arises from these two independent sources of complexity, namely, the selection of relevant information sources for a given query and the selection and ordering of data processing operations. In the remainder of this section we describe source selection and which mechanisms our system uses to handle this problem. Sections 4.2 and 4.3 show how source selection is combined with cost-based optimization in our system.

Mediators must provide a coherent conceptual view of the application domain. This requires providing mechanisms to resolve the semantic heterogeneity among the different sources. This is critical in order to select which information sources are relevant for a user query.

In order to reconcile the semantic differences the mediator designer defines a global model of the application domain, models of the contents of the sources, and integration statements that relate the source terms with the global domain model. There are two approaches to specify these integration statements. One approach [Arens *et al.*, 1996, Levy *et al.*, 1996a, Kwok and Weld, 1996, Duschka and Genesereth, 1997] is to define each source term as a logical formula over terms of the global domain model. Another approach is to define each domain term as a formula over source terms [Hammer *et al.*, 1995, Adali *et al.*, 1996, Haas *et al.*, 1997]. These two approaches have complementary strengths and weaknesses [Ullman, 1997]. The former approach has the advantage of facilitating the addition of new sources to the mediator, as the new source definitions do not interact with the previous ones. The disadvantage is that finding the relevant sources is computationally harder [Levy *et al.*, 1995, Duschka, 1997]. Conversely, the latter approach facilitates the query processing, which is reduced to unfolding (the terms in the user query, which are domain terms, are simply substituted by the formula of source terms given in the integration statement), but adding new sources may involve extensive changes to the mediator definitions.

In our mediators (SIMS [Arens *et al.*, 1996] and Ariadne [Knoblock *et al.*, 1998]) we combine the strengths of both approaches by defining source terms as formulas on the global model and *precompiling* the inverse formulas (domain terms as a combination of source terms) before any query planning starts. This allows our system to plan more efficiently by unfolding, but still accept new source definitions without restructuring the domain model. During query planning our system only uses the inverted formulas, which we will call *integration axioms* for the remainder of the chapter. Our system precompiles a set of maximal integration axioms when the domain model

is defined. These axioms are maximal in the sense that they specify the maximum number of attributes that can be obtained from a given combination of sources. The relevant axioms for a given user query can be efficiently computed by instantiating these maximal axioms at run time. A detailed explanation of the algorithm for automatic compilation of the integration axioms lies outside the scope of this work (see [?]). These integration axioms are analogous to the inverted rules in [Duschka and Genesereth, 1997, Duschka, 1997]. However, we are using a description logic formalism as opposed to datalog,[2] and we only precompile a selected small set of integration axioms (the maximal axioms). In any case, the query planning techniques that we introduce in this chapter are independent of the axiom compilation algorithm. For, example our planner could be adapted to use axioms compiled using the techniques in [Duschka and Genesereth, 1997] or [Levy *et al.*, 1996a] (for the non-recursive cases). For the purposes of this chapter we will consider these integration axioms as given.

4.2 Query Planning as a Classical Planning Problem

The first step in using PbR is to encode the query planning problem as a classical planning domain. The operators for query processing and the encoding of information goals that we use were introduced in [Knoblock, 1996]. In this encoding the main state predicate is essentially `(available ?source ?query)`, which states that a particular set of information, represented declaratively by a query, is available at a particular location (`?source`) in the network. A sample information goal is shown in Figure 4.2. This goal asks to send to the output device of the mediator all the names of airports in Tunisia. The desired query is expressed in the Loom query syntax that the SIMS mediator accepts.[3] Note that the query is represented as a complex term within the `available` predicate. The query language addressed in this planning domain is essentially union of conjunctive queries with interpreted predicates (over distributed and heterogeneous sources).

```
(available output (retrieve (?ap_name)
                            (:and (airport ?aport)
                                  (country-name ?aport "Tunisia")
                                  (port-name ?aport ?ap_name))))
```

Figure 4.2: Sample Information Goal

[2] In our mediators the global model and the user queries are specified in the Loom description logic [MacGregor, 1988], a very expressive knowledge representation system.

[3] Although other query languages could have been used. For example, the same query in SQL would be:
`select port-name from airport where country-name = 'Tunisia'`

The specification of the operators is shown in Figure 4.3.[4] The operators have a similar structure. The preconditions impose that certain sets of data, described intensionally by queries, are available at some location in the network, and that these queries satisfy some properties. Once the queries in the preconditions are obtained, the operator processes them to produce a new resulting query. In order to check for properties of the queries, the operators rely on user-defined interpreted predicates. Interpreted predicates are satisfied by evaluating a function on its arguments (possibly producing bindings for some of the arguments), as opposed to being satisfied by effects of other operators (or the initial state) as normal state predicates.

As an example of the behavior of these operators, consider the `join` operator. The `join` operator needs as preconditions two subqueries available locally at the mediator (`?query-a` and `?query-b`). Combining these subqueries using some join conditions (`?join-conds`) produces the joined query `?query`. Perhaps a simpler way to understand the definition of the `join` operator is to analyze its behavior in a regression planner (such as UCPOP [Penberthy and Weld, 1992] or Sage [Knoblock, 1995]). Consider the top query in Figure 4.4 (associated with the variable `?query`). This conjunctive query obtains the `port-name`s and `country-code`s of airports by joining the `airport` and `location` classes on the `geoloc-code` attribute. For brevity we will call this query `query0`. Assume a goal `(available sims query0)` is posted to the planner and that the `join` operator is selected to satisfy such goal (because the effect of the `join` operator unifies with this goal). The preconditions of the `join` are posted then as new subgoals. Assume the interpreted predicate `join-query` is chosen first. Conceptually, the predicate `join-query` checks that `?query` is a conjunctive query that can be decomposed into two subqueries, `?query-a` and `?query-b`, joined with some conditions `?join-conds`. In order for the predicate to be evaluated, either `?query` must have a binding (returning bindings for `?query-a`, `?query-b` and `?join-conds`), or `?query-a`, `?query-b`, and `?join-conds` must have bindings (returning a binding for `?query`), or all variables are bound (returning true or false). In our example `?query` is the only variable with a binding at the time in which the interpreted predicate is called (the binding was performed when the effect of the join operator was unified with the posted goal). As this query is indeed conjunctive, the interpreted predicate computes bindings that decompose `query0` into simpler subqueries. Thus, `?query-a`, `?query-b`, and `?join-conds` will receive the bindings shown in Figure 4.4. These values are propagated to the remaining preconditions, `(available sims ?query-a)` and `(available sims ?query-b)`, that become new subgoals and are solved in a similar fashion.

The remaining operators behave analogously to `join`, using supporting interpreted predicates to analyze the queries. The `select` operator performs relational algebra selections. The

[4] Note that the predicate `available` actually used in the operators is `(available ?source ?host ?query !result)`. This predicate allows for several sources to reside at the same host machine, as well as having the same source replicated at different machines. The run-time variable `!result` contains the result (tuples, objects) after execution of the query. Run-time variables are used by the planner to propagate data throughout the plan and to explicitly reason with data gathered at run-time. For simplicity in our examples, we will use `(available ?source ?query)`.

```
(define (operator output)
   :parameters (?query ?result)
   :resources ((processor sims))
   :precondition (available local sims ?query ?result)
   :effect (available output sims ?query))

(define (operator retrieve)
   :parameters (?source ?host ?query !result)
   :resources ((processor ?host))
   :precondition (:and (source-available ?source ?host)
                       ( source-acceptable-query ?query ?source))
   :effect (available local sims ?query !result))

(define (operator assign)
   :parameters (?assignment ?subquery ?query ?subresult !result)
   :precondition (:and (available local sims ?subquery ?subresult)
                       ( assignment-query ?query ?assignment ?subquery))
   :effect (available local sims ?query !result))

(define (operator select)
   :parameters (?selection ?subquery ?query ?subresult !result)
   :precondition (:and (available local sims ?subquery ?subresult)
                       ( selection-partition-all ?query ?selection ?subquery))
   :effect (available local sims ?query !result))

(define (operator join)
   :parameters (?join-conds ?query ?query-a ?query-b ?result-a ?result-b !result)
   :precondition (:and (available local sims ?query-a ?result-a)
                       (available local sims ?query-b ?result-b)
                       ( join-query ?query ?join-conds ?query-a ?query-b))
   :effect (available local sims ?query !result))

(define (operator binary-union)
   :parameters (?subquery-a ?subquery-b ?query ?result-a ?result-b !result)
   :precondition (:and (available local sims ?subquery-a ?result-a)
                       (available local sims ?subquery-b ?result-b)
                       ( binary-union-query ?query ?subquery-a ?subquery-b))
   :effect (available local sims ?query !result))
```

Figure 4.3: Operators for Query Planning (interpreted predicates in *italics*)

```
?query:   (retrieve (?pn ?cc)
            (:and (airport ?a) (port-name ?a ?pn) (geoloc-code ?a ?gc1)
                  (location ?l) (geoloc-code ?l ?gc2) (country-code ?l ?cc)
                  (= ?gc1 ?gc2)))

?query-a: (retrieve (?pn ?gc1)
            (:and (airport ?a) (port-name ?a ?pn) (geoloc-code ?a ?gc1)))

?query-b: (retrieve (?cc ?gc2)
            (:and (location ?l) (geoloc-code ?l ?gc2) (country-code ?l ?cc)))

?join-conds: (= ?gc1 ?gc2)
```

Figure 4.4: Behavior of the join-query interpreted predicate

`binary-union` performs the relational algebra union of two queries. The `assign` operator handles queries in which new attributes are generated on the fly as arithmetic expressions over other attributes of a query.

The `retrieve` operator executes a query at an information source provided that the source is in operation at some host machine `(source-available ?source ?host)` and that the source is capable of processing the query `(source-acceptable-query ?query ?source)`. The SIMS mediator keeps a model of the capabilities of the sources. The interpreted predicate `source--acceptable-query` checks the requirements of the query against the capabilities of the source. For example, `query0` could not be accepted by a source that cannot perform a join operation even if both `airport` and `location` were present at such source. Such a query would have to be decomposed first into the two simpler queries shown in Figure 4.4 and the join processed locally. After executing the retrieve operator the results are available locally at the SIMS mediator.

As defined in Figure 4.3, all these data processing operators are executed locally at the SIMS mediator. Note that generally a mediator does not have any control on the execution plans generated at a remote site. However, this planning domain can be easily generalized to represent query plans that could execute remotely. Simply we would replace the constants `local` and `sims` in the occurrences of the `available` predicate in the operators with variables that could be instantiated to the appropriate locations in the network.

Two plans generated using this planning domain specification appear in Figures 4.5 and 4.6. Both plans evaluate the query in Figure 4.2 but at a very different cost (as we will explain in Section 4.3.1). These plans involve three source accesses, two joins, a select and an output operations. The reason why the seemingly simple query in Figure 4.2 expands into a plan with seven operations lies in the fact that in the application domain there is not a single source that answers that query, but such information has to be composed from three different sources.

As we explained earlier, the mediator reconciles the semantic differences among the sources using integration axioms. The axiom relevant for the query in Figure 4.2 and the plans of Figures 4.5 and 4.6 is:

```
airport(country-code country-name geoloc-code port-name) ⇔
    airport(geoloc-code port-name) ∧ location(country-code geoloc-code) ∧
    country(country-code country-name)
```

This axiom states that the system can obtain the attributes `country-code`, `country-name`, `geoloc-code`, and `port-name` for the `airport` class, by performing the join of three sources: one that contains `geoloc-code` and `port-name` of `airport`s, a second source that provides `country-code` and `geoloc-code` of `location`s, and, finally, a third source that lists `country-code`s and `country-name`s.[5] This axiom shows the general structure of our integration axioms. They always have a single domain class on one side of the biconditional, called the axiom head, and a positive existential formula on the other side, called the axiom body.

The plans of Figures 4.5 and 4.6 are essentially alternative algebraic representations of the integration axiom above, with the addition of the selection on `country-name` required in the query. Note that the body of the axiom is expressed in terms of the domain model in order to provide a level of abstraction over the possible sources for a query. For example, `airport(geoloc-code port-name)` is obtained from the `geoh@higgledy.isi.edu` source in the plan of Figure 4.5 and from the `port@local` source in the plan of Figure 4.6. The interpreted predicates in the operators consult the integration axioms in the domain when processing a query. In our example, the `join-query` interpreted predicate, will successively partition the user query of Figure 4.2 making explicit the two joins present in the axiom above.

4.3 Planning by Rewriting for Query Planning in Mediators

This section describes in detail the application of the Planning by Rewriting framework to the problem of query planning in mediators. For expository purposes, we will explain the approach following the main issues of local search as we did in the previous chapter. First, we discuss the factors that affect the quality of a query plan. Second, we describe how to generate initial query plans. Third, we present the plan rewriting rules used to optimize query plans. Finally, we describe the search methods we used in this domain.

4.3.1 Query Plan Quality

A significant advantage of PbR is that a complete plan is always available during the planning process. Therefore, the user can specify complex plan quality metrics. In generative planners the complete plan is not available until the search for a solution is completed, so usually only very simple plan quality metrics, such as the number of steps, have been used. There are many factors

[5] Our source descriptions and integration axioms assume that the sources provide complete information about the domain classes (note the use of ⇔), as opposed to the axioms in [Duschka and Genesereth, 1997] or [Levy *et al.*, 1996a]. This represents no loss of generality. The fact that a source class, `S`, provides partial information on a domain class, `C`, (i.e. provides a subset of the extension of the class) can be easily represented by relating `S` to a subclass of `C`.

that may affect the quality of a query plan. Traditionally the most important factor is the query execution time. However many other considerations may be relevant. For example, in the Web some sources may charge for the information delivered. Therefore a slower but monetarily cheaper plan may be preferable.

For our query planning domain the quality of a plan is an estimation of its execution cost. The execution cost of a distributed query plan depends on the size of intermediate results, the cost of performing data manipulation operations (e.g., join, selection, etc.), and the transmission through the network of the intermediate results from the remote sources to the mediator. We estimate the execution cost based on the expected size of the intermediate results. We assume that the transmission and processing costs are proportional to the size of the data involved. The query size estimation is computed from simple statistics obtained from the source relations, such as the number of tuples in a relation, the number of distinct values for each attribute, and the maximum and minimum values for numeric attributes (see [Silberschatz *et al.*, 1997]). As an example of query size estimation consider a relation `R(x y)` that has 100 tuples and that the attributes `x` and `y` have 100 and 20 distinct values respectively (`x` is a key). Under a uniform distribution assumption the expected number of tuples returned by the query `q(x y) :- R(x y)` $\wedge$ `y = 7` is 5 (100/20). If the query were `q(x y) :- R(x y)` $\wedge$ `x = 7` we would expect 1 tuple. Our cost function also prefers plans that can be evaluated in parallel. When a subplan has two parallel branches the cost of the subplan is the maximum of the cost each of the parallel branches.

As an example of plan quality, consider the plans in Figures 4.5 and 4.6. These are two alternative evaluation plans for the query in Figure 4.2 but with very different costs. In our application domain there is no single source that can completely answer this query, so it has to be computed by joining data from different sources. The sources in our example can only answer very simple queries, that is, the remote queries cannot contain selections, joins, etc.

Figure 4.5 shows a randomly generated initial plan. Figure 4.6 shows an optimized plan. Note that the initial plan is of much lower quality. The initial plan performs three sequential retrievals from the same source, `geoh`, at the same host, `higgledy.isi.edu`, not taking advantage of the possibility of executing the queries in parallel at different hosts. Also it is generally more efficient to perform selections as early as possible in order to reduce the data that the subsequent steps of the plan must process. The initial plan performs the selection step last as opposed to the optimized plan where it is done immediately after the corresponding retrieve.

Estimating the execution cost of a complex query plan is a hard problem. A major difficulty is the propagation of error. Small variances on the estimates of the base relations may cause large differences of the estimated cost of the whole query because the error get compounded repeatedly through the intermediate results [Ioannidis and Christodoulakis, 1991]. Cost estimation has been a topic of considerable research in the database community. More sophisticated estimation models are surveyed in [Mannino *et al.*, 1988].

The fact that the cost function is an estimate with a considerable degree of uncertainty supports using local search methods. Finding the global optima is not that valuable given it is only an estimation of the real value. In practice, it is enough to find a good quality local minima.

Another advantage of the PbR approach is that because it allows the interleaving of planning and execution it can perform a type of dynamic query optimization. As subqueries are answered during the execution of a query plan, the system can refine the query size estimates based on the actual results returned. This opens the opportunity to rewrite the remainder of the plan if the difference between expected and actual costs warrants it.

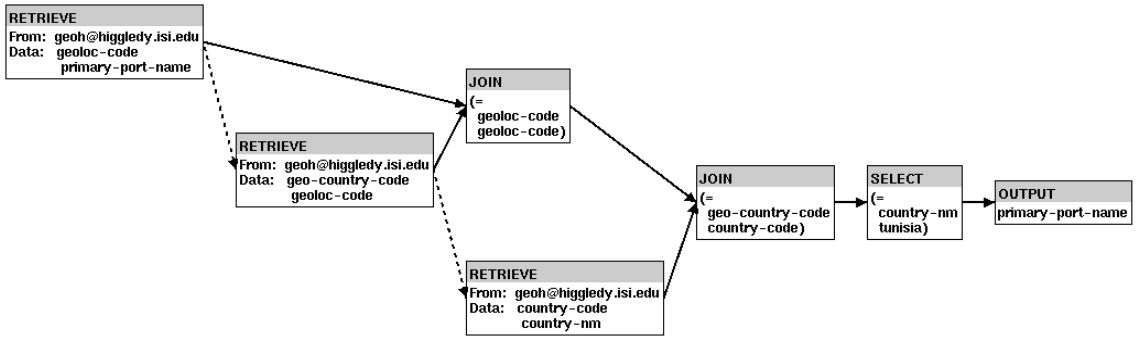


Figure 4.5: A Suboptimal Initial Query Plan

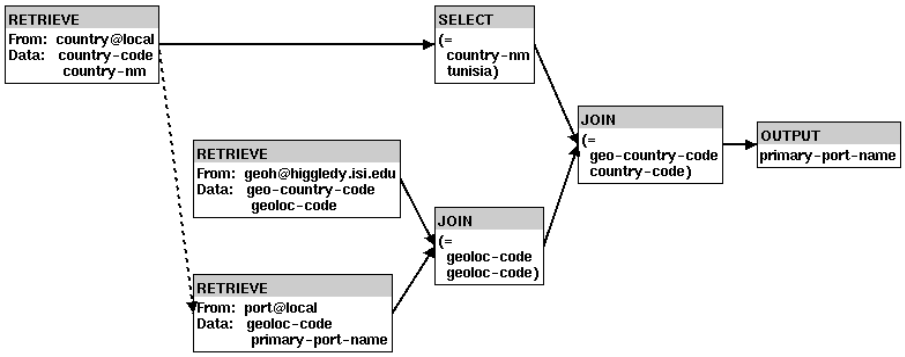


Figure 4.6: An Optimized Query Evaluation Plan

4.3.2 Initial Query Plan Generation

For the Planning by Rewriting framework to be applicable, there must exist an efficient mechanism to generate an initial solution plan. It is desirable that this mechanism also be able to produce several (possibly random) initial plans on demand. Both properties are satisfied by the query planning domain. Initial query evaluation plans can be efficiently obtained as random depth-first search parses of the query.

As an example, consider how the initial plan in Figure 4.5 is obtained from the query in Figure 4.2 by choosing randomly among the alternatives at each parsing and expansion point. The generation of this plan is best understood starting from the `output` step (from right to left). The first choice is either to introduce the `select` operator or expand the query using the integration

axiom above. The `select` operator is chosen. Because the remaining query contains only one class but no source can provide all the required attributes, the next decision is fixed. The `airport` class must be expanded using an integration axiom. Assume the axiom above is chosen. This new query is a conjunction of three classes. The initial plan generator chooses randomly in which order to perform the two joins. Finally, the three remaining single class queries can be executed at several alternative sources. For the initial plan of Figure 4.5 the same source `geoh@higgledy.isi.edu` happens to be selected. These initial plans are correct and efficient to produce, but they may be of very low quality.

The user can customize the initial plan generator to the characteristics of the domain. In domains where planning time is very limited, we may want to provide an initial plan generator that heuristically produces better quality plans than a random generator. For example, as we mentioned before, it is beneficial in most cases to perform selections as early as possible. The initial plan generator can incorporate this and other similar heuristics in order to achieve higher quality initial plans. Note that starting from a higher quality initial plan does not guarantee that we find the optimal any sooner. Sometimes, good local minima can be reached more easily from a random distribution of initial plans. Similarly, having a bias so strong that produces only a single initial plan may cause a less efficient search of the solution space. Given that there are many choice points in producing an initial plan, we may want to bias only a few aspects, such as the placement of selections, so that a good coverage of the space can still be achieved. The initial plan generator used in our experiments chooses randomly the sources, the integration axioms, and the join, union, and the assignment orders, but it tries to perform selections as early as possible.

4.3.3 Query Plan Rewriting Rules

The core of the planning process consists of the iterative application of a set of plan rewriting rules until a plan of acceptable quality is found. In our query planning domain, the rules can be grouped into three classes according to their origin. The rules are derived from properties of the distributed environment, the relational algebra, and the resolution of the semantic heterogeneity in the application domain. We describe each of these in the following subsections.

4.3.3.1 Rewriting Rules from the Distributed Environment

The first class of rules is derived from the properties of the distributed environment. A logical description of these rules is shown in Figure 4.7. The `Source-Swap` rule allows the planner to explore the choice of alternative information sources that can satisfy the same query but may have different retrieval or transmission costs. This rule is not only necessary for query plan optimization but it also serves as a repair rule when planning and execution are interleaved. Suppose that the planner started executing a plan and one of the sources needed went down, then the subquery sent to that source will fail. By applying this rule PbR can repair the plan and complete the execution without replanning and re-executing from scratch.

The `Remote-Join-Eval`, `Remote-Selection-Eval`, `Remote-Assignment-Eval`, and `Remote--Union-Eval` rules rely on the fact that, whenever possible, it is generally more efficient to execute a group of operations together at a remote information source than to transmit the data over the network and execute the operations at the local system. Note the need for checking the capabilities of the information sources, as we do not assume that sources are full databases. The sources may have no query processing capabilities (for example, wrapped WWW pages) or support very limited types of queries (for example, WWW forms).

Figure 4.8 shows the `Remote-Join-Eval` rule in the input syntax accepted by the PbR planner. This rule specifies that if in a plan there exist two retrieval operations at the same remote database, which results are consequently joined, and the remote source is capable of performing joins, the system can rewrite the plan into one that contains a single retrieve operation that pushes the join to the remote database. A graphical example of the application of this rule during query planning is shown in Figure 4.14. Similarly, the three remaining rules cover the other algebraic operators that the SIMS language supports (selection, assignment, and union). These rewriting rules apply successively to ensure that complex queries are evaluated remotely if possible.

