Flexible and Scalable Query Access Planning using an AI Planner*

José Luis Ambite & Craig A. Knoblock
Information Sciences Institute and
Department of Computer Science
University of Southern California
Marina del Rey, CA 90292, USA
{ambite, knoblock}@isi.edu

Abstract

In [1], we introduced Planning by Rewriting (PBR), a new paradigm for efficient high-quality planning that exploits plan rewriting rules and efficient local search techniques to transform an easy-to-generate, but possibly suboptimal, initial plan into a low-cost plan. In this paper we show in detail the application of the general framework to query planning in distributed environments. As a result we obtain an scalable and flexible query planner for use in an information mediator.

1 Introduction

Planning, the process of generating a network of actions that achieves a desired goal from an initial state of the world, is a problem of considerable practical significance. However planning is computationally hard except for its simplest formulations [3]. Moreover, in many circumstances it is not enough to find any solution plan since the quality of the solution is also critical. In [1] we presented Planning by Rewriting (PBR), a paradigm for efficient high-quality planning based on local search and plan rewriting. In this paper we focus on the application of the general framework to query planning in distributed environments and show how it results in a flexible and scalable query planner.

There are several advantages in using a general domain-independent planner to solve the query planning problem: declarativeness, flexibility, and reuse. First, declarative planners are easier to understand, maintain and evolve. Second, domain independence and uniformity yields flexibility. Different domains can be easily specified, for example, for different data models such as relational and object-oriented. The uniform specification of the planner facilitates its extension with capabilities such as learning mechanisms and interleaving planning and execution. Finally, a general planning architecture fosters reuse in the domain specifications, the search methods and the search control techniques. For example, the specification of the join operator translates straightforwardly from a relational to an object-oriented model. Likewise, search methods and search control mechanisms can be implemented once for the general planner and the most appropriate combination chosen for each particular domain. PBR satisfies these properties while addressing plan quality and planning efficiency.

The remainder of this paper is structured as follows. Section 2 reviews briefly the Planning by Rewriting paradigm. Section 3 presents the problem of query planning in distributed environments and describes how to cast it in the PBR framework. Section 4 shows some encouraging initial results. Section 5 discusses related work. Section 6 discusses future work. Section 7 concludes.

2 Review of Planning by Rewriting

Planning by Rewriting [1] follows the iterative improvement style of many optimization algorithms. The framework works in two phases:

1. Efficiently generate an initial solution plan.
2. Iteratively rewrite the current solution plan in order to improve its quality using a set of plan rewriting rules until either an acceptable solution is found or a resource limit is reached.

In Planning by Rewriting a plan is represented by a graph notation, in the spirit of partial-order causal-link planners such as UCPOP [20]. The nodes are

*Copyright ©1997 IEEE. Published in the Proceedings of KDEX'97, November 4, 1997 in Newport Beach, California. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE. Contact: Manager, Copyrights and Permissions / IEEE Service Center / 445 Hoes Lane / P.O. Box 1331 / Piscataway, NJ 08855-1331, USA. Telephone: + Intl. 908-562-3966.

domain actions. The edges specify a temporal ordering relation among steps, imposed by causal links and ordering constraints.

A plan rewriting rule, akin to term and graph rewriting rules, specifies the replacement under certain conditions of a partial plan by another partial plan. Our system ensures that the rewritten plan remains complete and consistent. These rules are intended to improve the quality of the plans.

An important advantage of PBR is its anytime nature, therefore being able to trade 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 when the cost of the current plan is small enough, even if a cheaper one could be found.

The following is a list of the main issues in Planning by Rewriting. The remainder of the paper discusses these issues in the context of query planning.

- *Efficient generation of an initial solution plan.* In many domains, such as query planning, obtaining a possibly suboptimal initial plan is easy.
- *Definition and application of the plan rewriting rules.* The user can specify appropriate rewriting rules for a domain in a simple, but quite general, rule definition language. These rules are matched against the current plan and generate new transformed plans (possibly of better quality).
- *Plan quality measure.* This is the given plan cost function to optimize.
- *Search of the space of rewritings.* There are many possible ways of searching the space of rewritten plans, for example, gradient descent, simulated annealing, etc.

3 Query Planning

Distributed query processing [22] involves generating a plan that efficiently computes a user query. This 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). This is a highly combinatorial problem in which the number of query evaluation plans grows exponentially with the number of relations in the given query.

Our distributed query processing and integration model follows that of the SIMS mediator system [2, 13]. Briefly, a set of information sources such as databases, knowledge bases, web servers, etc, supply data about a particular application domain. A SIMS mediator integrates and provides a single point of access for all the information in such a domain. The user interacts directly with the SIMS mediator without knowledge about the schemas and locations of the sources. Such is the task of the query planner. The encoding of information goals and the operator specification appear in [10, 12].

A sample information goal for this domain is shown in Figure 1. This goal asks to send to the output device of the SIMS mediator all the codes of ships that can dock in the port of Tabarka. The query is written in the Loom query language [16] which is used in the SIMS mediator system. Figure 2 shows a plan that computes this query by accessing three different information sources (namely, two replicas of the source `geo`, one at `isd18.isi.edu` and another at `higgledy.isi.edu`, and the source `assets` at `higgledy.isi.edu`).

```
(available output sims
    (sims-retrieve (?id_code)
         (:and (seaports ?port)
               (seaports.port_nm ?port "Tabarka")
               (seaports.glc_cd ?port ?glc_cd)
               (channels ?channel)
               (channels.glc_cd ?channel ?glc_cd)
               (channels.ch_depth_ft ?channel ?depth)
               (chstr_ship ?ship)
               (chstr_ship.min_draft ?ship ?min_draft)
               (chstr_ship.sht_cd ?ship ?id_code)
               (< ?min_draft ?depth))))
```

Figure 1: A Sample Information Goal