Source-Swap: $retrieve(Q, Source1) \wedge alternative\text{-}source(Q, Source1, Source2)$
$\Rightarrow retrieve(Q, Source2)$

Remote-Join-Eval: $(retrieve(Q1, Source) \bowtie retrieve(Q2, Source))$
$\wedge capability(Source, join)$
$\Rightarrow retrieve(Q1 \bowtie Q2, Source)$

Remote-Selection-Eval: $\sigma_A retrieve(Q1, Source) \wedge capability(Source, selection)$
$\Rightarrow retrieve(\sigma_A Q1, Source)$

Remote-Assignment-Eval: $assign_{X:=f(Ai)} retrieve(Q1(Ai), Source) \wedge$
$capability(Source, assignment)$
$\Rightarrow retrieve(assign_{X:=f(Ai)} Q1(Ai), Source)$

Remote-Union-Eval: $(retrieve(Q1, Source) \cup retrieve(Q2, Source)) \wedge$
$capability(Source, union)$
$\Rightarrow retrieve(Q1 \cup Q2, Source)$

Figure 4.7: Transformations: Distributed Environment

The transformations in Figure 4.7 are logically bidirectional. The designer has the choice of implementing rules that correspond to both directions or only one. Implementing both directions will cover the entire solution space. However, some directions may be much preferred. For example, the direction that pushes operations to the remote sources is generally more useful, thus only that direction may be implemented as a rewriting rule in the system. This results in a loss of accessibility to some optima, but the savings in planning time may outweigh reaching some rarely occurring optima. This is a version of the utility problem [Minton, 1990].

```
(define-rule :name remote-join-eval
  :if (:operators ((?n1 (retrieve ?query1 ?source))
                   (?n2 (retrieve ?query2 ?source))
                   (?n3 (join ?query ?jc ?query1 ?query2)))
       :constraints ((capability ?source 'join)))
  :replace (:operators (?n1 ?n2 ?n3))
  :with (:operators ((?n4 (retrieve ?query ?source))))
```

Figure 4.8: Remote-Join-Eval Rewriting Rule

4.3.3.2 Rewriting Rules from the Relational Algebra

The second class of rules are derived from the commutative, associative, and distributive properties of the operators of the relational algebra. A logical description of these rules is shown in Figure 4.9. The `Join-Swap` rule in the PbR syntax is shown in Figure 4.10 (a). A graphical schematic of the behavior of this rule appears in Figure 4.10 (b). This rule specifies that two consecutive joins operators can be reordered and allows the planner to explore the space of join trees. In our query planning domain [Knoblock, 1996] queries are expressed as complex terms. The PbR rules use the interpreted predicates in the `constraints` field to manipulate such query expressions. The interpreted predicates in the rewriting rules are analogous to the those in the operators. For example, the `join-swappable` predicate checks if the two join operators have queries that can be exchanged. This user-defined predicate takes as input the description of the two join operations (the first eight variables which must have bindings) and produces as output the description of the two reordered join operations (as bindings for the last eight variables). For example, in the schematic in Figure 4.10 (b) the two join operators, `join(q0 jc12 q1 q2)` and `join(q1 jc34 q3 q4)`, that compose the query tree on the left of Figure 4.10 (b) can be reordered in two ways generating the two trees on the right of Figure 4.10 (b): one tree with joins `join(q0 jc35 q5 q3)` and `join(q5 jc24 q2 q4)`, and the other tree with `join(q0 jc54 q5 q4)` and `join(q5 jc32 q3 q2)`. If two subqueries do not share any attributes, a join degenerates into a cross-product. Although a cross-product is inefficient, such rewritings are allowed for completeness. Sometimes in order to get to a low cost plan the search may step through a plan of cost higher than the current one.

Join-Swap: $Q1 \bowtie (Q2 \bowtie Q3) \Leftrightarrow Q2 \bowtie (Q1 \bowtie Q3) \Leftrightarrow Q3 \bowtie (Q2 \bowtie Q1)$

Selection-Join-Swap: $\sigma_A(Q1 \bowtie Q2) \Leftrightarrow \sigma_A Q1 \bowtie Q2$

Selection-Union-Swap: $\sigma_A(Q1 \cup Q2) \Leftrightarrow \sigma_A Q1 \cup \sigma_A Q2$

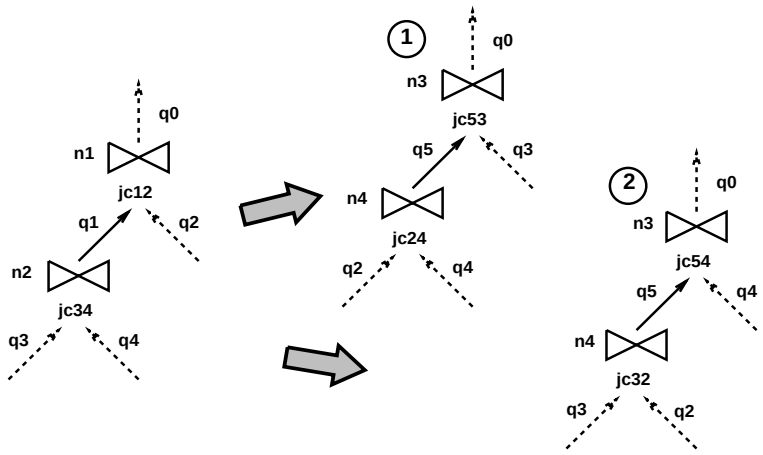
Assignment-Swap: $assign_{X:=f(Ai)}(Q1(Ai) \bowtie Q2) \Leftrightarrow assign_{X:=f(Ai)}Q1(Ai) \bowtie Q2$

Join-Union-Distributive: $Q1 \bowtie (Q2 \cup Q3) \Leftrightarrow (Q1 \bowtie Q2) \cup (Q1 \bowtie Q3)$

Figure 4.9: Transformations: Relational Algebra

```
(define-rule :name join-swap
  :if (:operators ((?n1 (join ?q1 ?jc1 ?sq1a ?sq1b))
                   (?n2 (join ?q2 ?jc2 ?sq2a ?sq2b)))
       :links (?n2 ?n1)
       :constraints (join-swappable ?q1 ?jc1 ?sq1a ?sq1b   ;; in
                                    ?q2 ?jc2 ?sq2a ?sq2b   ;; in
                                    ?q3 ?jc3 ?sq3a ?sq3b   ;; out
                                    ?q4 ?jc4 ?sq4a ?sq4b)) ;; out
  :replace (:operators (?n1 ?n2))
  :with (:operators ((?n3 (join ?q3 ?jc3 ?sq3a ?sq3b))
                     (?n4 (join ?q4 ?jc4 ?sq4a ?sq4b)))
         :links (?n4 ?n3)))
```

(a) Rule in PbR syntax



(b) Schematic

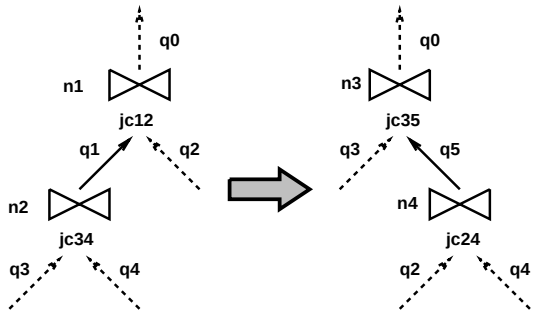
Figure 4.10: Join-Swap Rewriting Rule

Our rule rewriting language offers a great deal of flexibility to define different types of rewriting rules with different trade-offs between complexity and utility. For example, instead of the `join-swap` rule in Figure 4.10, we could have defined a join associativity rule more directly as in Figure 4.11. The `join-associativity` rule is shown in PbR syntax in Figure 4.11 (a) and as a graphical schematic in Figure 4.11 (b). This is a simpler rule but it also makes more commitments than `join-swap`. The `join-associative` interpreted predicate is simpler. It only has 3 output variables (`?jc24 ?q5 ?jc35`) as opposed to the 8 output variables of `join-swappable`. However, `join-associative` enforces the matching join structure to be a left tree and produces a right join tree as shown in Figure 4.11 (b). Therefore we would also need another rule for join commutatitivy as the one shown in Figure 4.12, or specify three more "associativity" rules in order to cover all four cases that result from the combination of join associativity and commutativity. A disadvantage

of adding the commutativity rule is that the rewritten plans have the same quality. It is only after some other rule applies (e.g., the `join-associativity`) when the advantage of some ordering becomes apparent. A greater number of rules decreases the performance of matching. Besides, rules that are discriminative in the cost surface are more useful. The `join-swap` conveniently combines all the four cases into one and produces rewritings of very different cost which helps to move through the cost space.

```
(define-rule :name join-associativity
  :if (:operators ((?n1 (join ?jc12 ?q0 ?q1 ?q2))
                   (?n2 (join ?jc34 ?q1 ?q3 ?q4)))
       :links (?n2 ?n1)
       :constraints (join-associative ?jc12 ?q0 ?q2 ?jc34 ?q1 ?q3 ?q4 ;; in
                                      ?jc24 ?q5 ?jc35))             ;; out
  :replace (:operators (?n1 ?n2))
  :with (:operators ((?n3 (join ?jc35 ?q0 ?q3 ?q5))
                     (?n4 (join ?jc24 ?q5 ?q4 ?q2)))))
```

(a) Rule in PbR syntax



(b) Schematic

Figure 4.11: Join-Associativity Rewriting Rule

```
(define-rule :name join-commutativity
  :if (:operators (?n1 (join ?jc12 ?q0 ?q1 ?q2)))
  :replace (:operators (?n1))
  :with (:operators (?n2 (join ?jc12 ?q0 ?q2 ?q1)))
```

Figure 4.12: Join-Commutativity Rewriting Rule

4.3.3.3 Rewriting Rules from the Integration Axioms

The third set of rewriting rules arises from the heterogeneous nature of the environment. As we explained earlier, our mediator reconciles the semantic differences among the sources using a set of integration axioms (cf. Section 4.2). Our system automatically derives query-specific plan rewriting

rules from these integration axioms in order to explore the alternative ways of obtaining each class of information in a user query. Two sample integration axioms relevant for the query in Figure 4.2 are:

(a) `airport(country-code port-name)` $\Leftrightarrow$
`airport(geoloc-code port-name)` $\wedge$ `location(geoloc-code country-code)`

(b) `airport(country-code port-name)` $\Leftrightarrow$
`airport(port-name)` $\wedge$ `port-location(port-name country-code)`

Axiom (a) states that in order to obtain the attributes `country-code` and `port-name` of the `airport` class, the system needs to join data from two sources. The first source provides `geoloc--code` and `port-name` of `airport`s, and the second provides `geoloc-code` and `country-code` of geographic `location`s. In our model the class `airport` is a subclass of `location` so it inherits its attributes. Thus, the `airport` class has three attributes: `geoloc-code`, `country-code`, and `port-name`. Axiom (b) is similar but uses a more specialized source for location only applicable to ports, `port-location`, and joins on the `port-name` key attribute (as opposed to joining on `geoloc-code`).

The rewriting rules corresponding to the integration axioms above are shown in Figure 4.13. Rule (a) corresponds to the axiom (a). This rule states that if in a plan there is a set of steps (`?nodes`) that obtain attributes `country-code` and `port-name` of the `airport` class, the planner could alternatively obtain this information using the axiom shown above. That is, the `?nodes` identified in the rule antecedent will be removed from the plan and replaced by the join and the two retrieve steps in the rule consequent. Similarly, rule (b) specifies the alternative of using axiom (b) to obtain `airport(country-code port-name)`. Note that the consequents of these rules are just the algebraic versions of the bodies of the axioms.

This type of rewriting rules are used to exchange alternative integration axioms in a plan. There are several subtle points to this rule specification. First, for n integration axioms for the same domain class and attributes, our system only writes n rules as opposed to the n^2 direct exchanges. This is possible because we use the axiom head as an abstraction of all these alternative axioms. Note how in the antecedents of the rules in Figure 4.13 only the axiom head is mentioned. Second, the interpreted predicate `identify-axiom-steps` finds all the steps that implement the body of an axiom in a very simple way, it just looks for an step in the query plan that produces a query that matches the axiom head. The subtree rooted at such step is the implementation of the axiom body. This fact follows from the encoding of the query planning domain with the operators in Figure 4.3.

This type of rewriting rules resemble task expansion in Hierarchical Task-Network Planning [Erol *et al.*, 1994, Tate, 1977]. The class of information desired, described by the axiom head, takes the role of a higher level task. The axiom body is analogous to the specification of the task expansion. However, in PbR the rewriting rules are used for local search as opposed to for generative planning.

```
(define-rule :name (<=> (airport country-code port-name)
                        (:and (airport geoloc-code port-name)
                              (location geoloc-code country-code)))
 :if (:constraints (identify-axiom-steps (airport country-code port-name) ?nodes))
 :replace (:operators ?nodes)
 :with (:operators ((?n1 (retrieve port@local (airport geoloc-code port-name)))
                    (?n2 (retrieve geoh@higgledy.isi.edu
                                   (location geoloc-code country-code)))
                    (?n3 (join (airport country-code port-name)
                               ((= geoloc-code.1 geoloc-code.2))
                               (airport geoloc-code.1 port-name)
                               (location geoloc-code.2 country-code))))))

                                   (a)

(define-rule :name (<=> (airport country-code port-name)
                        (:and (airport port-name)
                              (port-location port-name country-code)))
 :if (:constraints (identify-axiom-steps (airport country-code port-name) ?nodes))
 :replace (:operators ?nodes)
 :with (:operators ((?n1 (retrieve port@local (airport port-name)))
                    (?n2 (retrieve geoh@higgledy.isi.edu
                                   (port-location port-name country-code)))
                    (?n3 (join (airport country-code port-name)
                               ((= port-name.1 port-name.2))
                               (airport port-name.1)
                               (port-location port-name.2 country-code))))))

                                   (b)
```

Figure 4.13: Rewriting Rule for Integration Axiom

4.3.4 Searching the Space of Query Plans

The space of rewritings for query planning is too large for complete search methods to provide an acceptable performance. The fact that in many cases, such as query planning, the quality of a plan can only be estimated supports the argument for possibly incomplete search strategies, such as gradient descent or simulated annealing. The effort spent in finding the global optimum may not be justified given that the cost function only captures approximately the real costs in the domain. As the accuracy of the cost model increases the planner may perform a more complete search of the plan space. In order to explore the space of query plans our planner currently uses variations of gradient descent (steepest and first-improvement) with random restart to escape low-quality local minima and a fixed-length random walk to traverse plateaus.

Figure 4.14 shows an example of the local search through the space of query plan rewritings in a simple distributed domain that describes a company. The figure shows alternative query evaluation plans for a conjunctive query that asks for the names of employees, their salaries, and the projects

they are working on. The three relations requested in the query (`Employees`, `Payroll`, and `Project`) are distributed among two databases (one at the company's headquarters – `HQ-db` – and another at a branch – `Branch-db`). Assume that the leftmost plan is the initial plan. This plan first retrieves the `Employee` relation at the `HQ-db` and the `Project` relation at the `Branch-db`, and joins these two tables on the employee `name`. Then, the plan retrieves the `Payroll` relation from the `HQ-db` and joins it on `ssn` with the result of the previous join. Although a valid plan this initial plan is suboptimal. Applying the `join-swap` rule to this initial plan generates two rewritings. One of them involves a cross-product, which is a very expensive operation, so the system, following a gradient descent search strategy, prefers the other plan. Now the system applies the `remote-join-eval` rule and generates a new rewritten plan that evaluates the join between the employee and project tables remotely at the headquarters database. This final plan is of much better quality. A detailed example of the rewriting process in query planning in mediators appears in appendix A.

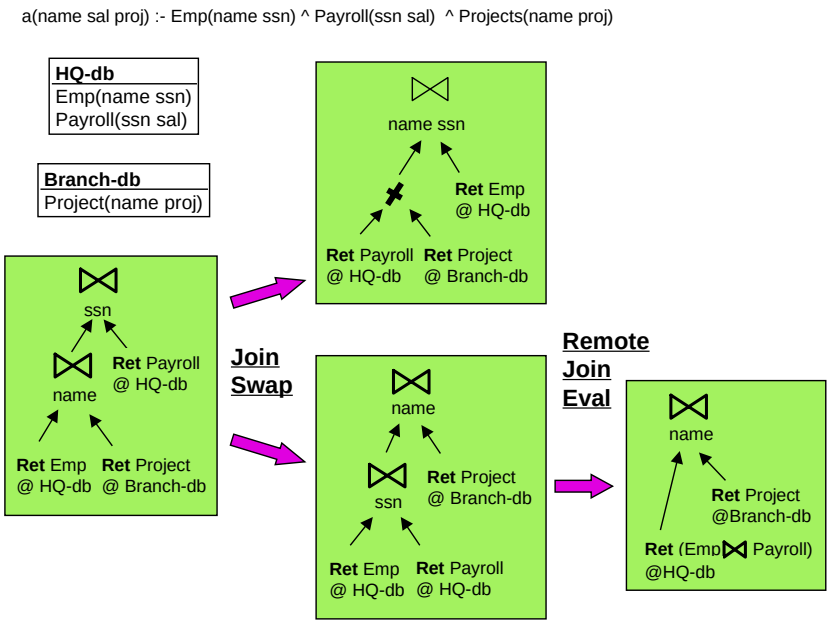


Figure 4.14: Rewriting in Query Planning

4.4 Experimental Results in Query planning

The Planning by Rewriting approach has been fully implemented and applied in several application domains. In this section we report some scalability results for PbR applied to query planning in

mediators. We compare the planning efficiency and plan quality of different configurations of PbR with other existing query planners. We present results for three query planners:

Sage: This is the original query planner [Knoblock, 1995, Knoblock, 1996] for the SIMS mediator, which performs a best-first search with a heuristic commonly used in query optimization that explores only the space of left join trees. Sage is a refinement planner that generates optimal left-tree query plans.

Initial: This is the initial plan generator for PbR. It generates random depth-first search parses of the query. It is the fastest planner but may produce very low quality plans.

PbR: We explored several gradient descent strategies and combinations of the rewriting rules introduced in Section 4.3.3.

We have designed a three controlled experiments that test our PbR-based query planner along each of the different factors that contribute to the complexity of query planing in mediators. In the first experiment we compare the behavior of PbR, Initial, and Sage in a distributed query planning domain as the size of the queries increases. In the second experiment we compare the planners in a distributed and heterogeneous domain by increasing the number of alternative sources per domain class. In the third experiment, we test the scalability of PbR and Sage in the presence of complex integration axioms, showing the effects of increasing the size of the integration axioms and the number of alternative axioms.

4.4.1 Distributed Query Planning

For the first experiment we generated a synthetic domain for the SIMS mediator and defined a set of conjunctive chain queries involving from 1 to 30 domain classes. The queries have one selection on an attribute of each class. The structure of the queries is shown in Figure 4.15. There is only one source class per domain class so the integration axioms are trivial. Each information source contains two source classes and can perform remote operations. Therefore, the optimal plans involve pushing operations to be evaluated remotely at the sources.

The initial plan generator splits the joins randomly but pushes the selections (e.g., `a3 >= 50`) to the sources. For PbR we defined the `Join-Swap` and the `Remote-Join-Eval` rules defined in Figures 4.10 and 4.8. These two rules are sufficient to optimize the queries in the test set. Rules involving selections are not used because the initial plans push the selections to the sources and this is a very good heuristic. We tested two gradient-descent search strategies for PbR: first improvement and steepest descent, both using three random restarts.

The results of this first experiment are shown in Figure 4.16. Figure 4.16 (a) shows the planning time (in CPU seconds) as the query size grows. The planning time is shown in a logarithmic scale. The times for PbR include both the generation of three random initial plans and their rewriting. The times for Initial are the average of the three random parses of each query. Sage is able to solve queries involving up to seven classes, but larger queries cannot be solved within the search

```
q2(j1 j2 a2 a3 b3) :-
    a(j1 a2 a3), b(j1 j2 b3), a3 >= 50, b3 >= 50.

q3(j1 j2 j3 a2 a3 b3 c3) :-
    a(j1 a2 a3), b(j1 j2 b3), c(j3 j2 c3),
    a3 >= 50, b3 >= 50, c3 >= 50.

q4(j1 j2 j3 j4 a2 a3 b3 c3 d3) :-
    a(j1 a2 a3), b(j1 j2 b3), c(j3 j2 c3), d(j3 j4 d3),
    a3 >= 50, b3 >= 50, c3 >= 50, d3 >= 50.

q5(j1 j2 j3 j4 j5 a2 a3 b3 c3 d3 e3) :-
    a(j1 a2 a3), b(j1 j2 b3), c(j3 j2 c3), d(j3 j4 d3), e(j5 j4 e3)
    a3 >= 50, b3 >= 50, c3 >= 50, d3 >= 50, e3 >= 50.
⋮
```

Figure 4.15: Queries for Distributed Query Planning

limit of 200,000 nodes. Both configurations of PbR scale better than Sage. The first improvement search strategy clearly dominates steepest descent in this experiment.

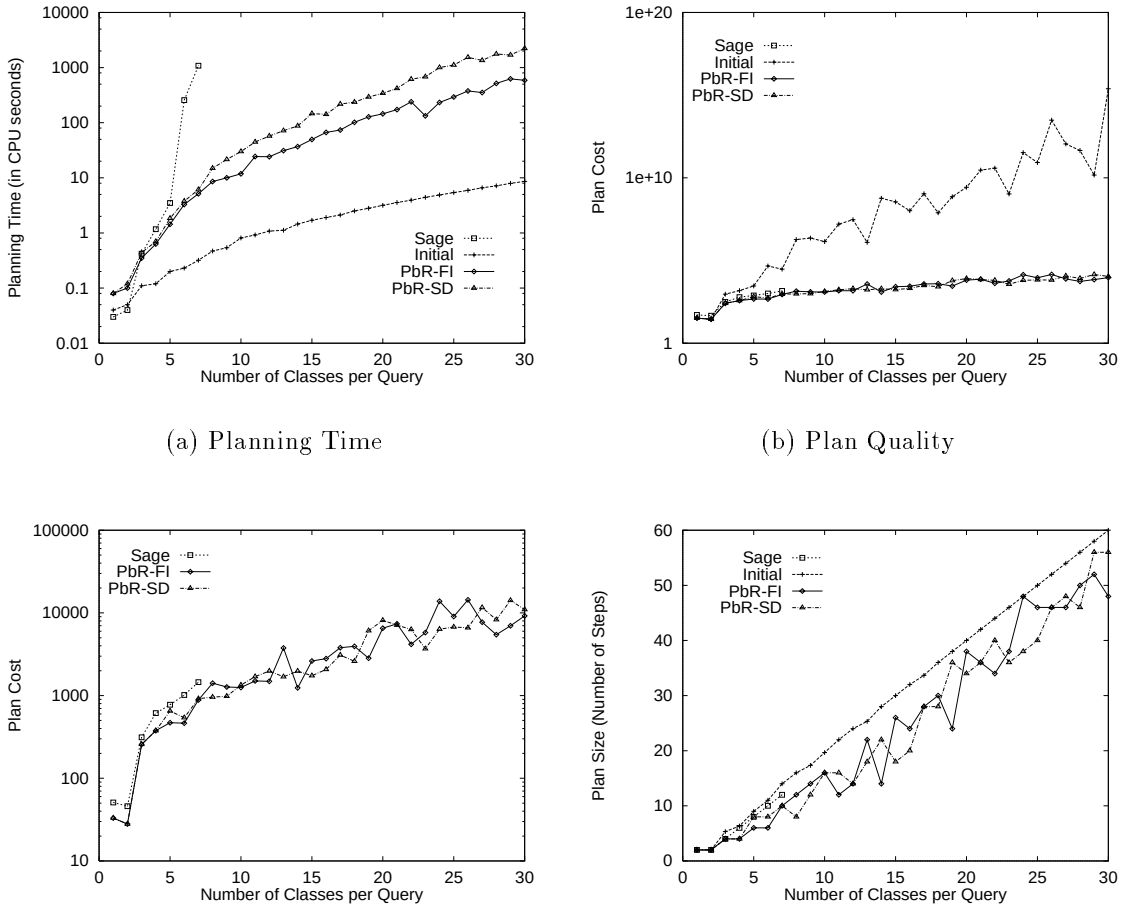
Figure 4.16 (b) shows the cost of the query plans for the four planners. The cost for Initial is the average of the three initial plans. Figure 4.16 (c) compares in detail the cost of the plans produced by Sage and the two configurations of PbR. The plan cost is an estimate of the query execution cost. A logarithmic scale is used because of the increasingly larger absolute values of the plan costs for our conjunctive chain queries and the very high cost of the initial plans. PbR rewrites the very poor quality plans generated by Initial into high-quality plans. PbR produces better quality plans than Sage for the range tractable for Sage and beyond that range it scales gracefully. PbR produces better quality plans because it searches the larger space of bushy query trees and can take greater advantage of parallel execution plans.[6]

Finally, Figure 4.16 (d) shows the number of steps in each plan. Note how the plans generated by PbR are smaller than those produced by Sage and Initial. This is due to two facts. First, the query plans that PbR produces are more parallel (the join trees are bushy) and thus they need fewer join operators. Second, the `Remote-Join-Eval` rule collapses three steps (two retrieves and join) into one (retrieve) when it is cost efficient to do so.

4.4.2 Scaling the Number of Alternative Sources

In the second experiment, we test the scalability of PbR as the number of alternative sources for each class of information in the domain model increases. We defined a set of synthetic domains with up to five domain classes but varying the number of alternative sources (source classes) per

[6] In order not to be unfair to Sage given its left-tree bias, all information sources in the experiment reside at the same host so that true parallel execution plans do not exist. However, the join of the two source classes at each information source can be executed remotely, which lowers the plan cost, and PbR can benefit from this situation.



(a) Planning Time

(b) Plan Quality

(c) Plan Quality (Detail)

(d) Plan Size

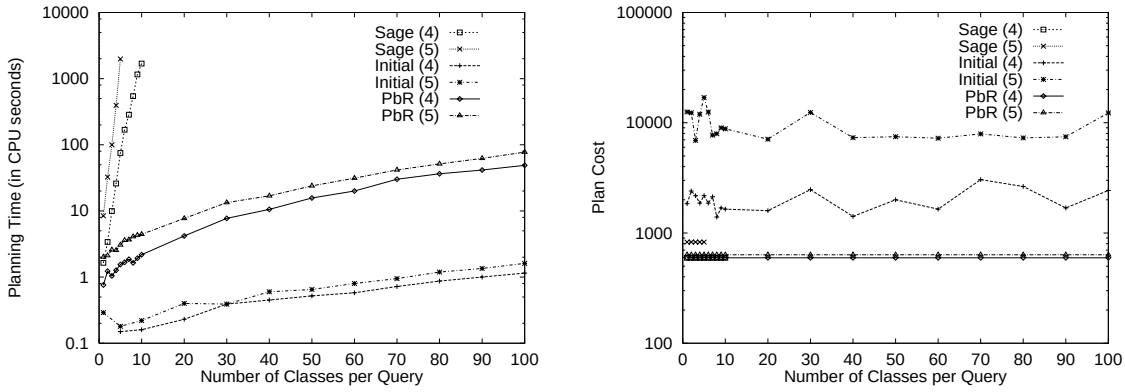
Figure 4.16: Experimental Results: Distributed Query Planning

domain class from 1 to 100. That is, there are up to 500 information sources for the largest model. PbR used a first improvement gradient descent search strategy with three random restarts and the `Join-Swap` and `Source-Swap` rules defined in Figures 4.10 and 4.7 respectively.

Figure 4.17 (a) shows the planning time for PbR, Sage, and Initial for queries involving four and five domain classes (queries `q4` and `q5` in Figure 4.15). Sage can solve the five-class queries in domains with up to five sources per class and the four-class queries up to ten sources per class within its 200,000 nodes search limit. PbR scales up much better solving all the tested queries regardless of the increasingly large number of alternative sources for each domain class.

Figure 4.17 (b) shows the planning time for PbR, Sage, and Initial. In order to control for quality we assigned the same cost of access to all sources relevant to a given domain class. However, the cost increases for each different domain class. Thus, we know that the cost of the optimal plan is constant (depending essentially on the join order). Even with this very simple cost model, the sheer number of alternatives causes a combinatorial explosion in the best-first search used of Sage.

PbR achieves this optimal cost regardless of the increasing complexity of the search space and the very high cost of the initial plans, which is over an order of magnitude higher than the optimal cost. The plan size is constant at 8 operators for the four-class queries and at 10 operators for the five-class queries.



(a) Planning Time (b) Plan Quality

Figure 4.17: Experimental Results: Scaling Alternative Sources

4.4.3 Scaling the Size of the Integration Axioms

The third collection of experiments analyzes the effect of complex integration axioms in the efficiency and quality of planning in PbR. We designed a set of parameterized integration models in which we can control the size of the integration axioms for each domain class as well as the number of alternative (and structurally different) integration axioms for each domain class.

As we mentioned before, these integration axioms are compiled automatically from the domain and source models (cf. [?]). The resulting integration axioms are shown in Figure 4.18. In the axioms, a domain class has the `dc` suffix (e.g. `Adc`, `Bdc`, ...), a source class contains `sc` (e.g. `Asc01`, `Bsc11`, ...), the attributes named `ki` are keys, and the attributes named `ri` are non-key attributes. In this experiment each domain class has the same axiom structure. All the axioms are conjunctive formulas. The length of the conjunction is one of the parameters in the experiment. For example, consider domain class `Adc` in Figure 4.18. The first axiom is a conjunction of source classes (`Asc00`, `Asc01`, `Asc02`, ...) joining in the key attribute `k0`. Notice that each source class uniquely produces an attribute (i.e., `Asc00` is the only source class to produce `r0`, only `Asc01` produces `r1`, etc.). This is used to control which source classes are used in the axiom by just requesting the corresponding attributes in a query. The second axiom for `Adc` is structurally identical but the axiom body joins on the key `k1`. Note that except for the keys (`k0`, `k1`, ...) the returned attributes of both axiom are the same. Therefore for any query that requests any combination of attributes of a class not involving the `ki` all the axioms are relevant. In other words, each of the axioms for a domain class is a valid alternative to answer those queries. The number of alternative axioms for each given class is another parameter in the experiment.

Another characteristic of this experiment is that all source classes with the same numerical suffix are in located in the same information source. For example, `Asc00`, `Bsc00`, ..., are all located on one information source `db00`. These information sources are capable of performing complex queries. Therefore the space of valid query plans contains plans in which complex queries are evaluated remotely at a source. For example, `Asc00(k0 r0 rc)` $\wedge$ `Bsc00(k0 r0 rc)` $\wedge$ ... can be executed at `db00`.

```
Adc(k0 rc r0 r1 ...) ⇔ Asc00(k0 r0 rc) ∧ Asc01(k0 r1) ∧ Asc02(k0 r2) ∧ ...
Adc(k1 rc r0 r1 ...) ⇔ Asc10(k1 r0 rc) ∧ Asc11(k1 r1) ∧ Asc12(k1 r2) ∧ ...
    ⋮
Bdc(k0 rc r0 r1 ...) ⇔ Bsc00(k0 r0 rc) ∧ Bsc01(k0 r1) ∧ Bsc02(k0 r2) ∧ ...
Bdc(k1 rc r0 r1 ...) ⇔ Bsc10(k1 r0 rc) ∧ Bsc11(k1 r1) ∧ Bsc12(k1 r2) ∧ ...
    ⋮
```

Figure 4.18: Parameterized Integration Axioms

In this experiment we tested conjunctive star queries. Each query joins the domain classes on a single non-key attribute (`rc`). For example, consider the following test query that involves two domain concepts (`Adc` and `Bdc`):

```
q(rc r0 r1 r2) :- Adc(rc r0 r1 r2) ∧ Bdc(rc br0 br1 br2)
```

As the domain concepts Adc and Bdc in this query ask for attributes `r0`, `r1`, and `r2`, the required axioms must have 3 conjuncts. A possible source-level expansion of this domain query is:

```
q(rc r0 r1 r2) :- Asc00(k0 r0 rc) ∧ Asc01(k0 r1) ∧ Asc02(k0 r2) ∧
                  Bsc10(k1 br0 rc) ∧ Bsc11(k1 br1) ∧ Bsc12(k1 br2)
```

We scaled the number of alternative axioms per domain class from 1 to 5, the length of each individual axiom from 1 to 5 conjuncts, and we tested with conjunctive queries involving from 1 to 5 domain classes. Note that the real query size depends on the length of the axioms. For example, a query involving 5 domain classes with an axiom size of 5 source classes unfolds into an equivalent retrievable query involving 25 source classes. We generated a data cube of 125 points along these three dimensions on which we measured planning time and plan quality for Sage, Initial, and PbR.

PbR used the `Join-Swap` and the `Remote-Join-Eval` rules defined in Figures 4.10 and 4.8 in addition to the rules derived from the relevant integration axioms for each query. The number of relevant integration axioms is the number of domain concepts in the query times the number of alternative integration axioms. For example, for a query with 5 domain concepts and 5 alternative integration axioms the system applies 27 rules at each rewriting phase. PbR used a first improvement search strategy with five restarts.

The results appear in Figures 4.19 and 4.20. In these graphs PbR data appears with dashed lines and Sage data with solid lines. In the graphs Sage and PbR use the same dot marker to denote the same experiment parameters. The abbreviation `qn` in the graphs denotes the query size, `aax`

the number of alternative axioms, and `axl` the axiom size (length). When the Sage lines stop, it means that Sage could not solve more complex problems within its search limit of 100000 nodes or a 1000 CPU seconds time limit — whichever comes first. Sage could not solve many queries as we scaled the complexity of the query planning problem.

Figure 4.19 shows the evolution of planning time as we scale several dimensions that affect the complexity of query planning. The first dimension is query size. Each of the five graphs in the Figure 4.19 represents the results for a fixed (domain) query size. The second dimension is the length of the integration axioms. In each graph we show the length of the relevant integration axioms on the x-axis and the planning time in CPU seconds for PbR and Sage in the y-axis (using a logarithmic scale). The third dimension is the number of alternative axioms. In each graph there are up to 5 lines per planner parameterized by the number of alternative axioms.

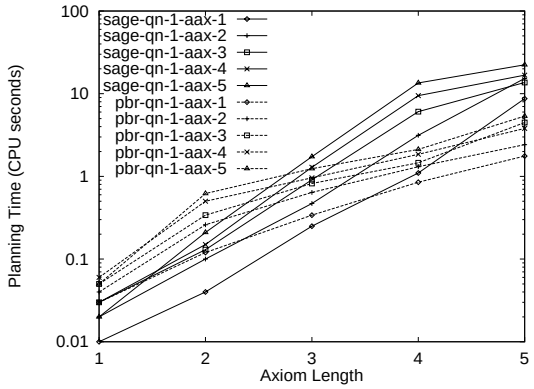
Consider Figure 4.19 (b) which shows the results for queries with two domain classes. We can draw the following conclusions. First, PbR scales much better than Sage with the length of the integration axioms. The graph shows how the PbR lines scale with a very low slope. Sage can solve only queries with an axiom length up to three conjuncts. And that only up to three alternative axioms. Second, PbR is not strongly affected by the number of alternative axioms for each given query size and axiom length. This is reflected in the graph by having all the PbR lines stay closely together as the axiom length increases. On the other hand, Sage is severely affected by the number of alternative axioms. This can be readily seen on the increasingly diverging lines at axiom length 3 and the disappearance of data points of Sage for queries involving more than 4 alternative axioms.

The conclusions we derive from the analysis of Figure 4.19 (b) hold for all the rest of the planning time data in Figure 4.19, namely:

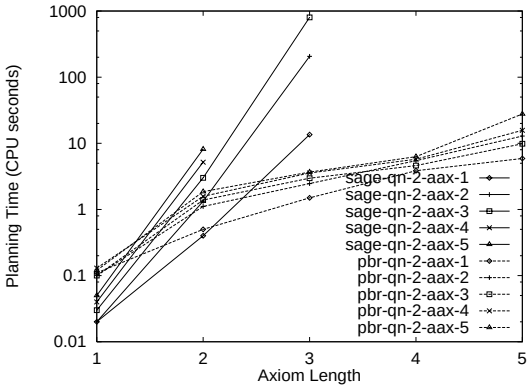
- PbR scales gracefully with query and axiom size
- PbR is not strongly affected by the number of alternative axioms

Consider the whole of Figure 4.19. As we increase the domain query size from 1 to 5, parts (a) to (e) of the figure, Sage is able to solve fewer and fewer queries, disappearing from the graphs until it only solves the queries involving axioms with one conjunct at query size 5 within the given time limit (Sage points are on the y-axis). Moreover, the number of alternative axioms also contributes to the performance degradation of the best-first search procedure of Sage. This is easily seen from the increasing slope of the curves for a fixed axiom length and number of alternative axioms as we increase the query size. For example, see the evolution of the lines labeled sage-qn-2-aax1, sage-qn-3-aax1, and sage-qn-4-aax1 in graphs (b), (c) and (d) of the figure. On the other hand, PbR scales well and solves all queries efficiently.

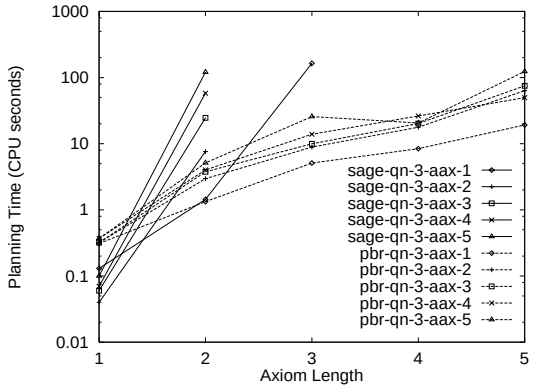
Figure 4.20 shows the evolution of plan cost along the same dimensions. Fortunately, the planning efficiency of PbR is not achieved by decreasing the quality of the query plans. For the increasing query size, axiom length, and number of alternative axioms, PbR produces good quality plans. The quality of PbR is comparable to that of Sage in the range of problems solvable by Sage and beyond it scales gracefully. Actually PbR produces better plans because by exploring the space of bushy trees it can generate more parallel plans.



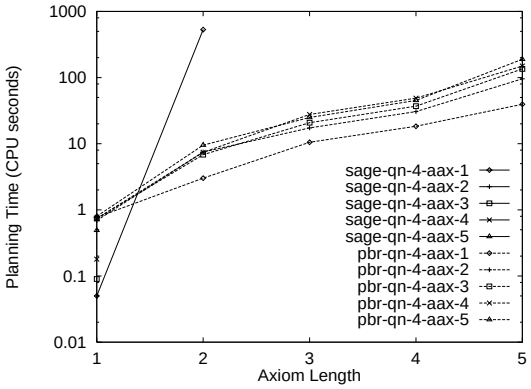
(a) Domain Query Size = 1



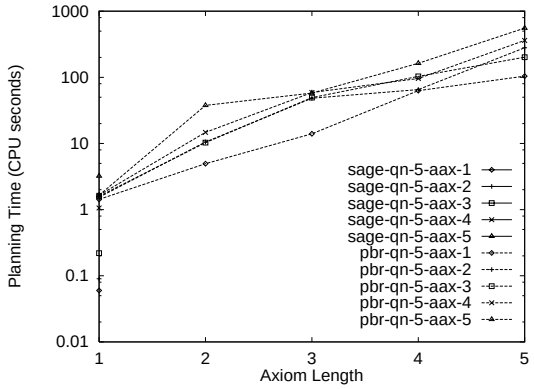
(b) Domain Query Size = 2



(c) Domain Query Size = 3

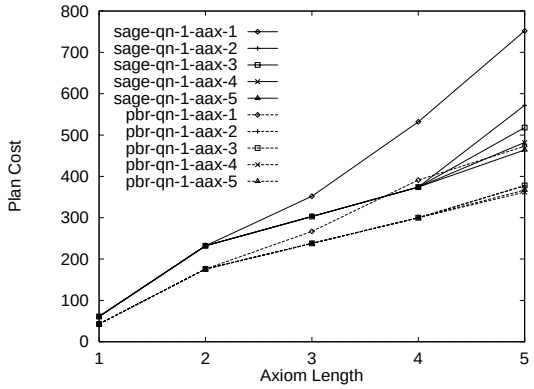


(d) Domain Query Size = 4

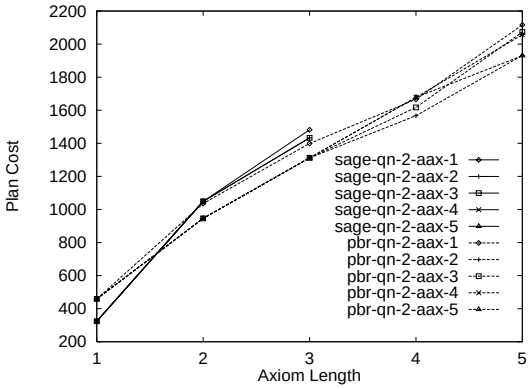


(e) Domain Query Size = 5

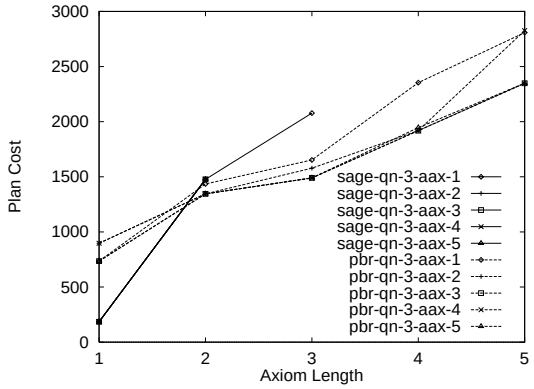
Figure 4.19: Experimental Results: Complex Axioms, Planning Time



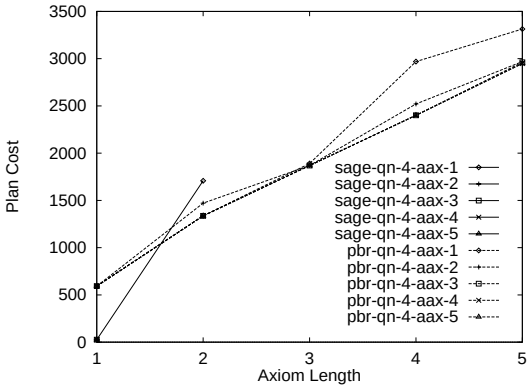
(a) Domain Query Size = 1



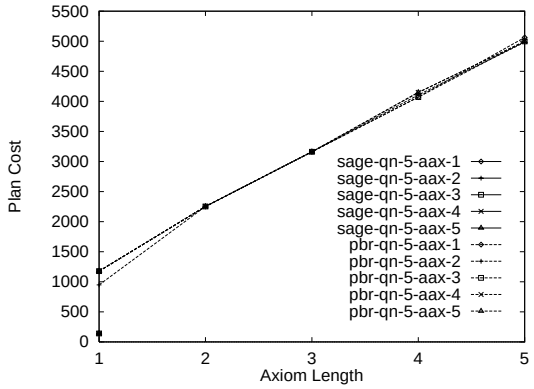
(b) Domain Query Size = 2



(c) Domain Query Size = 3



(d) Domain Query Size = 4



(e) Domain Query Size = 5

Figure 4.20: Experimental Results: Complex Axioms, Plan Cost

4.5 Advantages of PbR for Query Planning

Query planning in mediators presents particular challenges for planning technology. First, it is a highly combinatorial problem, where complex queries have to be composed from the relevant sources among hundreds of available information sources. Second, query plans often have to be produced rapidly. Third, finding any valid plan is not enough, plan quality is also critical. Finally, mediators need to be easily extensible in order to incorporate new sources and new capabilities, such as replanning after failures and information gathering actions.

The Planning by Rewriting paradigm is designed to address planning efficiency and plan quality, while providing the benefits of domain-independence. Its characteristics make it especially well-suited for query planning. First, our PbR-based query planner is *scalable*. As shown in the results section, our query planner scales gracefully to hundreds of sources and large query plans. In spite of the complexity of query planning our system produces good quality plans.

Second, an important advantage of PbR is its *anytime* nature, which allows it to trade off planning effort and plan quality. For example, a typical quality metric in query planning is the plan execution time. It may not make sense to keep planning if the cost of the current plan is small enough, even if a cheaper one could be found.

Third, PbR provides a *declarative domain-independent* framework that is easier to understand, maintain and extend than traditional query optimizers. Different query planning domains can be conveniently specified, for example, for different data models such as relational and object-oriented. A general planning architecture fosters *reuse* in the domain specifications and the search methods. For example, the specification of the join operator translates straightforwardly from a relational to an object-oriented model. Likewise, search methods can be implemented once for the general planner and the most appropriate configuration chosen for each particular domain. The uniform and principled specification of the planner facilitates its *extension* with new capabilities, such as learning mechanisms or interleaving planning and execution.

Finally, the generality of the PbR framework has allowed the design of a *novel combination of cost-based query optimization and source selection*. In previous work these two types of query processing had been performed in different stages. First, the mediator would find the set of all retrievable queries, that is, all the queries which are equivalent to the user query but only use source terms. Then, for each of these retrievable queries, the traditional query optimization problem remains. The final plan would be the best among the optimizations of the retrievable queries. Given that both finding retrievable queries and traditional query optimization are both combinatorial, the two-stage approach of previous work cannot scale. The problem is particularly acute in domains in which there exist many alternative sources of information, such as is the case in the WWW. PbR performs both optimizations in a single search process. By using local search techniques the combined optimization can be performed efficiently. Moreover, PbR supports an anytime behavior while the two-stage approach cannot.

Chapter 5

Other Application Domains

The last section showed the detailed application of PbR to the query planning domain. In this section we show the broad applicability of Planning by Rewriting by analyzing some additional domains with different characteristics. In particular, we consider the process manufacturing domain of [Minton, 1988b] and the Blocks World domain that we used in the examples throughout the thesis.

5.1 Manufacturing Process Planning

The task in the manufacturing process planning domain is to find a plan to manufacture a set of parts. We implemented a PbR translation of the domain specification in [Minton, 1988b]. This domain contains a variety of machines, such as a lathe, punch, spray painter, welder, etc, for a total of ten machining operations. The operator specification is shown in Figures 5.1, 5.2, and 5.3. The features of each part are described in the by a set of predicates, such as `temperature`, `painted`, `has-hole`, etc. These features are changed by the operators. Other predicates in the state, such as `has-clamp`, `is-drillable`, etc, are set in the initial state of each problem.

As an example of the behavior of an operator, consider the `polish` operator in Figure 5.1. It requires the part to manufacture to be cold and that the polisher has a clamp to secure the part to the machine. The effect of applying this operator its to leave the surface of the part polished. The universally quantified effect is an idiom we used to enforce that the `surface-condition` of an part has a single value. Other features of a part can be multivalued, for example, a part can have several holes. Note how the `drill-press` and the `punch` operators in Figure 5.2 do not prevent several `has-hole` conditions from being asserted on the same part. Other interesting operators are `weld` and `bolt`. These operators join two parts in a particular orientation to form a new part. No further operations can be performed on the separate parts once they have been joined.

In this domain all of the machining operations are assumed to take unit time. The machines and the objects (parts) are modeled as resources in order to enforce that only one part can be placed on a machine at a time and that a machine can only operate on a single part at a time

(except `bolt` and `weld` that operate on two parts simultaneously). We take the schedule length, the time to manufacture `all` parts, as the measure of plan cost.

We have already shown some of the types of rewriting rules for this domain in Figures 3.4 and 3.7. The set of rules that we used for our experiments are shown in Figures 5.4 and 5.5. The rules in Figure 5.4 are quite straightforward once one becomes familiar with this domain. The first two rules explore the space of alternative orderings originated by resource conflicts. The `machine-swap` rule allows the system to explore the possible orderings of operations that require the same machine. This rule finds two consecutive operations on the same machine and swaps their order. Similarly, the rule `object-swap` allows the system to explore the orderings of the operations on the same object. These two rules use the interpreted predicate `adjacent-in-critical-path` to focus the attention on the steps that contribute to our cost function. `Adjacent-in-critical-path` checks if two steps are consecutive along one of the critical paths of an schedule. A critical path is a sequence of steps that take the longest time to accomplish. In other words, a critical path is one of the sequences of steps that determine the schedule length.

The remaining rules of Figure 5.4 exchange operators that are equivalent with respect to achieving some effects. For example, rules `IP-by-SP` and `SP-by-IP` propose the exchange of `immersion-paint` and `spray-paint` operators. By examining the operator definitions in Figure 5.2, it can be readily noticed that both operators change the value of the `painted` predicate. Similarly, `PU-by-DP` and `DP-by-PU` exchange `drill-press` and `punch` operators, which produce the `has-hole` predicate. Finally, `roll-by-lathe` and `lathe-by-roll` exchange `roll` and `lathe` operators as they both can make parts cylindrical. To focus the search on the most promising exchanges these rules only match operators in the critical path (by means of the interpreted predicate `in-critical-path`).

The rewriting rules in Figure 5.5 and 5.6 are more sophisticated. The first rule, `lathe+SP-by-SP`, takes care of an undesirable effect of the simple depth-first search used by our initial plan generator. In this domain, in order to spray paint a part, the part must have a regular shape. Being cylindrical is a regular shape, therefore the initial planner may decide to make the part cylindrical by lathing it in order to paint it (!). However, this may not be necessary as the part may already have a regular shape (for example, it can be rectangular which is also a regular shape). Thus, the `lathe+SP-by-SP` substitutes the pair `spray-paint` and `lathe` by a single `spray-paint` operation. The supporting `regular-shapes` interpreted predicate just enumerates which are the regular shapes. These rules are partially specified and are not guaranteed to always produce a valid rewriting. Nevertheless, they are often successful in producing plans of lower cost.

The remaining rules in Figures 5.5 and 5.6 explore bolting two parts using bolts of different size if fewer operations may be needed for the plan. We developed these rules by analyzing differences in the quality of the optimal plans and the rewritten plans. For example, consider the `both-providers-diff-bolt` rule. This rule states that if the parts to be bolted already have compatible holes in them, it is better to reuse those operators that produced the holes. The initial plan generator may have drilled (or punched) holes which only purpose was to bolt the parts. However, the goal of the problem may already require some holes to be performed on the parts to

```
(define (operator POLISH)
    :parameters (?x)
    :resources ((machine POLISHER) (is-object ?x))
    :precondition (:and (is-object ?x)
                        (temperature ?x COLD)
                        (has-clamp POLISHER))
    :effect (:and (:forall (?oldsurf) (:when (:neq ?oldsurf POLISHED)
                                             (:not (surface-condition ?x ?oldsurf))))
                  (surface-condition ?x POLISHED)))

(define (operator ROLL)
    :parameters (?x)
    :resources ((machine ROLLER) (is-object ?x))
    :precondition (is-object ?x)
    :effect (:and (:forall (?color) (:not (painted ?x ?color)))
                  (:forall (?shape) (:when (:neq ?shape CYLINDRICAL)
                                           (:not (shape ?x ?shape))))
                  (:forall (?temp) (:when (:neq ?temp HOT)
                                          (:not (temperature ?x ?temp))))
                  (:forall (?cond) (:not (surface-condition ?x ?cond)))
                  (:forall (?width ?orientation)
                           (:not (has-hole ?x ?width ?orientation)))
                 (temperature ?x HOT)
                 (shape ?x CYLINDRICAL)))

(define (operator LATHE)
    :parameters (?x)
    :resources ((machine LATHE) (is-object ?x))
    :precondition (is-object ?x)
    :effect (:and (:forall (?color) (:not (painted ?x ?color)))
                  (:forall (?shape) (:when (:neq ?shape CYLINDRICAL)
                                           (:not (shape ?x ?shape))))
                  (:forall (?cond) (:when (:neq ?cond ROUGH)
                                          (:not (surface-condition ?x ?cond))))
                  (surface-condition ?x ROUGH)
                  (shape ?x CYLINDRICAL)))

(define (operator GRIND)
    :parameters (?x)
    :resources ((machine GRINDER) (is-object ?x))
    :precondition (is-object ?x)
    :effect (:and (:forall (?color) (:not (painted ?x ?color)))
                  (:forall (?cond) (:when (:neq ?cond SMOOTH)
                                          (:not (surface-condition ?x ?cond))))
                  (surface-condition ?x SMOOTH)))
```

Figure 5.1: Operators for Manufacturing Process Planning (I)

```
(define (operator PUNCH)
    :parameters (?x ?width ?orientation)
    :resources ((machine PUNCH) (is-object ?x))
    :precondition (:and (is-object ?x)
                        (is-punchable ?x ?width ?orientation)
                        (has-clamp PUNCH))
    :effect (:and (:forall (?cond)(:when (:neq ?cond ROUGH)
                                         (:not (surface-condition ?x ?cond))))
                  (surface-condition ?x ROUGH)
                  (has-hole ?x ?width ?orientation)))

(define (operator DRILL-PRESS)
    :parameters (?x ?width ?orientation)
    :resources ((machine DRILL-PRESS) (is-object ?x))
    :precondition (:and (is-object ?x)
                        (is-drillable ?x ?orientation)
                        (have-bit ?width))
    :effect (has-hole ?x ?width ?orientation))

(define (operator SPRAY-PAINT)
    :parameters (?x ?color ?shape)
    :resources ((machine SPRAY-PAINTER) (is-object ?x))
    :precondition (:and (is-object ?x)
                        (sprayable ?color)
                        (temperature ?x COLD)
                        (regular-shape ?shape)
                        (shape ?x ?shape)
                        (has-clamp SPRAY-PAINTER))
    :effect (painted ?x ?color))

(define (operator IMMERSION-PAINT)
    :parameters (?x ?color)
    :resources ((machine IMMERSION-PAINTER) (is-object ?x))
    :precondition (:and (is-object ?x)
                        (have-paint-for-immersion ?color))
    :effect (painted ?x ?color))
```

Figure 5.2: Operators for Manufacturing Process Planning (II)

```
(define (operator WELD)
    :parameters (?x ?y ?new-object ?orientation)
    :resources ((machine WELDER) (is-object ?x) (is-object ?y))
    :precondition (:and (is-object ?x)
                        (is-object ?y)
                        (composite-object ?new-object ?orientation ?x ?y)
                        (can-be-welded ?x ?y ?orientation))
    :effect (:and (temperature ?new-object HOT)
                  (joined ?x ?y ?orientation)
                  (:not (is-object ?x))
                  (:not (is-object ?y))))

(define (operator BOLT)
    :parameters (?x ?y ?new-object ?orientation ?width)
    :resources ((machine BOLTER) (is-object ?x) (is-object ?y))
    :precondition (:and (is-object ?x)
                        (is-object ?y)
                        (composite-object ?new-object ?orientation ?x ?y)
                        (has-hole ?x ?width ?orientation)
                        (has-hole ?y ?width ?orientation)
                        (bolt-width ?width)
                        (can-be-bolted ?x ?y ?orientation))
    :effect (:and (:not (is-object ?x))
                  (:not (is-object ?y))
                  (joined ?x ?y ?orientation)))
```

Figure 5.3: Operators for Manufacturing Process Planning (III)

be joined. Reusing the available holes produces a more economical plan. The rules `has-hole-x--diff-bolt-add-PU`, `has-hole-x-diff-bolt-add-DP`, `has-hole-y-diff-bolt-add-PU`, and `has--hole-y-diff-bolt-add-DP` address the cases in which only one of the holes can be reused, and thus an additional `punch` or `drill-press` operation needs to be added.

As an illustration of the rewriting process in the manufacturing domain, consider the Figure 5.7. The plan at the top of the figure is the result of a simple initial plan generator that solves each part independently and concatenates the subplans. Although presumably such plan is generated efficiently, it is of poor quality. It requires 6 time-steps to manufacture all parts. The figure shows the application of two rewriting rules `machine-swap` and `IP-by-SP` that improve the quality of this plan. The operators matched by the rule antecedent are shown in *italics*. The operators introduced in the rule consequent are shown in **bold**. First, the `machine-swap` rules reorders the punching operations of parts `A` and `B`. This breaks the long critical path that resulted from the simple concatenation of their respective subplans. The schedule length improves from 6 to 4 time-steps. Still, the three parts `A`, `B`, and `C` use the same painting operation (`immersion-paint`). As the immersion-painter can only process one piece at a time, the three operations must be done serially. Fortunately, in our domain there is another painting operation: `spray-paint`. The `IP-by-SP` rule takes advantage of this fact and substitutes an `immersion-paint` operation on part `B` by a `spray-paint` operation. This further parallelizes the plan obtaining a schedule length of 3 time-steps that is the optimal for this plan.

```
(define-rule :name machine-swap
  :if (:operators ((?n1 (machine ?x) :resource)
                   (?n2 (machine ?x) :resource))
       :links ((?n1 :threat ?n2))
       :constraints ((adjacent-in-critical-path ?n1 ?n2)))
  :replace (:links (?n1 ?n2))
  :with (:links (?n2 ?n1)))

(define-rule :name object-swap
  :if (:operators ((?n1 (is-object ?x) :resource)
                   (?n2 (is-object ?x) :resource))
       :links ((?n1 :threat ?n2))
       :constraints ((adjacent-in-critical-path ?n1 ?n2)))
  :replace (:links (?n1 ?n2))
  :with (:links (?n2 ?n1)))

(define-rule :name IP-by-SP
  :if (:operators ((?n1 (immersion-paint ?x ?color)))
       :constraints ((regular-shapes ?shape)
                     (in-critical-path ?n1)))
  :replace (:operators (?n1))
  :with (:operators (?n2 (spray-paint ?x ?color ?shape))))

(define-rule :name SP-by-IP
  :if (:operators ((?n1 (spray-paint ?x ?color ?shape)))
       :constraints ((in-critical-path ?n1)))
  :replace (:operators (?n1))
  :with (:operators (?n2 (immersion-paint ?x ?color))))

(define-rule :name PU-by-DP
  :if (:operators ((?n1 (punch ?x ?width ?orientation)))
       :constraints ((in-critical-path ?n1)))
  :replace (:operators (?n1))
  :with (:operators (?n2 (drill-press ?x ?width ?orientation))))

(define-rule :name DP-by-PU
  :if (:operators ((?n1 (drill-press ?x ?width ?orientation)))
       :constraints ((in-critical-path ?n1)))
  :replace (:operators (?n1))
  :with (:operators (?n2 (punch ?x ?width ?orientation))))

(define-rule :name roll-by-lathe
  :if (:operators ((?n1 (roll ?x)))
       :constraints ((in-critical-path ?n1)))
  :replace (:operators (?n1))
  :with (:operators (?n2 (lathe ?x))))

(define-rule :name lathe-by-roll
  :if (:operators ((?n1 (lathe ?x)))
       :constraints ((in-critical-path ?n1)))
  :replace (:operators (?n1))
  :with (:operators (?n2 (roll ?x))))
```

Figure 5.4: Rewriting Rules for Manufacturing Process Planning (I)

```
(define-rule :name lathe+SP-by-SP
  :if (:operators ((?n1 (lathe ?x))
                   (?n2 (spray-paint ?x ?color ?shape1)))
       :constraints ((regular-shapes ?shape2)))
  :replace (:operators (?n1 ?n2))
  :with (:operators (?n3 (spray-paint ?x ?color ?shape2))))

(define-rule :name both-providers-diff-bolt
  :if (:operators ((?n3 (bolt ?x ?y ?z ?o ?w1)))
       :links ((?n1 (has-hole ?x ?w1 ?o) ?n3)
               (?n2 (has-hole ?y ?w1 ?o) ?n3)
               (?n4 (has-hole ?x ?w2 ?o) ?n5)
               (?n6 (has-hole ?y ?w2 ?o) ?n7))
       :constraints ((:neq ?w1 ?w2)))
  :replace (:operators (?n1 ?n2 ?n3))
  :with (:operators ((?n8 (bolt ?x ?y ?z ?o ?w2)))
         :links ((?n4 (has-hole ?x ?w2 ?o) ?n8)
                 (?n6 (has-hole ?y ?w2 ?o) ?n8))))

(define-rule :name has-hole-x-diff-bolt-add-PU
  :if (:operators ((?n3 (bolt ?x ?y ?z ?o ?w1)))
       :links ((?n1 (has-hole ?x ?w1 ?o) ?n3)
               (?n2 (has-hole ?y ?w1 ?o) ?n3)
               (?n4 (has-hole ?x ?w2 ?o) ?n5))
       :constraints ((:neq ?w1 ?w2)))
  :replace (:operators (?n1 ?n2 ?n3))
  :with (:operators ((?n8 (bolt ?x ?y ?z ?o ?w2))
                     (?n6 (punch ?y ?w2 ?o)))
         :links ((?n4 (has-hole ?x ?w2 ?o) ?n8)
                 (?n6 (has-hole ?y ?w2 ?o) ?n8))))

(define-rule :name has-hole-x-diff-bolt-add-DP
  :if (:operators ((?n3 (bolt ?x ?y ?z ?o ?w1)))
       :links ((?n1 (has-hole ?x ?w1 ?o) ?n3)
               (?n2 (has-hole ?y ?w1 ?o) ?n3)
               (?n4 (has-hole ?x ?w2 ?o) ?n5))
       :constraints ((:neq ?w1 ?w2)))
  :replace (:operators (?n1 ?n2 ?n3))
  :with (:operators ((?n8 (bolt ?x ?y ?z ?o ?w2))
                     (?n6 (drill-press ?y ?w2 ?o)))
         :links ((?n4 (has-hole ?x ?w2 ?o) ?n8)
                 (?n6 (has-hole ?y ?w2 ?o) ?n8))))
```

Figure 5.5: Rewriting Rules for Manufacturing Process Planning (II)

```
(define-rule :name has-hole-y-diff-bolt-add-PU
  :if (:operators ((?n3 (bolt ?x ?y ?z ?o ?w1)))
       :links ((?n1 (has-hole ?x ?w1 ?o) ?n3)
               (?n2 (has-hole ?y ?w1 ?o) ?n3)
                           (?n6 (has-hole ?y ?w2 ?o) ?n7))
                     :constraints ((:neq ?w1 ?w2)))
  :replace (:operators (?n1 ?n2 ?n3))
  :with (:operators ((?n8 (bolt ?x ?y ?z ?o ?w2))
                     (?n4 (punch ?x ?w2 ?o)))
         :links ((?n4 (has-hole ?x ?w2 ?o) ?n8)
                 (?n6 (has-hole ?y ?w2 ?o) ?n8))))

(define-rule :name has-hole-y-diff-bolt-add-DP
  :if (:operators ((?n3 (bolt ?x ?y ?z ?o ?w1)))
       :links ((?n1 (has-hole ?x ?w1 ?o) ?n3)
               (?n2 (has-hole ?y ?w1 ?o) ?n3)
               (?n6 (has-hole ?y ?w2 ?o) ?n7))
       :constraints ((:neq ?w1 ?w2)))
  :replace (:operators (?n1 ?n2 ?n3))
  :with (:operators ((?n8 (bolt ?x ?y ?z ?o ?w2))
                     (?n4 (drill-press ?x ?w2 ?o)))
         :links ((?n4 (has-hole ?x ?w2 ?o) ?n8)
                 (?n6 (has-hole ?y ?w2 ?o) ?n8))))
```

Figure 5.6: Rewriting Rules for Manufacturing Process Planning (III)

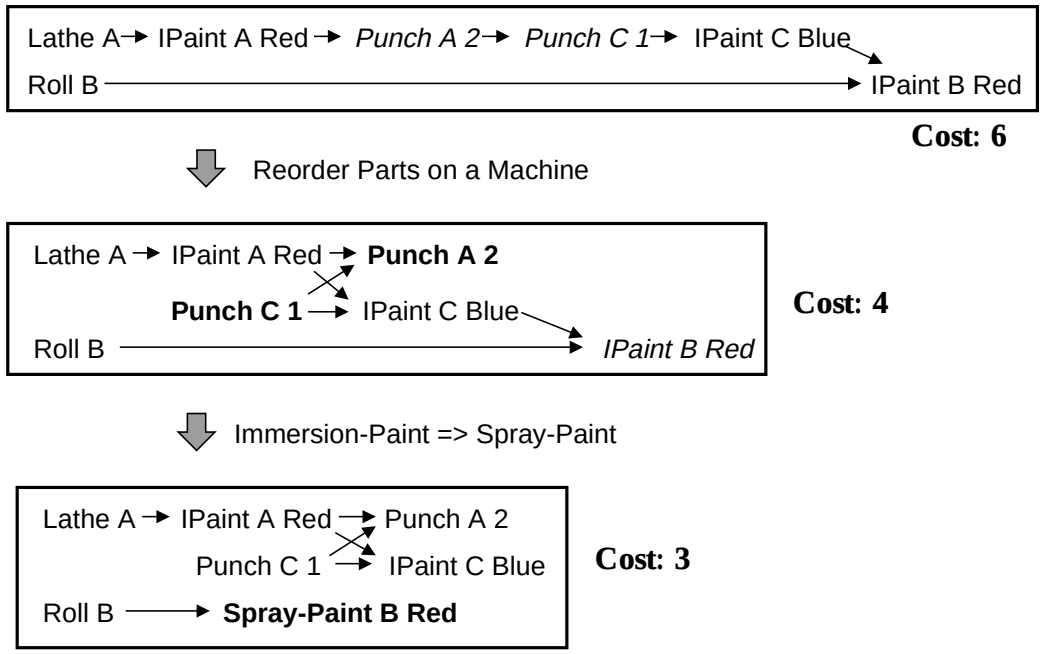


Figure 5.7: Rewriting in the Manufacturing Domain

In this experiment we compare four planners: IPP, Initial, and two configurations of PbR:

IPP: This is one of the most efficient domain-independent planners [Koehler *et al.*, 1997] as was demonstrated in the planning competition held at the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98). IPP is an optimized re-implementation and extension of Graphplan [Blum and Furst, 1995]. IPP produces shortest parallel plans. For our manufacturing domain this is exactly the schedule length, the cost function that we are optimizing.[1]

Initial: The initial plan generator uses a divide and conquer heuristic in order to generate plans as fast as possible. First, it produces subplans for each part and for the `joined` goals independently. These subplans are generated by Sage using a depth-first search without any regard to plan cost. Then, it concatenates the subsequences of actions and merges them plans using the facilities of Section 3.1.2.

PbR: We present results for two configurations of PbR, that we will refer to as PbR-100 and PbR-300. Both configurations use a first improvement gradient search strategy with random walk on the cost plateaus. The rewriting rules used are those of Figures 5.4, 5.5, and 5.6. For each problem PbR starts its search from the plan generated by Initial. The two configurations differ only on how many total plateau plans are allowed. PbR-100 allows considering up to 100 plans that do not improve the cost without terminating the search. Similarly, PbR-300 allows 300 plateau plans. Note that the limit is across all plateaus encountered during the search for a problem, not for each plateau. For example, assume PbR reaches a plateau of cost 9, it may need to consider 25 plans before finding a lower cost plan, say of cost 7, then consider another 15 non-improving plans of cost 7, find an improving plan to cost 6, walk for 20 plans in the plateau of cost 6, improve to cost 5, and finally consider another 40 non-improving plans until it reaches the limit of 100 plateau which ends the search.

We tested each of the four systems on 200 problems, for machining 10 parts, ranging from 5 to 50 goals.[2] The goals are distributed randomly over the 10 parts. So, for the 50-goal problems,

[1] The operators for IPP are identical to those of PbR shown in Figures 5.1, 5.2, and 5.3, but we added *complementary* side effects to each operator to enforce the contention for machines and parts. Each operator has a side-effect `uses-`*machine*`()` for each *machine* its uses, and a side-effect `uses(?x)` for each object `?x` it operates upon. For example, the IPP specification of the `polish` operator is:

```
polish
:v ?x object
:p is-object(?x) temperature(?x COLD) has-clamp(POLISHER)
:e ADD surface-condition(?x POLISHED)
       uses-polisher() not uses-polisher() uses(?x) not uses(?x);
   ALL ?oldsurf surf-cond !eq(?oldsurf POLISHED) =>
       ADD not surface-condition(?x ?oldsurf).
```

[2] The problems for IPP, PbR, and Initial are, of course, identical. However, as IPP cannot generate new objects, we listed in the initial state of the IPP problems all of the possible `composite-objects`. To limit the number of these ground facts, the problems only join any given pair of parts once (thus there are 90 possible composite objects for the 10 parts in the problems).

there is an average of 5 goals per part. The results are shown in Figure 5.8. In this graphs each data point is the average of 20 problems for each given number of goals. There were 10 provably unsolvable problems. Initial and PbR solved all the 200 problems (or proved them unsolvable). IPP solved 65 problems in total: all problems at 5 and 10 goals, 19 at 15 goals, and 6 at 20 goals. IPP could not solve any problem with more than 20 goals under the 1000 CPU seconds time limit.

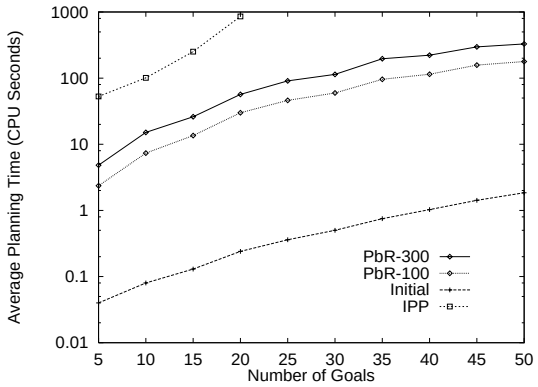
Figure 5.8 (a) shows the average time on the solvable problems for each problem set for the four planners. Figure 5.8 (b) shows the average schedule length for the problems solved by *all* the planners, that is, over the 65 problems solved by IPP up to 20 goals. The fastest planner is Initial, but it produces plans with a cost of about twice the optimal. IPP produces the optimal plans, but it cannot solve problems of more than 20 goals. The two configurations of PbR scale much better than IPP solving all problems and producing good quality plans. PbR-300 matches the optimal cost of the IPP plans, except in one problem (the reason for the difference is interesting and we explain it below). The faster PbR-100 also stays very close to the optimal.

Figure 5.8 (c) shows the average schedule length for the problems solved by *each* of the planners for the 50 goal range. The PbR configurations scale gracefully across this range improving considerably the cost of the plans generated by Initial. The additional exploration of PbR-300 allows it to improve the plans even further. The reason for the difference between PbR and IPP at the 20-goal complexity level is because the cost results for IPP are only for the 6 problems that it could solve, while the results for PbR and Initial are the average of all of the 20 problems. The problems solved by IPP are arguably the easiest and PbR matches their (optimal) cost as shown in Figure 5.8 (b). However, the average cost of for all problems at the 20-goals complexity level seems to be higher.

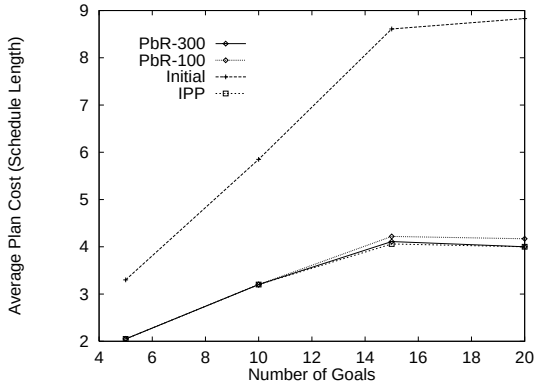
Figure 5.8 (d) shows the average number of operators in the plans for the problems solved by *all* three planners (up to 20 goals). Figure 5.8 (e) shows the average number of operators in the plans for the problems solved by *each* planner across the whole range of 50 problems. The plans generated by Initial use about 2-3 additional operators. Both PbR and IPP produce plans that requires fewer steps. Interestingly, IPP sometime produces plans than use more operations than PbR. IPP produces the shortest parallel plan, but not the one with the minimum number of steps. In particular, we observed that some of the IPP plans suffer from the same problem as Initial. IPP would also lathe a part in order to paint it, but as opposed to Initial it would only do so if it did not affect the optimal schedule length. Surprisingly, adding such additional steps in this domain may improve the schedule length, albeit in fairly rare situations. This was the case in the only problem in which IPP produced a better schedule than PbR-300.[3]

This experiment illustrates the flexibility of PbR in specifying complex rules for a planning domain. The results show the benefits of finding a suboptimal initial plan quickly and then efficiently transforming it to improve its quality.

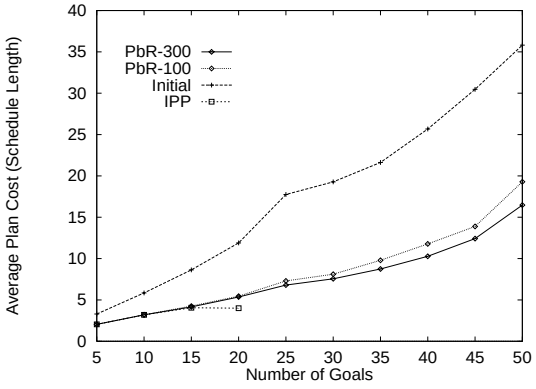
[3] We could have introduced a rewriting rule that substituted an `immersion-paint` operator by both a `lathe` and `spray-paint` operators for such cases. However, such rule is of very low utility (in the sense of [Minton, 1988b]). It expands the rewriting search space, adds to the cost of match, and during the random search provides some benefit very rarely.



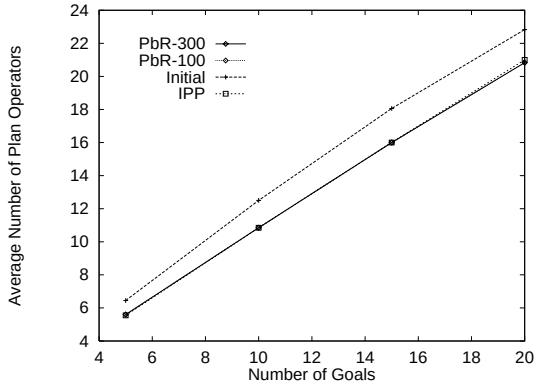
(a) Average Planning Time



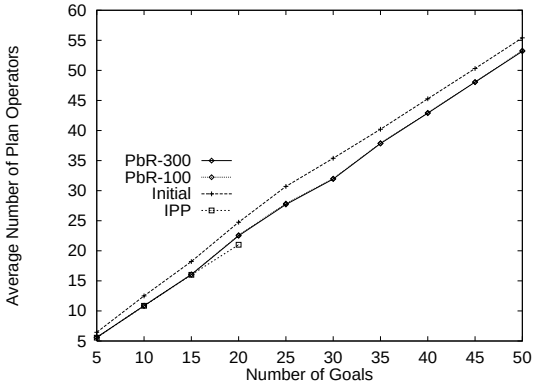
(b) Average Plan Cost (Problems Solved by All)



(c) Average Plan Cost (Problems Solved by Each)



(d) Number of Plan Operators (Problems Solved by All)



(e) Number of Plan Operators (Problems Solved by Each)

Figure 5.8: Experimental Results: Manufacturing Process Planning

5.2 Blocks World

We implemented a classical Blocks World domain with the two operators in Figure 2.1. This domain has two actions: `stack` that puts one block on top of another, and, `unstack` that places a block on the table to start a new tower. Plan quality in this domain is simply the number of steps. Optimal planning in this domain is NP-hard [Gupta and Nau, 1992]. However, it is trivial to generate a correct, but suboptimal, plan in linear time using the naive algorithm: put all blocks on the table and build the desired towers from the bottom up. We compare three planners on this domain:

IPP: In this experiment we used the GAM goal ordering heuristic [Koehler, 1998] that had been tested in simple Blocks World problems with very good scaling results.

Initial: This planner is a programmatic implementation of the naive algorithm using the facilities introduced in Section 3.1.2.

PbR: This configuration of PbR starts from the plan produced by Initial and uses the two plan rewriting rules shown in Figure 5.9 to optimize plan quality. PbR applies a first improvement strategy with only one run (no restarts).

```
(define-rule :name avoid-move-twice
  :if (:operators ((?n1 (unstack ?b1 ?b2))
                   (?n2 (stack ?b1 ?b3 Table)))
       :links (?n1 (on ?b1 Table) ?n2)
       :constraints ((possibly-adjacent ?n1 ?n2)
                     (:neq ?b2 ?b3)))
  :replace (:operators (?n1 ?n2))
  :with (:operators (?n3 (stack ?b1 ?b3 ?b2))))

(define-rule :name avoid-undo
  :if (:operators ((?n1 (unstack ?b1 ?b2))
                   (?n2 (stack ?b1 ?b2 Table)))
       :constraints ((possibly-adjacent ?n1 ?n2))
  :replace (:operators (?n1 ?n2))
  :with NIL))
```

Figure 5.9: Blocks World Rewriting Rules

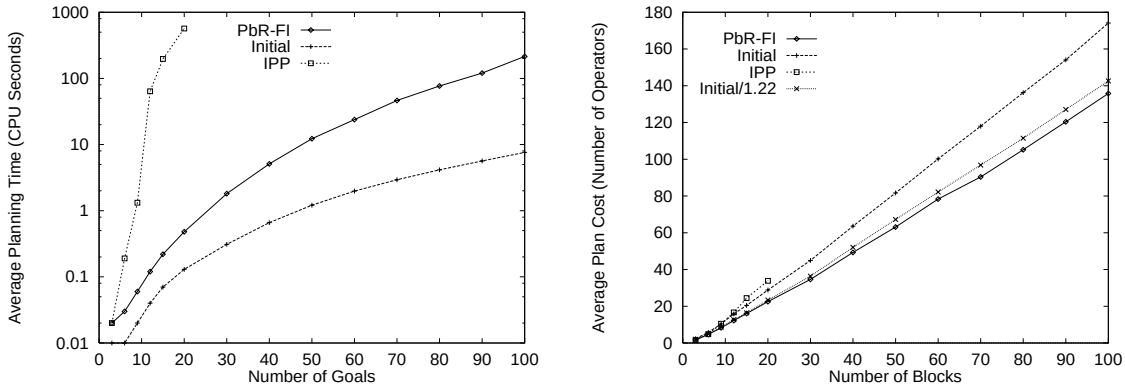
We generated random Blocks World problems scaling the number of blocks. The problem set consists of 25 random problems at 3, 6, 9, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90, and 100 blocks for a total of 350 problems. The problems may have multiple towers in the initial state and in the goal state.

Figure 5.10 (a) shows the average planning time of the 25 problems for each block quantity. IPP cannot solve problems with more than 20 blocks within the time limit of 1000 CPU seconds. The problem solving behavior of IPP was interesting. IPP either solved a given problem very fast

or it timed out. For example, it was able to solve 11 out of the 25 20-block problems under 100 seconds, but it timed out at 1000 seconds for the remaining 14 problems. This seems to be the typical behavior of complete search algorithms [Gomes *et al.*, 1998]. The local search of PbR allows it to scale much better and solve all the problems.

Figure 5.10 (b) shows the average plan cost as the number of blocks increases. PbR improves considerably the quality of the initial plans. The optimal quality is only known for very small problems, where PbR approximates it, but does not achieve it (we run Sage for problems of less than 9 blocks). For larger plans we do not know the optimal cost. However, [Slaney and Thiébaux, 1996] performed an extensive experimental analysis of Blocks World planning using a domain like ours. In their comparison among different approximation algorithms they found that our initial plan generator (unstack-stack) achieves empirically a quality around 1.22 the optimal for the range of problem sizes we have analyzed (Figure 7, page 1214). The value of our average initial plans divided by 1.22 suggests the quality of the optimal plans. The quality achieved by PbR is comparable with that value. In fact it is slightly better which may due to the relatively small number of problems tested (25 per block size) or to skew in our random problem generator. Interestingly the plans found by IPP are actually of low quality. This is due to the fact that IPP produces shortest parallel plans. That means that the plans can be constructed in the fewest time steps, but IPP may introduce more actions in each time step that could be required.

In summary, the experiments in this and the previous chapter show that across a variety of domains Planning by Rewriting scales to much larger problems than previous approaches to planning while still producing high-quality plans.



(a) Average Planning Time

(b) Average Plan Cost

Figure 5.10: Experimental Results: Blocks World, Scaling the Number of Blocks

Chapter 6

Related Work

In this chapter we review previous work related to the general Planning by Rewriting framework, as well as to the application of PbR on several domains, with an emphasis on query planning.

6.1 Foundations

In this section we review related work on several of the disciplines upon which the Planning by Rewriting framework builds, namely, classical AI planning, local search, and graph rewriting.

6.1.1 AI Planning

PbR is designed to find a balance among the requirements of planning efficiency, high quality plans, flexibility, and extensibility. A great amount of work on AI Planning has focussed on improving its average-case efficiency given that the general cases are computationally hard [Erol *et al.*, 1995]. One possibility is to incorporate domain knowledge in the form of search control rules. PbR rewriting rules can be seen as a form of "post facto" search control for plan optimization. Instead of guiding a generative planner towards the optimal solution, PbR improves the quality of a given complete plan. Some advantages of our approach is that quality can be better assessed in a complete plan and the anytime behavior.

Some systems automatically learn search control for a given planning domain or even specific problem instances. Minton [Minton, 1988b] shows how to deduce search control rules for a problem solver by applying explanation-based learning to traces of problem-solving traces. He also discusses the impact of the utility problem. The utility problem, simply stated, says that the (computational) benefits of using the additional knowledge must outweigh the cost of applying it. PbR plan rewriting rules also are subject to the utility problem. The quality improvement obtained by adding more rewriting rules to a PbR-based planner may not be worth the performance degradation. Another approach to automatically generating search control rules is by analyzing statically the operators in the planning domain [Etzioni, 1993]. Abstraction provides another form of search control. Knoblock [Knoblock, 1994a] presents a system that automatically learns abstraction hierarchies from a planning domain or a particular problem instance in order to speed up planning. PbR does

not currently learn the rewriting rules but we believe that similar learning methods can be applied. Finding the differences among a suboptimal and optimal plan may suggest rules to transform one into the other. Analyzing the planning operators and which combinations of operators are equivalent with respect to the achievement of some goals also can lead to automatically generating the rewriting rules.

Local search algorithms have also been used to improve planning efficiency although in a somewhat indirect way. Planning can be reduced to solving a series of propositional satisfiability problems [Kautz and Selman, 1992]. Therefore, Kautz and Selman used an efficient satisfiability testing algorithm based on local search to solve the SAT encodings of a planning problem [Kautz and Selman, 1996]. Their approach proved more efficient than specialized planning algorithms. We believe that the power of their approach stems from the use of local search. PbR directly applies local search on the plan structures, as opposed to translating it first to a larger propositional representation.

Although all these approaches do improve the efficiency of planning, they do not specifically address plan quality, or else they consider only very simple cost metrics (such as the number of steps). Quality-improving control rules are learned in [Pérez, 1996], but planning efficiency was not significantly improved. By exploiting domain-specific knowledge, conveniently expressed as plan rewriting rules, and the local search approach, we improve both plan efficiency and quality. Moreover, we provide an anytime algorithm while other approaches must run to completion.

6.1.2 Local search

Local search has a long tradition in combinatorial optimization [Aarts and Lenstra, 1997, Papadimitriou and Steiglitz, 1982]. Local improvement ideas have found application in many domains. Some of the general work most relevant for PbR is on constraint satisfaction, satisfiability testing, and heuristic search.

In constraint satisfaction, local search techniques have been able to solve problems orders of magnitude more complex than the respective complete (backtracking) approaches. Minton et al. [Minton *et al.*, 1990, Minton, 1992] developed a simple repair heuristic, min-conflicts, that could solve large constraint satisfaction and scheduling problems, such as the scheduling of operations in the Hubble Space Telescope. The min-conflicts heuristic just selects the variable value assignment that minimizes the number of constraints violated. This heuristic was used as the cost function of a gradient-descent search and also in an informed backtracking search.

In satisfiability testing a similar method, GSAT, was introduced by [Selman *et al.*, 1992]. GSAT solves hard satisfiability problems using local search where the repairs consist in changing the truth value of a randomly chosen variable. The cost function is the number of clauses satisfied by the current truth assignment. Their approach scales much better than the corresponding complete method (the Davis-Putnam procedure).

Our work is inspired by these approaches but there are several differences. First, PbR operates on complex graph structures — partial-order plans — as opposed to variable assignments. Second,

our repairs are declaratively specified and may be changed for each problem domain, as opposed to their general but fixed repair strategies. Third, PbR accepts arbitrary measures of quality not just constraint violations as in min-conflicts. GSAT is a satisfiability approach so it does not addresses cost optimization. Finally, and most importantly, PbR searches the space of solution plans, which are complete and sound, as opposed to the space of variable assignments which may be internally inconsistent.

Iterative repair ideas have also been used in heuristic search. In [Ratner and Pohl, 1986] a two-phase approach similar to PbR is presented. In the first phase, they find a initial valid sequence of operators using an approximate algorithm. In the second phase, they perform local search from that initial sequence. The cost function is the plan length. The local neighborhood is generated by identifying segments in the current solution sequence and attempting to optimize them. The repair consists of a heuristic search with the initial state being the beginning of the segment and the goal the end of the segment. If a shorter path is found, the original sequence is repaired by the replacing the old by the new shorter segment. A significant difference with PbR is that they are doing a state-space search, while PbR is doing a plan-space search. The least-committed partial-order nature of PbR allows it to optimize the plans in ways that cannot be achieved by optimizing linear subsequences.

6.1.3 Graph Rewriting

PbR builds on ideas from graph rewriting (see [Schürr, 1996a] for a survey). The plan rewriting rules in PbR are an extension to traditional graph rewriting rules. By taking advantage of the semantics of planning PbR introduces partially-specified plan rewriting rules, where the rules do not need to specify the completely detailed embedding of the consequent as in pure graph rewriting. Nevertheless, there are several techniques that can transfer from graph rewriting into Planning by Rewriting. In particular, [Dorr, 1995] defines an abstract machine for graph isomorphism and studies a set of conditions under which traditional graph rewriting can be performed efficiently. Perhaps a similar abstract machine for plan rewriting can be defined. The idea of rule programs also appear in this field and has been implemented in the PROGRES system [Schürr, 1990, Schürr, 1996b]. Perhaps we could reimplement PbR by extending some of the existing graph rewriting systems.

6.2 Plan Rewriting

The most closely related work to the plan rewriting algorithm we have introduced in this thesis is plan merging [Foulser *et al.*, 1992]. Foulser et al. provide a formal analysis and algorithms for exploiting positive interactions within a plan or across a set of plans. However, their work only considers the case on which a set of operators can be replaced by *one* operator that provides the same effects to the rest of the plan and consumes the same or fewer preconditions. Their focus in on optimal and approximate algorithms for this type of operator merging. Plan rewriting in PbR can

be seen as a generalization of operator merging where a subplan can replace another subplan. In fact, the conditions for valid plan rewriting of Section 3.2.2 generalize their mergeability conditions. A difference is that PbR is not concerned with finding the optimal merge (or rewritten plan) in a single pass of an optimization algorithm as their approach does. In PbR we are interested in generating possible plan rewritings during each rewriting phase, not the optimal one. The optimization occurs as the (local) search progresses.

Case-based planning (e.g. [Kambhampati, 1992, Veloso, 1994, Nebel and Koehler, 1995, Hanks and Weld, 1995, Muñoz-Avila, 1998]) solve a problem by modifying a previous solution. There two phases in case-based planning. The first one identifies a plan from the plan library that is most similar to the current problem. In the second phase this previous plan is adapted to solve the new problem. PbR modifies a solution to the current problem, so there is no need for a retrieval phase nor the associated similarity metrics. Plan rewriting in PbR can be seen as type of adaptation from a solution to a problem to an alternate solution for the same problem. That is, a plan rewriting rule in PbR identifies a pair of subplans (the replaced and replacement subplans) that may be interchangeable.

Veloso [Veloso, 1994] describes a general approach to case-based planning based on derivational analogy. Her approach works in three steps. First, the retrieval phase selects a similar plan from the library. Second, the parts of this plan irrelevant to the current problem are removed. Finally, her system searches for a completion of this plan selecting as much as possible the same decisions as the old plan did. In this sense the planning knowledge encoded in the previous solution is transferred to the generation of the new solution plan. The plan rewriting algorithm for partially-specified rules of PbR can be seen as a strongly constrained version of this approach. In PbR the subplan in the rule consequent fixes the steps that can be added to repair the plan. We could use her technique of respecting previous choice points when completing the plan as a way of ensuring that most of the structure of the plan before and after the repair is maintained. This could be useful to constrain the number of rewritten plans for large rewriting rules.

Nebel and Koehler [Nebel and Koehler, 1995] present a computational analysis of case-based planning. In this context they show that the worst case complexity of plan modification is no better than plan generation and point to the limitations of reuse methods. The related problem in the PbR framework is the embedding of the replacement subplan for partially specified rules. As we explained in Section 3.2.4 there may be pathological cases in which the number of embeddings is exponential in the size of the plan or deciding if the embedding exists is NP-hard. However, often we are not interested in finding all rewritings, for example when following a first improvement search strategy. Also the average case behavior seems to be much better in our experience as was presented in Chapters 4 and 5.

Systematic algorithms for case-based planning, such as [Hanks and Weld, 1995], invert the decisions done in refinement planning to find a path between the solution to a similar old problem and the new problem. The rewriting rules in PbR indicate how to transform a solution into another solution plan based on domain knowledge, as opposed to the generic inversion of the refinement operations. Moreover, plan rewriting in PbR is done in a very constrained way instead of an open

search up and down the space of partial plans. However, the rules in PBR may search the space of rewritings non-systematically. Such an effect is ameliorated by using local search.

6.3 Query Planning

In this section, we discuss related approaches both in traditional query optimization and in query planning in mediators.

6.3.1 Traditional Query Optimization

In the database literature, query optimization has been extensively studied (see [Jarke and Koch, 1984] and [Graefe, 1993] for surveys). Query optimizers attempt to both find the most efficient algebraic form of a query and to choose specific methods to implement each data processing operation. For example, a join can be performed by a variety of algorithms, such as nested loops, merge scan, hash join, etc. In our analysis of query planning in mediators we have focused on the algebraic part of query optimization because in our distributed environment the mediator does not have any control over the optimizations employed in the remote information sources, and, so far, the size of data the mediator needs to manipulate locally has not required very sophisticated consideration of implementation algorithms. However, our work on the query planning domain could easily be extended with more operators for each of the distinct implementation methods of the algebraic operations and a cost function that reflects the characteristics of different physical evaluation plans.

The research on traditional query optimization most relevant to our approach lies in three areas: distributed query optimization, declarative and extensible query optimizers, and efficient search algorithms. An algorithm for distributed query optimization based on query tree transformations is presented in [Chu and Hurley, 1982]. Our plan rewriting rules for the relational algebra and the distributed environment are similar to their transformations, but our system accepts arbitrary specification of rules as opposed to a hand-coded algorithm. Chu and Hurley also provide several theorems that describe the optimality of some types of plans under several cost models. A sample theorem is: "if the unit communication cost among databases is the same and the processing cost if a given operation is the same for all computers and is proportional to the volume of data, then placing the unary operations at the lowest possible position in a query tree is a necessary condition to obtain the optimal query processing policy". Using this type of results would enable us to prove that some potential rewriting rules are unnecessary. For example, in a system where the previous theorem hold we could leave out the rewriting rule that pulls selections up in the query tree without any loss of solutions.

The second area is declarative and extensible query optimizers. An extensible query optimizer based on query rewriting was implemented for the Starburst system [Haas *et al.*, 1989, Pirahesh *et al.*, 1992]. Their rules are just condition-action pairs of C functions that manipulate a representation of the queries called the query graph. Although representing the transformations as arbitrary code

allows for maximum generality in the type of rules that can be specified, it certainly does not help in the understandability or maintainability of the system. Our PbR-based query planner uses a declarative rule definition language which is easier to understand and maintain. At the same time, we also provide support for interpreted predicates that can be arbitrary functions in order to analyze the query structures and check for conditions that cannot be easily specified in a declarative manner. Starburst follows a two phase approach to optimization. First, their system generates a set of candidate query graphs by using a set of query graph rewriting rules. Then, for each query graph a query plan is generated and optimized independently. Finally, the best among these optimized query plans is selected. The PbR-based query planner described in chapter 4 performs mainly transformations that correspond to the query rewrite level in Starburst. However, because we assume that cost estimates for the data processing operators are available, our query planner is also performing cost-based transformations, such as join orderings. Moreover, our query plan optimization occurs in a single search space. Starburst covers a greater number of transformations than the ones we have implemented so far. For example, we do not currently support the optimization of aggregation operations, but we could incorporate rewriting rules such as those presented in [Yan and Larson, 1994, Yan and Larson, 1995]. Another difference between the two approaches is the control of the search. Starburst follows a production rule system paradigm in which rules apply until quiescence or a time limit. Our model is that of (local) search and the particular search algorithm can be changed independently of the rest of the query planner.

Another influential work in query optimization is Exodus [Graefe and DeWitt, 1987]. Exodus is a query optimizer generator that compiles a query optimizer out of a given set of operators, transformation rules and the code for the methods that implement each operator. Thus, it is much more declarative than the Starburst system. Although Exodus strives for extensibility, its operator definition language is more restricted than ours. Also it has a fixed search strategy (a form of hill climbing). Exodus distinguishes between transformation rules, that denote the algebraic equivalence of two queries (such as join associativity), and implementation rules, that map an algebraic operator into a particular implementation method (such as performing a join by using a hash join). In our work, we have not considered implementation rules but we could add similar rules. Exodus operates on centralized databases. Volcano [Graefe and McKenna, 1993, Graefe *et al.*, 1994] improves the capabilities of Exodus. Volcano adds a goal-directed dynamic backtracking search, that is more efficient than Exodus' search, along with the ability to define other search strategies. In addition, Volcano offers support for parallel execution, as well as a more modular and extensible design. Our planner shares with Exodus and Volcano the idea of performing all optimization on a single search space. Another interesting idea emerging from these projects is that of dynamic query evaluation plans [Graefe and Ward, 1989, Cole and Graefe, 1994]. Dynamic query evaluation plans include several alternative subplan which are chosen for execution at depending on run-time conditions. From the planning perspective, this is a simple form of contingency planning. We expect that a general planning framework such as PbR would be able to provide more general interleaving of planning and execution [Knoblock, 1995]. In particular, the inclusion of information gathering actions [Ashish *et al.*, 1997]. This actions do not have a

correspondence in traditional query optimization. However, they are quite useful in a distributed and heterogeneous environment. For example, consider a simple query that requests the work phone number of Steve Minton. One possibility is to query in parallel all the phone books to which our system has access. This can be very expensive. A better alternative is to consult a source that can provide the affiliation of Steve Minton, which would produce ISI. Then, go to the phone list of ISI to find his phone. This is a much more cost effective evaluation plan.

The third area of work is on efficient search algorithms for query optimization. Local search techniques have proven the most scalable. Ioannidis and Kang applied iterative improvement, simulated annealing, and a combination of the two, called two phase optimization, to large join queries [Ioannidis and Kang, 1990]. Two phase optimization consists in applying iterative improvement up to a local minima, which in turn serves as the starting point to a simulated annealing search. This last method performs best in their experiments. In [Swami and Gupta, 1988, Swami, 1989] another set of local search strategies is presented with similarly good scaling results. Since our PbR framework is modular, these different search methods can be incorporated. More importantly, methods that perform well in one domain may transfer to other planning domains.

6.3.2 Query Planning in Mediators

A number of projects have focussed on query planning for mediators. For example, the Information Manifold [Levy *et al.*, 1996b], TSIMMIS [Hammer *et al.*, 1995], HERMES [Adali *et al.*, 1996], and Garlic [Tork Roth *et al.*, 1996, Haas *et al.*, 1997, Roth and Schwarz, 1997]. TSIMMIS and the Information Manifold do not specifically address cost-based optimization. Although the Information Manifold does find retrievable plans that access the minimum number of sources. Cost based optimization could be incorporated in these systems in a two phase approach in which a set retrievable plans are found first, and then each optimized independently.

HERMES [Adali *et al.*, 1996] does address issues of cost-based optimization. Their mediator uses an expressive logic language to integrate a set of information sources. Their system includes a rule rewriter that transforms the logic programs that evaluate a user query to a more cost effective form by pushing selections to the sources, reordering subgoals in a rule, and using cached relations. However these transformations are expressed procedurally. They do not focus in an extensible or declarative query optimization framework, such as our PbR-based query planner.

The Garlic project [Tork Roth *et al.*, 1996, Haas *et al.*, 1997, Roth and Schwarz, 1997] does consider cost optimization for mediators in the style of Starburst. In addition to traditional query optimization rewrites, their system also considers remote evaluation of subqueries at the information sources. Query optimization proceeds in three stages. For example, for conjunctive queries, their system first selects the sources, then it uses a dynamic programming enumeration of join orders, and finally, it places the operations not achieved so far. At that point they choose the best cost plan. Their query rewritings are implemented as C++ functions. Our PbR-based query planner is declarative and performs the search in a single space in the style of Exodus and Volcano.

Moreover our local search approach is more scalable than their complete enumeration of all query plans.

Despite the practical importance of query planning, there has been little work in the planning literature (either in traditional or in query planning for mediators). Occam [Kwok and Weld, 1996] is a planner for information gathering in distributed and heterogeneous domains that focuses on the source selection problem. Our work combines both source selection and traditional cost-based query optimization. Sage [Knoblock, 1996] considers plan quality and supports interleaving of planning and execution. PbR does not currently interleave planning and execution, but it is as general as Sage with better scaling properties as shown in Section 4.4.

6.4 Other Applications

Transformation-based optimization techniques such as PbR have appeared in several domains. The code optimization phase of compilers performs rewritings analogous to the PbR-rewriting rules [Aho *et al.*, 1986]. However, PbR currently works only on acyclic plans. Therefore our plan language cannot describe directly code segments containing loops. However, this is a limitation of the underlying planning model not of the PbR framework. The plan rewriting engine and plan rewriting language of PbR would apply directly to plans with loops for fully-specified rewriting rules. For partially-specified rules the algorithm would need to be extended according to the semantics of plans with iteration.

Iterative repair has been applied successfully in scheduling applications. Above we already mentioned [Minton, 1992] where large scheduling problems, operations on the Hubble Space Telescope, were solved by local search using the simple min-conflicts heuristic. In work on scheduling and rescheduling, [Zweben *et al.*, 1994] define a set of general, but fixed, repair methods, and use simulated annealing to search the space of schedules. Our plans are networks of actions as opposed to the metric-time totally-ordered tasks. Also we can easily specify different rewriting rules (general or specific) to suit each domain, as opposed to their fixed strategies.

In summary, the advantage of PbR is being domain-independent so that the core rewriting engine and search strategies need to be implemented only once. Furthermore, this generality enables results from one application domain to be easily transferable to another.

Chapter 7

Discussion

This thesis has presented Planning by Rewriting, a new paradigm for efficient high-quality domain-independent planning. PbR adapts graph rewriting and local search techniques to the semantics of domain-independent partial-order planning. The basic idea of PbR consists in transforming an easy-to-generate, but possibly suboptimal, initial plan into a high-quality plan by applying declarative plan rewriting rules in an iterative repair style.

There are several important advantages to the PbR planning approach. First, PbR is a *declarative domain-independent* framework, which brings the benefits of reusability and extensibility. Second, it addresses *sophisticated plan quality* measures, while most work in domain-independent planning has not addressed quality or does it in very simple ways. Third, PbR is *scalable* because it uses efficient local search methods. Finally, PbR is an *anytime* planning algorithm that allows trading planning effort for plan quality in order to maximize the utility of the plans.

Planning by Rewriting can be viewed as a domain-independent framework for local search. PbR accepts domain specifications in an expressive operator language, declarative plan rewriting rules (to generate the neighborhood of a plan), complex quality metrics, and arbitrary (local) search methods. In our work, we provide domain-independent solutions to the main issues in local search. First, we provide two general methods to efficiently construct initial plans: a generative planner with an strong search bias, and high-level programmatic facilities. Second, we define a declarative language in which to specify plan rewriting rules and a rewriting algorithm for this language. The rewriting language is very flexible and can be extended to be quite expressive while remaining efficient. Finally, we adapt several local search strategies to the PbR framework, such as variations of gradient descent.

Planning by Rewriting is not only useful as a black-box planning approach, PbR is also well-suited to mixed-initiative planning. In mixed-initiative planning, the user and the planner interact in defining the plan. For example, the user defines the quality criteria of interest at the moment, specifying which are the available actions, etc. Some domains can only be approached through mixed-initiative planning. For example, when the quality metric is very expensive to evaluate, such as in geometric analysis in manufacturing, the user must guide the planner towards good quality plans in a way that a small number of plans are generated and evaluated. Another example

is when the plan quality metric is multi-objective or changes over time. Several characteristics of PbR support mixed-initiative planning. First, because PbR offers complete plans, the user can easily understand the plan and perform complex quality assessment. Second, the rewriting rule language is a convenient mechanism by which the user can propose modifications to the plans. Third, by selecting which rules to apply or their order of application the user can guide the planner.

PbR also introduces a framework in which to perform sensibility analysis for plans. By finding the quality of the possible rewritings around a solution we can evaluate the robustness of a plan with respect to a set of possible changes.

Our framework achieves a balance between domain knowledge, expressed as plan rewriting rules, and general local search techniques that have been proved useful in many hard combinatorial problems. We expect that these ideas will push the frontier of solvable problems for many practical domains in which high quality plans and anytime behavior are needed.

7.1 Contributions

The main contribution of this thesis is the definition of a novel planning paradigm, Planning by Rewriting, that addresses the combined challenges of flexibility, planning efficiency, and support for complex quality metrics. In order to meet these challenges PbR offers a framework for the *principled* application of iterative repair techniques to classical domain-independent planning.

We have defined and implemented the algorithms for domain-independent plan rewriting. We described a declarative language in which to specify plan rewriting rules. We provided support for two types of plan rewriting rules: fully-specified and partially-specified. Fully-specified rewriting rules are typical of graph rewriting systems. The novel class of partially-specified rewriting rules has no counterpart in general graph rewriting. Partially-specified rules capture naturally and concisely powerful plan transformations. By relying on the strong semantics of partial-order planning we defined a plan rewriting algorithm for partially-specified rules.

We have applied our framework to several domains with an emphasis on query planning in mediator systems. The resulting PbR-based query planner is scalable, flexible, has anytime behavior, and yields a novel combination of traditional query optimization and heterogeneous information source selection. This novel approach to query optimization in mediators constitutes a second contribution of this thesis. Previous approaches perform the query processing in two stages. First, they find all possible translations to the user query in terms of source terms (source selection). Then each of these retrievable queries is optimized by traditional methods separately. Finally, the best alternative is selected. This approach does not scale when there are many alternative information sources such as is the case in the Web. PbR performs both aspects of query processing in mediators in a single search process. By using local search techniques the combined optimization is scalable and supports an anytime behavior.

7.2 Future Work

The planning style introduced by PbR opens many areas of research. In the first place, there is a great potential for applying machine learning techniques to PbR. An important issue is the generation of the plan rewriting rules. Conceptually, plan rewriting rules arise from the chosen plan equivalence relation. All valid plans that achieve the given goals in a finite number of steps, i.e. all solution plans, are (satisfiability) equivalent. Each rule arises from a theorem that states that two subplans are equivalent for the purposes of achieving some goals, with the addition of some conditions that indicate in which context that rule can be usefully applied.

We believe that the plan rewriting rules can be generated by fully automated procedures. The methods can range from static analysis of the domain operators to analysis of sample equivalent plans that achieve the same goals but at different costs. Note the similarity with methods to automatically deduce search control [Minton, 1988b, Etzioni, 1993], and also the need to deal with the utility problem.

Beyond learning the rewriting rules, we intend to develop a system that can automatically learn the optimal planner configuration for a given planning domain and problem distribution in a manner analogous to Minton's Multi-TAC system [Minton, 1996]. Our system would perform a search in the configuration space of the PbR planner proposing candidate sets of rewriting rules and different search methods. By testing each proposed configuration against a training set of simple problems, the system would hill-climb in the configuration space in order to arrive at the most useful rewriting rules and search strategies for the given planning domain and distribution of problems.

There are many advanced techniques in the local search literature that can be adapted and extended in our framework. In particular, the idea of variable-depth rewriting leads naturally to the creation of rule programs, which specify how a set of rules are applied to a plan. We have already seen how in query planning we could find transformations that are better specified as program of simple rewriting rules. For example, a sequence of `Join-Swap` transformations may put two retrieve operators on the same database together in the query tree and then `Remote-Join-Eval` would collapse the explicit join operator and the two retrieves into a single retrieval of a remote join. More complex examples of this sort of programs are presented in [Cherniack and Zdonik, 1996, Cherniack and Zdonik, 1998]. This work defines a query optimizer for object-oriented languages based on sophisticated programs of rewriting rules.

Another area for further research is the interplay of plan rewriting and plan execution. Sometimes the best transformations for a plan may only be known after some portion of the plan has been executed. This information obtained at run-time can guide the planner to select the appropriate rewritings. For example, in query planning the plans may contain information gathering actions [Ashish *et al.*, 1997] and depend on run-time conditions. This yields a form of dynamic query optimization. Interleaved planning and execution is also necessary in order to deal effectively with unexpected situations in the environment such as database or network failures.

An open area of research is to relax our framework to accept incomplete plans during the rewriting process. This expands the search space considerably and some of the benefits of PbR, such as its anytime property, are lost. But for some domains the shortest path of rewritings from the initial plan to the optimal may contain incomplete or inconsistent plans. This idea could be embodied as a planning style that combines the characteristics of generative planning and Planning by Rewriting. Note that the plan refinements both of partial order planning [Kambhampati *et al.*, 1995] and Hierarchical Task Network Planning [Erol *et al.*, 1994] can be easily specified as plan rewriting rules.

The query planning domain presents a great number of opportunities to push the limits of our framework (and planning in general), as we develop more sophisticated planning domains. There are several directions in which to develop more sophisticated planning domains. First, we could incorporate more complex query processing operators, such as aggregation and grouping. For example, we could build on the rewriting rules introduced in [Yan and Larson, 1995]. Second, we could push towards a more refined consideration of evaluation methods for each of the data processing operators as in traditional query optimization. For example, the algebraic join can be evaluated by a variety of methods, such as sort-join, hash-join, etc. Third, we could define operators and rewriting rules specialized for non-traditional data types, such as multimedia data. Fourth, we could incorporate more sophisticated model of the capabilities of the sources. In fact, we are in the process of adding support for binding patterns pattern constraints in source access. Binding patterns are annotations on the schema of an information source that indicate that some arguments have to be bound to constants. Their role is similar to input parameters that have to be provided in order to retrieve a set of output values. Many sources on the Web have binding constraints. For example, some web sites that provide stock information on a company need as input the ticker symbol of the company. Finally, interleaving query planning and execution presents very interesting opportunities for optimization. Specially in domains in which sources have limited query capabilities such as binding pattern constraints. Information gathering on the WWW is such an environment.

Applying PbR to other domains will surely provide new challenges and the possibility of discovering and transferring general planning techniques from one domain to another. We hope that the local search methods used by PbR will help planning techniques to scale to large practical problems and conversely that the domain-independent nature of PbR will help in the analysis and principled extension of local search techniques.

Appendix A

Detailed Example of PbR for Query Planning

In this section we present a detailed example of PbR applied to query planning in mediators. First, we introduce a simple domain model and a set of sources relevant to this model. Second we present the set of integration axioms compiled from this domain model. Third, we show a portion of the space of solution plans as it is explored by the application of the rewriting rules. Finally, we discuss the limitations of the current encoding and rewriting rules for query planning.

Consider the domain model and sources shown in Figure A.1. This model is a small fraction of a logistics application. We show a hierarchy with different classes of seaports and some sources for those classes of information. For example, the model states that `Large Seaport` and `Small Seaport` form a covering of the class `Seaport`, and that the `Large Seaport`s are exactly those with more than seven cranes. The description of the sources in terms of the domain model is depicted with dashed lines. For example, the source `S4` provides the port name (`pn`), geographic code (`gc`), and number of cranes (`cr`) for `American Large Seaport`s.

Given a domain model and source descriptions, our system automatically compiles a set of integration axioms (cf. Section 4.1). Each integration axiom indicate an alternative combination of sources that provide a given set of attributes for a domain class. In order to facilitate the retrieval and the computation of new axioms at runtime the integration axioms are organized into a lattice.[1] The complete lattice of integration axioms for the concepts `large-seaport` and `seaport` are shown in Figures A.2 and A.3 respectively. Each node of this lattice contains a set of alternative axioms for a given set of attributes of a domain model class. For example the bottom node of Figures A.2 describe five alternative axioms for the attributes port name (`pn`), geographic code (`gc`), and number of cranes (`cr`). The first axiom in this node ($LS^1_{gc,pn,cr}$) states that these attributes can be obtained from the union of the sources `S4` and `S5`. The second axiom ($LS^2_{gc,pn,cr}$) presents an alternative: accessing source `S1`, which contains all seaports, and selecting those with more than 7 cranes, which ensures that are large seaports.

In order to explain the details of our approach to query planning in mediators, we will use a simple query based on the previous integration model and axioms:

[1] See [?] for details on how some of these integration axioms are precompiled and the rest computed on demand taking advantage of the lattice structure.

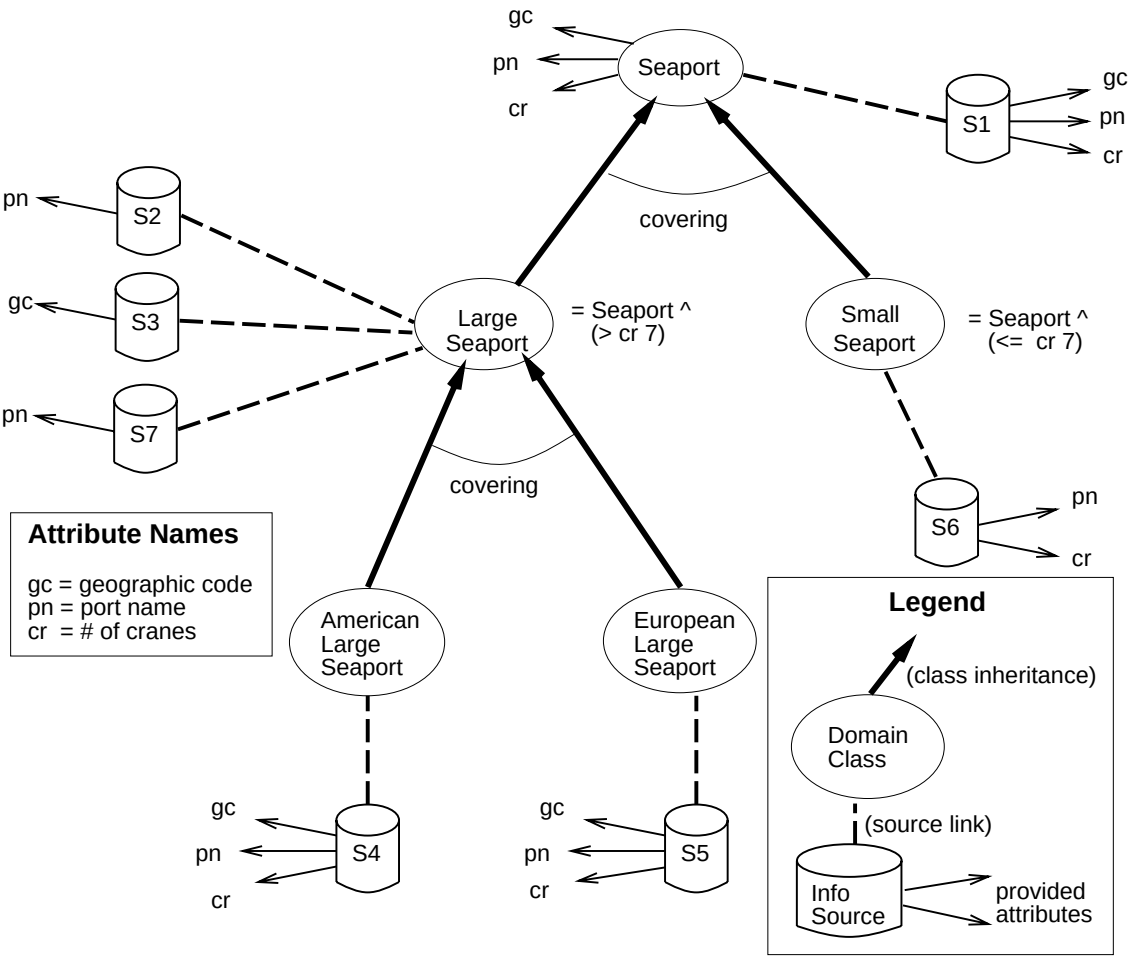


Figure A.1: Sample Domain Model and Available Sources

$$\texttt{q(gc pn cr g c)} \leftarrow \texttt{LS(gc pn cr)} \wedge \texttt{LZ(g p c)} \wedge \texttt{cr = c}$$

In this simple conjunctive query, `LS` represents large seaports. For brevity we assume that there exists a concept `LZ` isomorphic to `LS`, thus having the analogous axioms. Figure A.4 shows in detail an initial plan that evaluates this query using axioms $LS^1_{gc,pn,cr}$ and $LZ^1_{gc,pn,cr}$. The initial plan was generated by first randomly parsing the given query, then, for each domain concept (LS and LZ in this case), finding an integration axiom that obtains the required attributes for the domain concept (in this figure axioms $LS^1_{gc,pn,cr}$ and $LZ^1_{gc,pn,cr}$).

There are two important features of our encoding. First, the query expressions in the operators are kept at the domain level as a way of providing a layer of abstraction on the choices of particular sources. The plan only commits to which sources it uses on the retrieve operators. Second, the relational algebra operators are extended with reasoning about integration axioms. For example, the union operator in the left of the figure can decompose the query `LS(gc pn cr)`, which is a

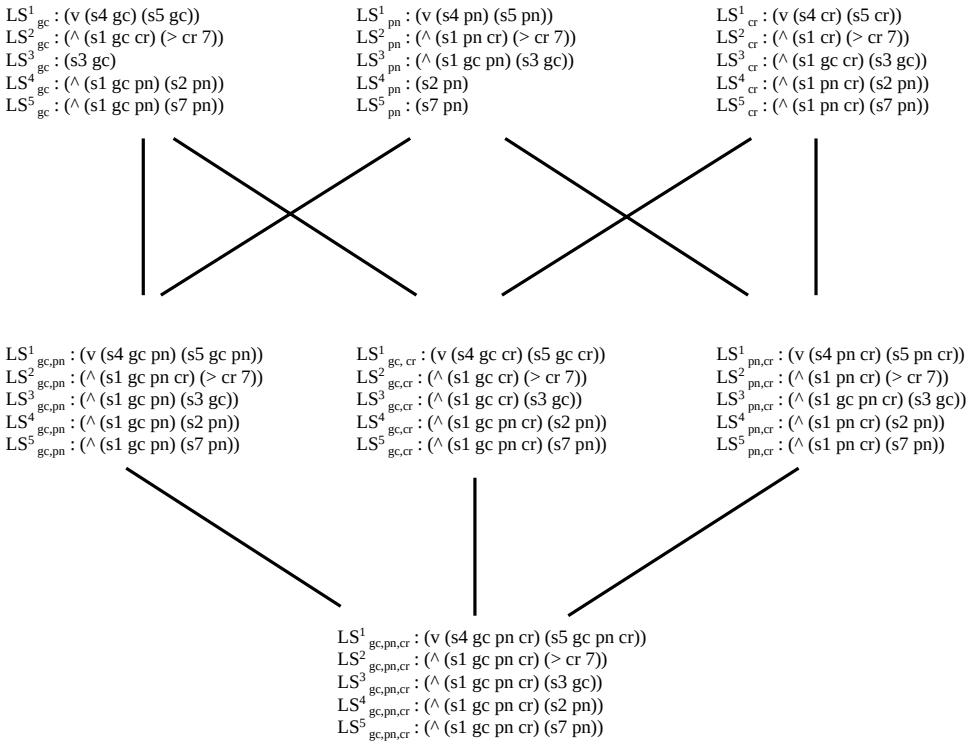


Figure A.2: Lattice of Integration Axioms for Large Seaport

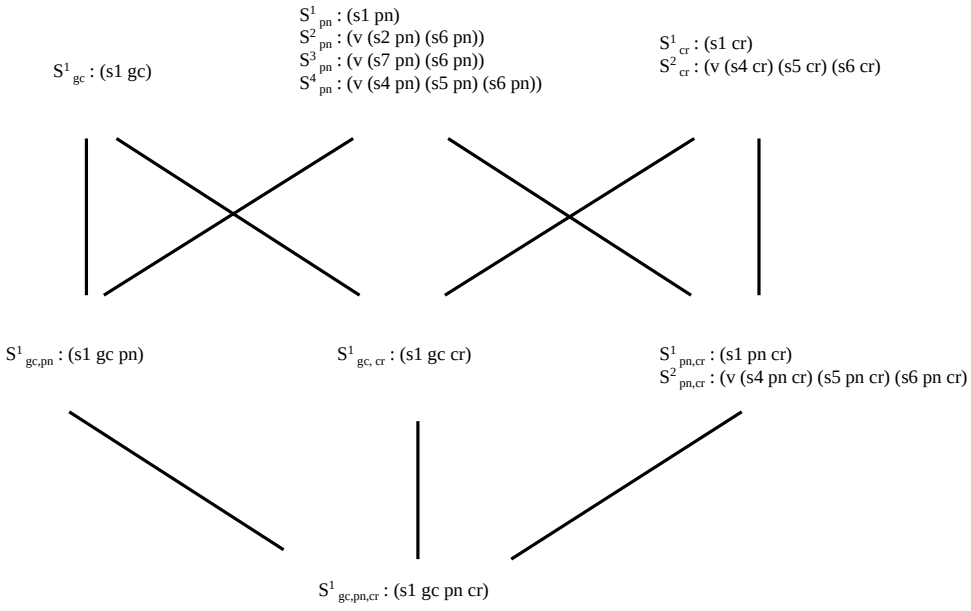


Figure A.3: Lattice of Integration Axioms for Seaport

single domain concept, because there exists an integration axiom, namely $LS^1_{gc,pn,cr}$, that obtains `LS(gc pn cr)` by the union of two sources, namely `s4` and `s5`.

In the figures in this section the most important parameters of each operator are shown (according to the specification in Figure 4.3). For example, the topmost join of Figure A.4 is annotated with two subqueries it joins (`sq1: LS(gc pn cr)` and `sq2 LZ(g p c)`), the join conditions (`cr = c`), and the query that results from the join (q: `LS(gc pn cr)` $\wedge$ `sq2 LZ(g p c)` $\wedge$ `cr = c`). Also, each operator is annotated with the axiom that introduced it in the plan, or the letter Q to indicate that the operator was needed to implement the original query, but this annotation is not available to the query planner.

Figure A.5 shows the plan resulting from substituting axiom $LS^1_{gc,pn,cr}$ by $LS^5_{gc,pn,cr}$ in the plan in Figure A.4 according to the approach presented in Section 4.3.3.3. A rule analogous to the one in Figure 4.13 performs this transformation. Note that the algebraic implementation of the axiom lies completely in a subtree of the query plan.

Figures A.6 and A.7 show two rewritten plans after the `Join-Swap` rule is applied to the plan in Figure A.5. Figure A.8 shows the rewriting after applying `Remote-Join-Eval` to the plan in Figure A.5. Applying the distributive property of join and union to the plan in Figure A.6 results in the plan in Figure A.9. The search process will proceed in this fashion applying rewriting rules and moving to new solution plans.

Our `Swap-Axioms` rule is very conservative: it only replaces an integration axiom when its implementation lies in a subtree of the query plan. We considered transformations that exchange axioms even when the components of the axiom are distributed arbitrarily in the plan. However, the transformations in such cases may require extensive changes to the plan. The complexity of reasoning what are these changes and applying the changes to the plan argue against implementing such more powerful transformations. One way of implementing them could be as a sequence of simpler transformations in a variable-depth rewriting fashion. Another alternative could be to use a version of derivational analogy [Veloso, 1994] that respected as much of the old plan as possible but could change the plan freely.

As an example of the complexity of exchanging axioms consider the plan in Figure A.10. This is the plan that we would like to obtain after replacing axiom $LS^5_{gc,pn,cr}$ by $LS^3_{gc,pn,cr}$ in the plan in Figure A.9. By careful reasoning the rewriting would realize that it can reuse the steps involving `S(gc pn cr)`, replace the retrieval of `LS(pn)` by `LS(gc)`, and change the join conditions to be on `gc` as opposed to `pn`. This is complicated enough but seems feasible. However, consider a slight variation of our query in which `gc` is not required in the output. A plan for this new query, analogous to the one in Figure A.9, is shown in Figure A.11. Replacing axiom $LS^5_{pn,cr}$ by $LS^3_{pn,cr}$ on this plan is much harder. Note that because `gc` is not a part of the left tree of the root join of plan `1a1a`, the subexpressions involving S(pn cr) cannot not be reused directly. There are two options. If we want to conserve as much as possible the structure of the current plan, the transformation would add `gc` throughout the queries and operators in left tree. Obviously this solution is not scalable, given that the tree can be arbitrarily large. The resulting plan would be the same as in Figure A.12. The second option amounts to reconstructing the plan from the highest node that

involves some component of the axiom to be replaced. In the worst case this is the root node of the query tree and the whole query plan needs to be replanned from scratch. Such is the case in our example, the resulting plan, that uses the new axiom $LS^3_{pn,cr}$, is shown in Figure A.13.

Finally, Figure A.14 shows a query plan with intermediate queries represented at the source level. That representation is considerably more constrained than the one we have used, that is, to keep intermediate queries at the domain level. Figure A.15 shows the result of exchanging one axiom. As it can be seen the query plan required extensive changes even though the axiom was clustered in a subtree. Our encoding using domain level terms as an abstraction mechanism avoids such limitations and allows a greater number of transformations to apply to the plans.

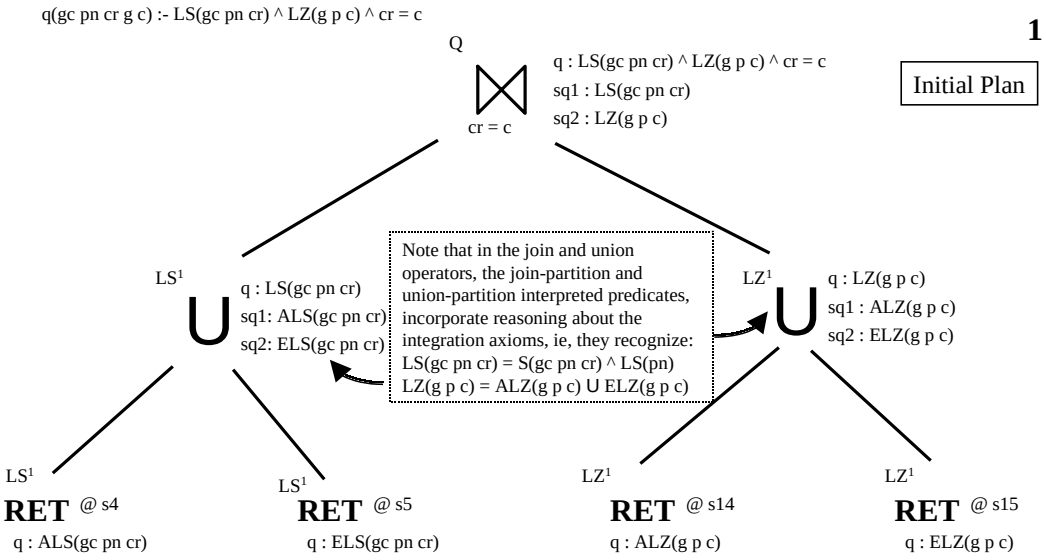


Figure A.4: Query Plan P1

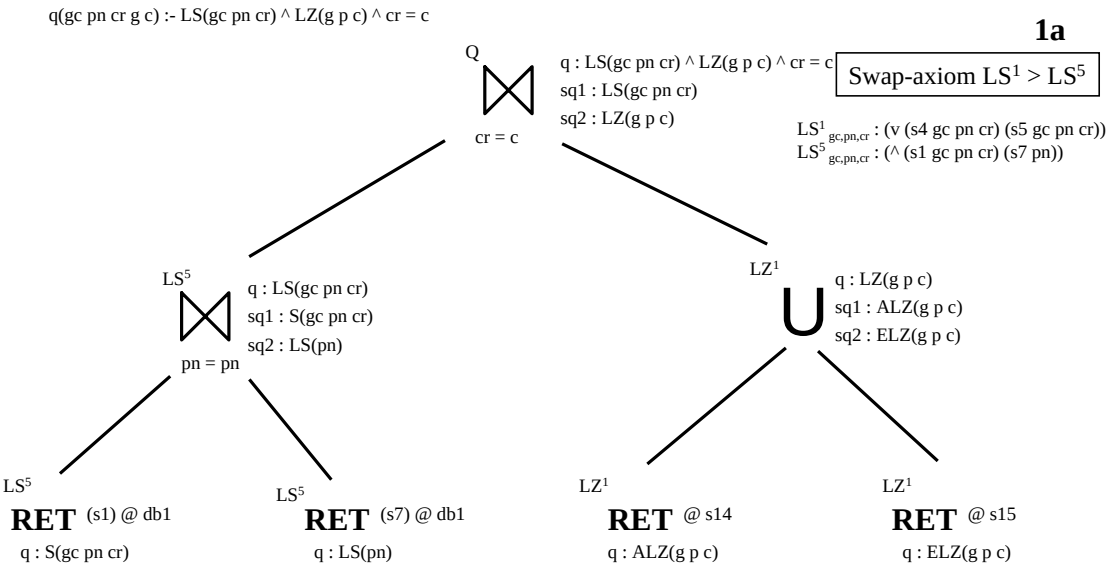


Figure A.5: Query Plan P1a

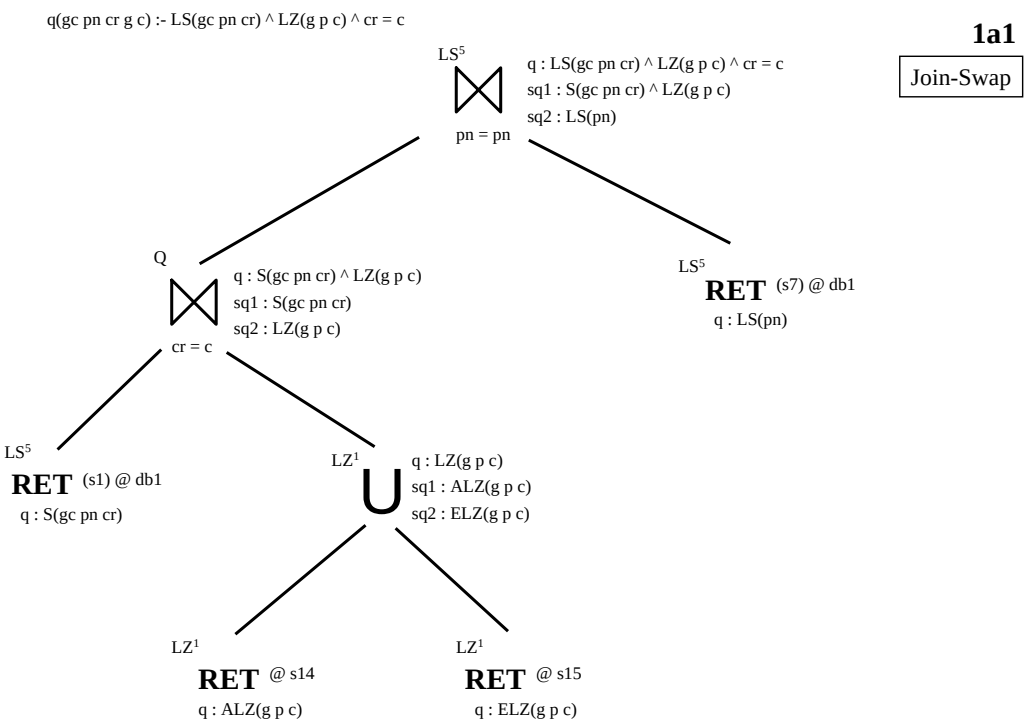


Figure A.6: Query Plan P1a1

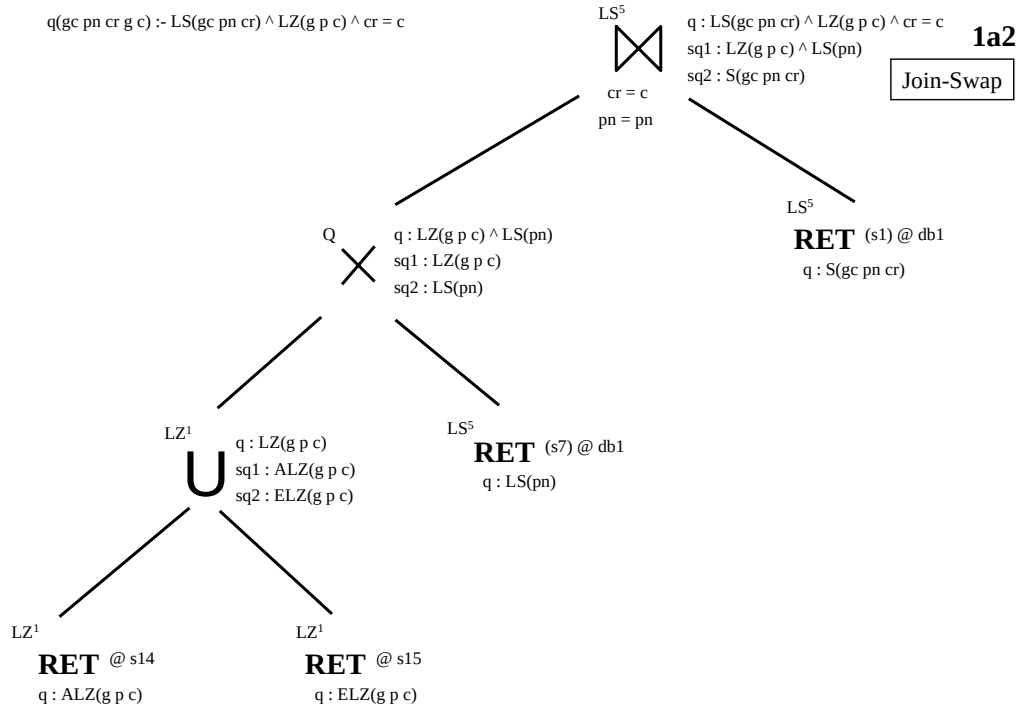


Figure A.7: Query Plan P1a2

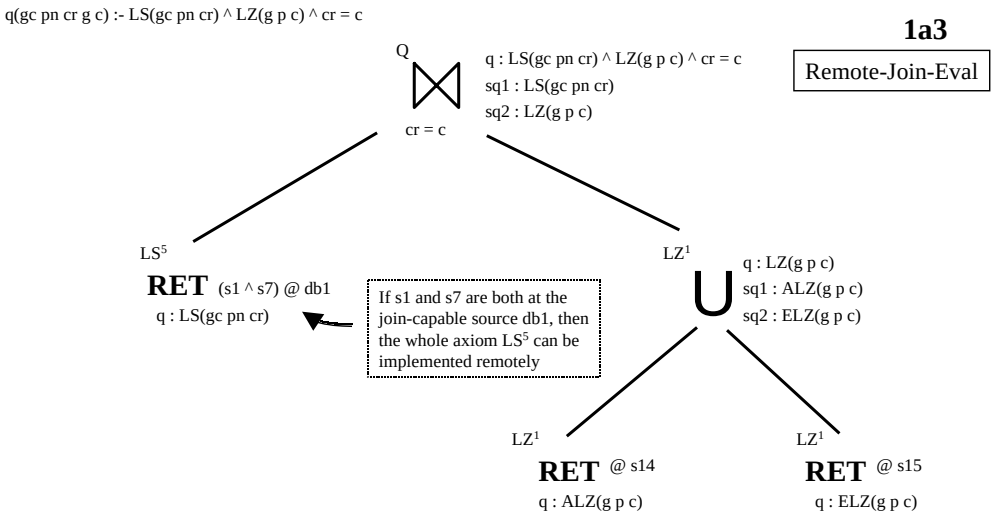


Figure A.8: Query Plan P1a3

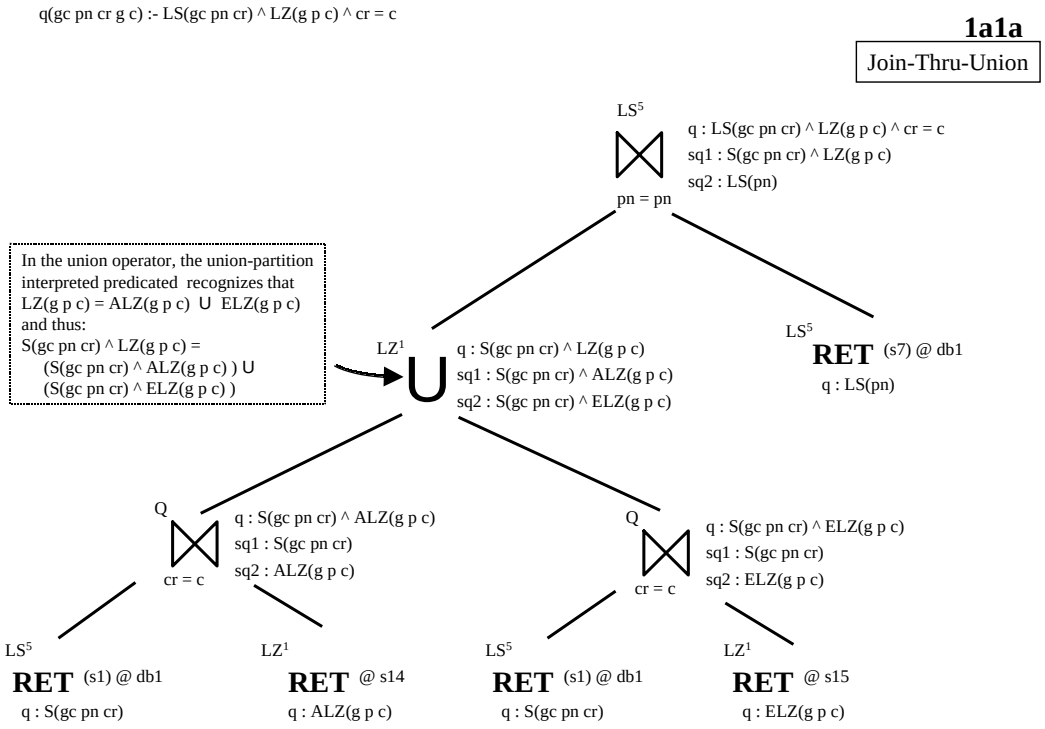


Figure A.9: Query Plan P1a1a

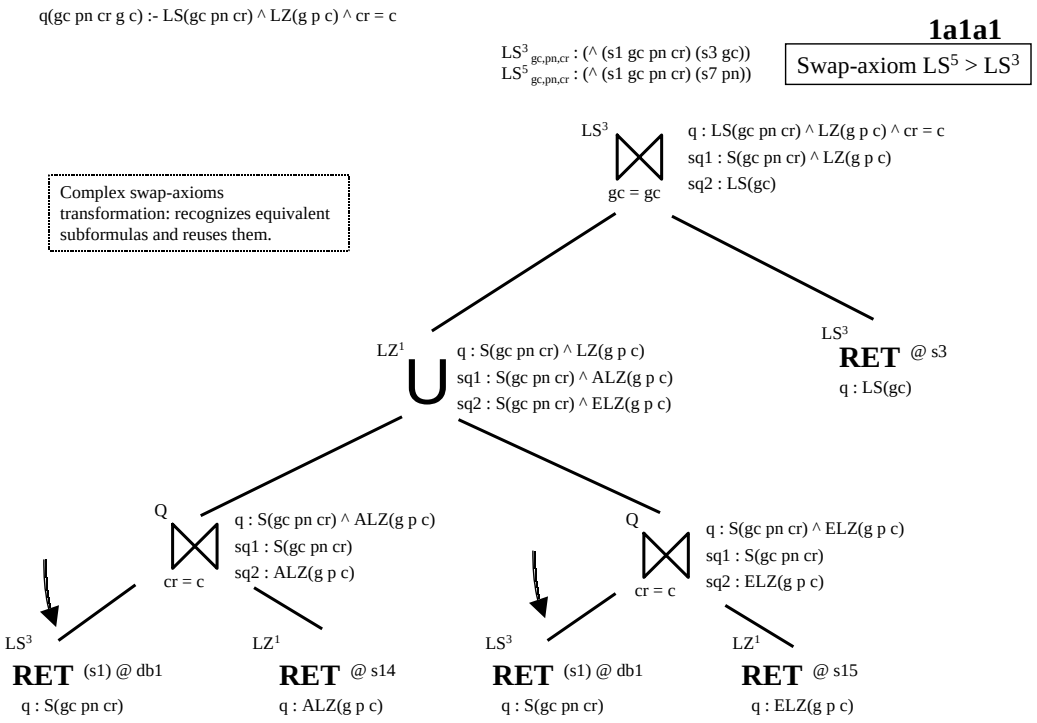


Figure A.10: Query Plan P1a1a1

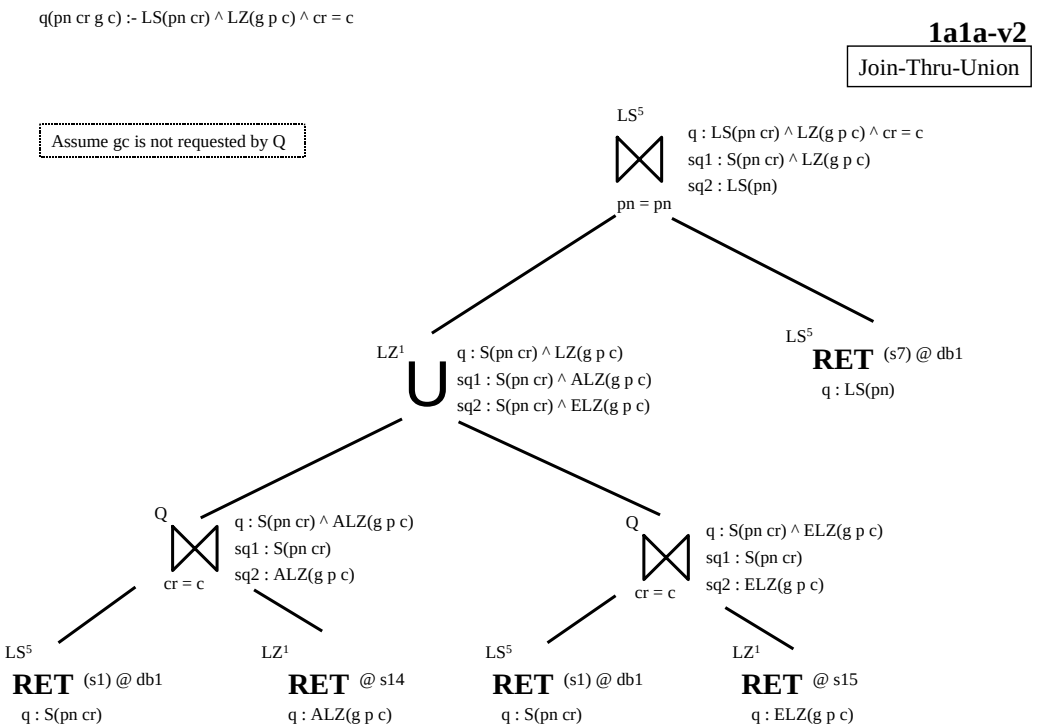


Figure A.11: Query Plan P1a1a-v2

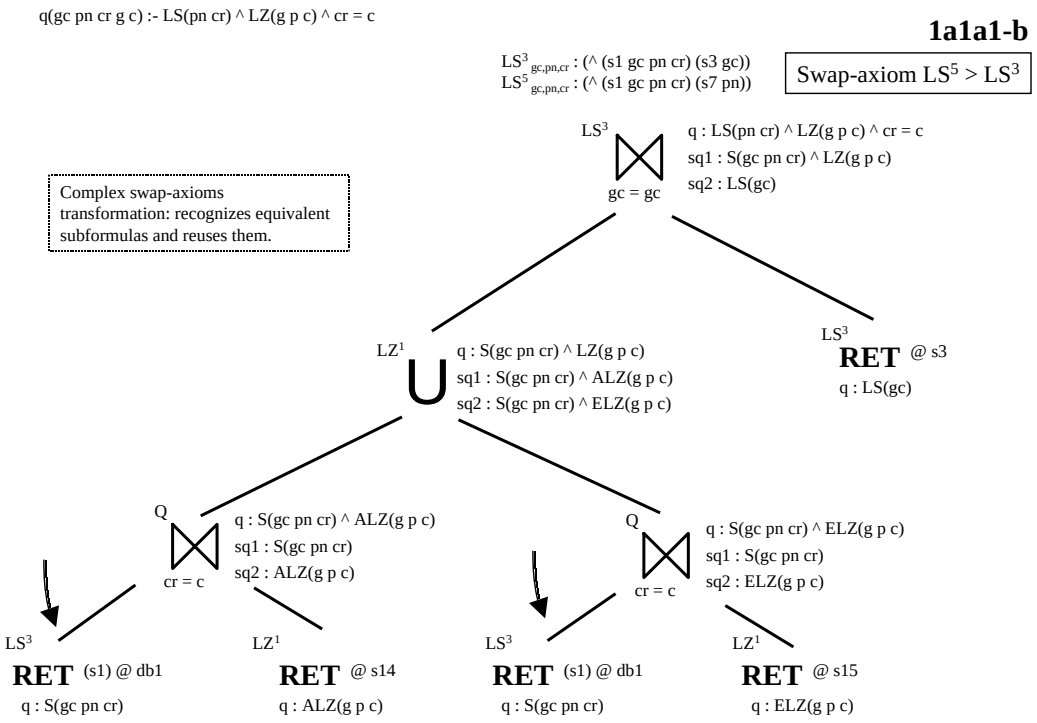


Figure A.12: Query Plan P1a1a1-b

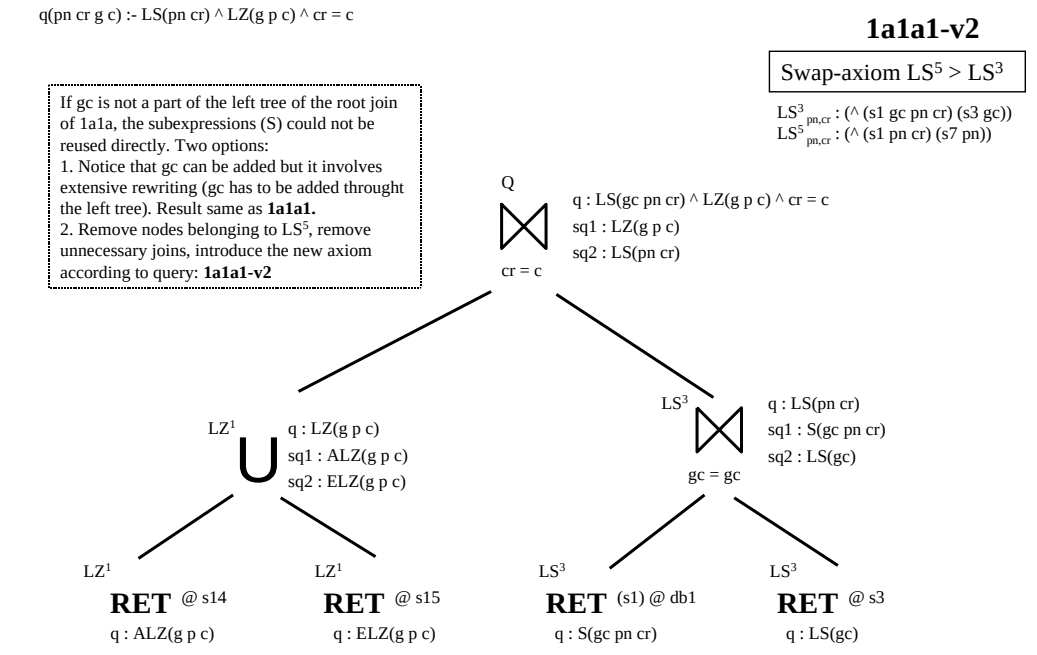


Figure A.13: Query Plan P1a1a1-v2

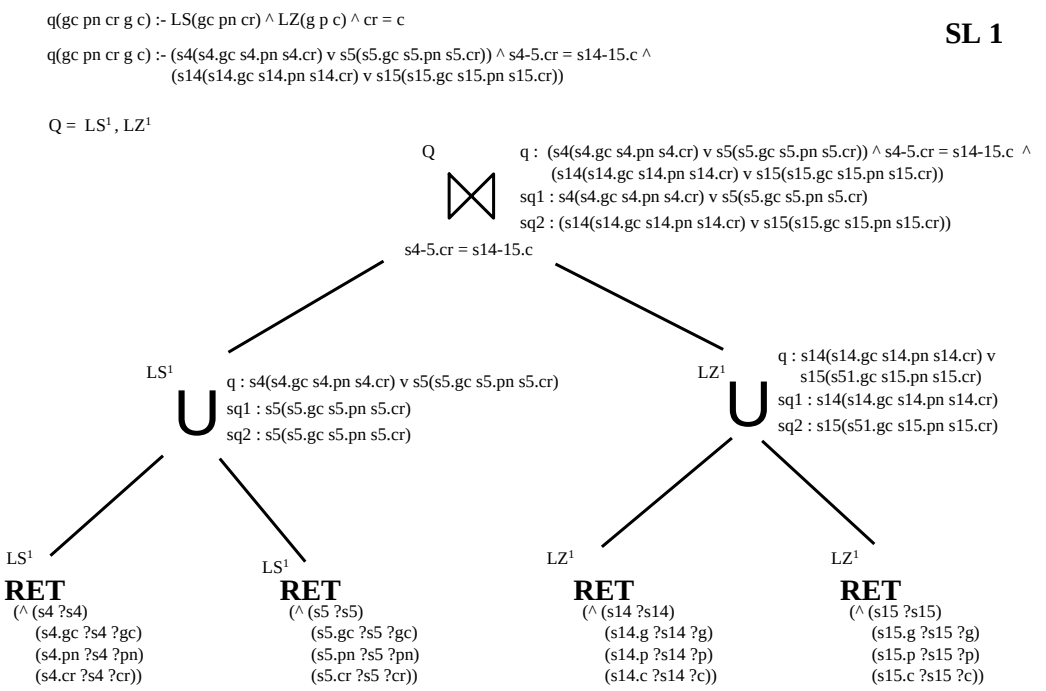


Figure A.14: Query Plan Psl1

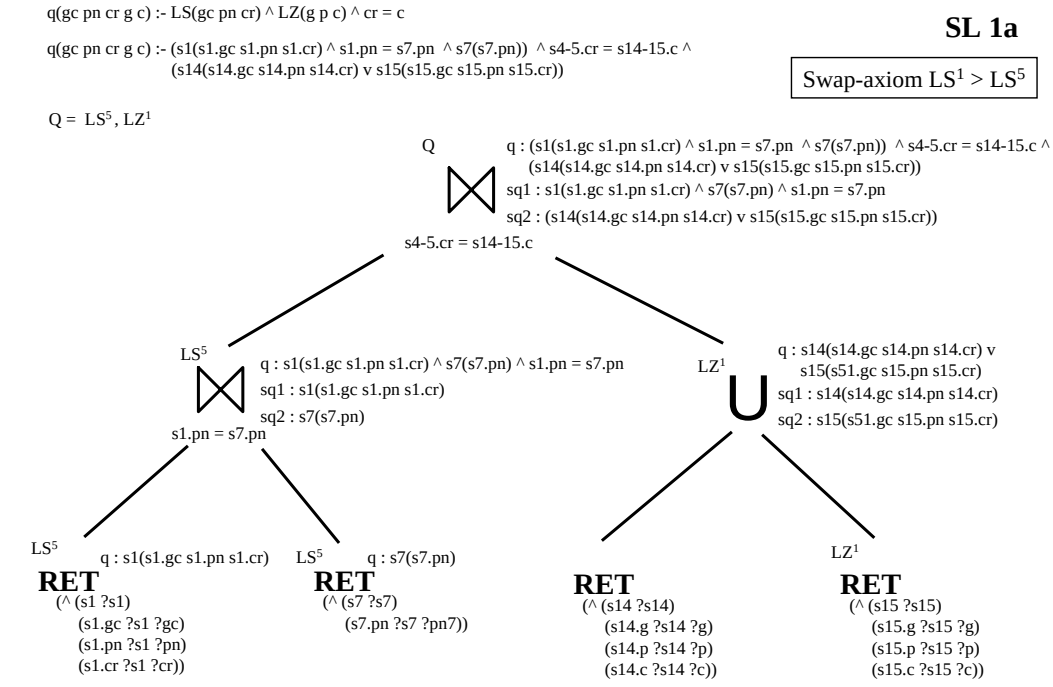


Figure A.15: Query Plan Psl1a

Reference List

[Aarts and Lenstra, 1997] Emile Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley and Sons, Chichester, England, 1997.

[Abiteboul *et al.*, 1995] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[Adali *et al.*, 1996] Sibel Adali, Kasim Selcuk Candan, Yannis Papkonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2):137–148, June 1996.

[Aho *et al.*, 1986] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers Principles, Techniques and Tools*. Addison-Wesley, 1986.

[Ambite and Knoblock, 1997] José Luis Ambite and Craig A. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, RI, 1997.

[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*, 6(2/3):99–130, 1996.

[Ashish *et al.*, 1997] Naveen Ashish, Craig A. Knoblock, and Alon Levy. Information gathering plans with sensing actions. In Sam Steel and Rachid Alami, editors, *Recent Advances in AI Planning: 4th European Conference on Planning, ECP'97*. Springer-Verlag, New York, 1997.

[Avenhaus and Madlener, 1990] Juergen Avenhaus and Klaus Madlener. Term rewriting and equational reasoning. In *Formal Techniques in Artificial Intelligence*, pages 1–43. Elsevier, North Holland, 1990.

[Baader and Nipkow, 1998] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[Bacchus and Kabanza, 1995] Fahiem Bacchus and Froduald Kabanza. Using temporal logic to control search in a forward chaining planner. In *Proceedings of the 3rd European Workshop on Planning*, 1995. Available via the URL ftp://logos.uwaterloo.ca:/pub/tlplan/tlplan.ps.

[Bäckström and Nebel, 1995] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[Bäckström, 1994a] Christer Bäckström. Executing parallel plans faster by adding actions. In A. G. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence*, pages 615–619, Amsterdam, Netherlands, August 1994. John Wiley and Sons.

[Bäckström, 1994b] Christer Bäckström. Finding least constrained plans and optimal parallel executions is harder that we thought. In C. Bäckström and E. Sandewell, editors, *Current Trends in AI Planning: Proceedings of the 2nd European Workshop on Planning (EWSP-93)*, pages 46–59, Vadstena, Sweeden, December 1994. IOS Press (Amsterdam).

[Blum and Furst, 1995] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[Bonet *et al.*, 1997] Blai Bonet, Gábor Loerincs, and Héctor Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 714–719, Providence, RI, 1997.

[Bylander, 1994] Tom Bylander. The computation complexity of propositional strips. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[Carbonell *et al.*, 1991] Jaime G. Carbonell, Craig A. Knoblock, and Steven Minton. PRODIGY: An integrated architecture for planning and learning. In Kurt VanLehn, editor, *Architectures for Intelligence*, pages 241–278. Lawrence Erlbaum, Hillsdale, NJ, 1991.

[Cherniack and Zdonik, 1996] Mitch Cherniack and Stanley B. Zdonik. Rule languages and internal algebras for rule-based optimizers. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2):401–412, June 1996.

[Cherniack and Zdonik, 1998] Mitch Cherniack and Stanley B. Zdonik. Changing the rules: Transformations for rule-based optimizers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 61–72, Seattle, WA, 1998.

[Chu and Hurley, 1982] Wesley W. Chu and Paul Hurley. Optimal query processing for distributed database systems. *IEEE Transactions on Computers*, 31(9):835–850, September 1982.

[Cole and Graefe, 1994] Richard L. Cole and Goetz Graefe. Optimization of dynamic query evaluation plans. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):150–160, June 1994.

[Dean and Boddy, 1988] Tom Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, Saint Paul, Minnesota, 1988.

[Dorr, 1995] Heiko Dorr. *Efficient graph rewriting and its implementation*, volume 922 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1995.

[Duschka and Genesereth, 1997] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, May 1997.

[Duschka, 1997] Oliver M. Duschka. *Query Planning and Optimization in Information Integration*. PhD thesis, Stanford University, 1997.

[Erol *et al.*, 1994] Kutluhan Erol, Dana Nau, and James Hendler. UMCP: A sound and complete planning procedure for hierarchical task-network planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 249–254, Chicago, IL, 1994.

[Erol *et al.*, 1995] Kutluhan Erol, Dana Nau, and V. S. Subrahmanian. Decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1-2):75–88, 1995.

[Etzioni, 1993] Oren Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2):255–302, 1993.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

[Forgy, 1982] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[Foulser *et al.*, 1992] David E. Foulser, Ming Li, and Qiang Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2–3):143–182, 1992.

[Gomes *et al.*, 1998] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, WI, 1998.

[Graefe and DeWitt, 1987] Goetz Graefe and David J. DeWitt. The EXODUS optimizer generator. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Francisco, California, 1987.

[Graefe and McKenna, 1993] G. Graefe and W. J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings IEEE International Conference on Data Engineering.*, pages 209–218, Vienna, Austria, April 1993.

[Graefe and Ward, 1989] Goetz Graefe and Karen Ward. Dynamic query optimization plans. *ACM SIGMOD RECORD*, 18(2), June 1989. Also published in/as: 19 ACM SIGMOD Conf. on the Management of Data, (Portland OR), May.-Jun.1989.

[Graefe *et al.*, 1994] Goetz Graefe, Richard L. Cole, Diane L. Davison, William J. McKenna, and Richard H. Wolniewicz. Extensible query optimization and parallel execution in volcano. In J. C. Freytag, G. Vossen and D. Maier, editor, *Query Processing for Advanced Database Applications*, pages 305–381. Morgan Kaufmann, San Francisco, California, 1994.

[Graefe, 1993] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[Gupta and Nau, 1992] Naresh Gupta and Dana S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2–3):223–254, 1992.

[Haas *et al.*, 1989] L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh. Extensive query processing in starburst. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 18(2):377–388, June 1989.

[Haas *et al.*, 1997] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 276–285, 1997.

[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, California, 1995.

[Hanks and Weld, 1995] Steven Hanks and Daniel S. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificicial Intelligence Research*, 2:319–360, 1995.

[Ioannidis and Christodoulakis, 1991] Yannis E. Ioannidis and Stavros Christodoulakis. On the propagation of errors in the size of join results. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 20(2):268–277, June 1991.

[Ioannidis and Kang, 1990] Yannis Ioannidis and Younkyung Cha Kang. Randomized algorithms for optimizing large join queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 312–321, Atlantic City, NJ, May 1990.

[Jarke and Koch, 1984] Matthias Jarke and Jurgen Koch. Query optimization in database systems. *ACM Computing Surveys*, 16(2):111–152, 1984.

[Johnson, 1990] David S. Johnson. Local optimization and the traveling salesman problem. In M. S. Paterson, editor, *Automata, Languages and Programming: Proc. of the 17th International Colloquium*, pages 446–461. Springer, New York, 1990.

[Kambhampati *et al.*, 1995] Subbarao Kambhampati, Craig A. Knoblock, and Qiang Yang. Planning as refinement search: A unified framework for evaluating the design tradeoffs in partial order planning. *Artificial Intelligence*, 76(1-2), 1995.

[Kambhampati, 1992] Subbarao Kambhampati. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3), 1992.

[Kautz and Selman, 1992] H. Kautz and B. Selman. Planning as satisfiability. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363, Vienna, Austria, August 1992. John Wiley & Sons.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1194–1201, Portland, OR, 1996.

[Knoblock and Ambite, 1997] Craig A. Knoblock and José Luis Ambite. Agents for information gathering. In J. Bradshaw, editor, *Software Agents*. AAAI/MIT Press, Menlo Park, CA, 1997.

[Knoblock *et al.*, 1998] Craig A. Knoblock, Steven Minton, José Luis Ambite, Naveen Ashish, Pragnesh Jay Modi, Ion Muslea, Andrew G. Philpot, and Sheila Tejada. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, WI, 1998.

[Knoblock, 1994a] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2), 1994.

[Knoblock, 1994b] Craig A. Knoblock. Generating parallel execution plans with a partial-order planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, Chicago, IL, 1994.

[Knoblock, 1995] Craig A. Knoblock. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[Knoblock, 1996] Craig A. Knoblock. Building a planner for information gathering: A report from the trenches. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, Edinburgh, Scotland, 1996.

[Koehler *et al.*, 1997] Jana Koehler, Bernard Nebel, Jörg Hoffman, and Yannis Dimopoulos. Extending planning graphs to an ADL subset. In Sam Steel and Rachid Alami, editors, *Proceedings of the Fourth European Conference on Planning (ECP-97): Recent Advances in AI Planning*, volume 1348 of *LNAI*, pages 273–285, Berlin, September24 –26 1997. Springer.

[Koehler, 1998] Jana Koehler. Solving complex planning tasks through extraction of subproblems. In Reid Simmons, Manuela Veloso, and Stephen Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, pages 62–69, Pittsburgh, PA, June7 –10 1998.

[Kushmerick, 1997] Nicholas Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1997.

[Kwok and Weld, 1996] Chung T. Kwok and Daniel S. Weld. Planning to gather information. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, 1996.

[Levy *et al.*, 1995] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the 14th ACM Symposium on Principles of Database Systems*, San Jose, California, 1995.

[Levy *et al.*, 1996a] 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*, Portland, OR, 1996.

[Levy *et al.*, 1996b] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of 22th International Conference on Very Large Data Bases*, Bombay, India, 1996.

[MacGregor, 1988] Robert MacGregor. A deductive pattern matcher. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Saint Paul, Minnesota, 1988.

[Mannino *et al.*, 1988] Michael V. Mannino, Paicheng Chu, and Thomas Sager. Statistical profile estimation in database systems. *ACM Computing Surveys*, 20(3):191–221, 1988.

[Minton *et al.*, 1990] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 17–24, Boston, MA, 1990.

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

[Minton, 1988b] Steven Minton. *Learning Search Control Knowledge: An Explanation-Based Approach*. Kluwer, Boston, MA, 1988.

[Minton, 1990] Steven Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2-3):363–392, 1990.

[Minton, 1992] Steven Minton. Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.

[Minton, 1996] Steven Minton. Automatically configuring constraint satisfaction programs: A case study. *Constraints*, 1(1), 1996.

[Muñoz-Avila, 1998] Hector Muñoz-Avila. *Integrating Twofold Case Retrieval and Complete Decision Replay in CAPlan/CbC*. PhD thesis, University of Kaiserslautern, 1998.

[Muslea *et al.*, 1998] Ion Muslea, Steven Minton, and Craig A. Knoblock. Wrapper induction for semistructured, web-based information sources. In *Proceedings of the Conference on Automated Learning and Discovery Workshop on Learning from Text and the Web*, Pittsburgh, PA, 1998.

[Nau *et al.*, 1995] Dana S. Nau, Satyandra K. Gupta, and William C. Regli. AI planning versus manufacturing-operation planning: A case study. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[Nebel and Koehler, 1995] Bernhard Nebel and Jana Koehler. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, 76((1-2)):427–454, 1995.

[Papadimitriou and Steiglitz, 1977] Christos H. Papadimitriou and Kenneth Steiglitz. On the complexity of local search for the traveling salesman problem. *SIAM*, 6(1):76–83, March 1977.

[Papadimitriou and Steiglitz, 1982] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[Penberthy and Weld, 1992] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 189–197, Cambridge, MA, 1992.

[Pérez, 1996] M. Alicia Pérez. Representing and learning quality-improving search control knowledge. In *Proceedings of the Thirteenth International Conference on Machine Learning*, Bari, Italy, 1996.

[Pirahesh *et al.*, 1992] Hamid Pirahesh, Joseph M. Hellerstein, and Waqar Hasan. Extensible/rule based query rewrite optimization in starburst. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, California, June 1992.

[Ratner and Pohl, 1986] Daniel Ratner and Ira Pohl. Joint and LPA*: Combination of approximation and search. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, 1986.

[Roth and Schwarz, 1997] Mary Tork Roth and Peter M. Schwarz. Don't scrap it, wrap it! A wrapper architecture for legacy data sources. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 266–275, 1997.

[Russell and Norvig, 1995] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[Savage *et al.*, 1976] Sam Savage, Peter Weiner, and A. Bagchi. Neighborhood search algorithms for guaranteeing optimal traveling salesman tours must be inefficient. *Journal of Computer and System Sciences*, 12(1):25–35, February 1976.

[Schürr, 1990] Andy Schürr. Introduction to PROGRES, an attribute graph grammar based specification language. In M. Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, volume 411 of *Lecture Notes in Computer Science*, pages 151–165, 1990.

[Schürr, 1996a] Andy Schürr. *Programmed Graph Replacement Systems.* World Scientific, 1996.

[Schürr, 1996b] Andy Schürr. Programmed graph transformations and graph transformation units in GRACE. In J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg, editors, *Proc. Fifth Intl. Workshop on Graph Grammars and Their Application to Comp. Sci.*, volume 1073 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 1996.

[Selman *et al.*, 1992] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, San Jose, California, July 1992. AAAI Press.

[Silberschatz *et al.*, 1997] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts.* McGraw-Hill, third edition edition, 1997.

[Simon, 1969] Herbert Simon. *The sciences of the artificial.* MIT Press, 1969 (2nd edition 1981), 247 pages, 1969.

[Slaney and Thiébaux, 1996] John Slaney and Sylvie Thiébaux. Linear time near-optimal planning in the blocks world. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1208–1214, Menlo Park, August 1996. AAAI Press / MIT Press.

[Swami and Gupta, 1988] Arun Swami and Anoop Gupta. Optimization of large join queries. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 17(3):8–17, September 1988.

[Swami, 1989] Arun Swami. Optimization of large join queries: Combining heuristic and combinatorial techniques. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 367–376, Portland, Oregon, May 1989.

[Tate, 1977] Austin Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–893, Cambridge, MA, 1977.

[Tork Roth *et al.*, 1996] Mary Tork Roth, Manish Arya, Laura M. Haas, Michael J. Carey, William Cody, Ron Fagin, Peter M. Schwarz, John Thomas, and Edward L. Wimmers. The garlic project. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2):557–558, 1996.

[Ullman, 1997] Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of the Sixth International Conference on Database Theory*, Delphi, Greece, January 1997.

[Veloso *et al.*, 1990] Manuela M. Veloso, M. Alicia Perez, and Jaime G. Carbonell. Nonlinear planning with parallel resource allocation. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 207–212, San Diego, CA, 1990.

[Veloso, 1994] Manuela Veloso. *Planning and Learning by Analogical Reasoning.* Springer Verlag, December 1994.

[Weiner *et al.*, 1973] P. Weiner, S. L. Savage, and A. Bagchi. Neighborhood search algorithms for finding optimal traveling salesman tours must be inefficient. In *Conference Record of Fifth Annual ACM Symposium on Theory of Computing*, pages 207–213, Austin, Texas, 30 April–2 May 1973.

[Yan and Larson, 1994] Weipeng P. Yan and Per-Åke Larson. Performing group-by before join. In Ahmed K. Elmagarmid and Erich Neuhold, editors, *Proceedings of the 10th International Conference on Data Engineering*, pages 89–101, Houston, TX, February 1994. IEEE Computer Society Press.

[Yan and Larson, 1995] Weipeng P. Yan and Per-Åke Larson. Eager aggregation and lazy aggregation. In Dennis McLeod, Ron Sacks-Davis, and Hans Schek, editors, *Proceedings of 21th International Conference on Very Large Data Bases*, Zurich, Switzerland, 1995.

[Yu and Chang, 1984] C.T. Yu and C.C. Chang. Distributed query processing. *ACM Computing Surveys*, 16(4):399–433, 1984.

[Zweben *et al.*, 1994] Monte Zweben, Brian Daun, and Michael Deale. Scheduling and rescheduling with iterative repair. In *Intelligent Scheduling*, pages 241–255. Morgan Kaufman, San Mateo, CA, 1994.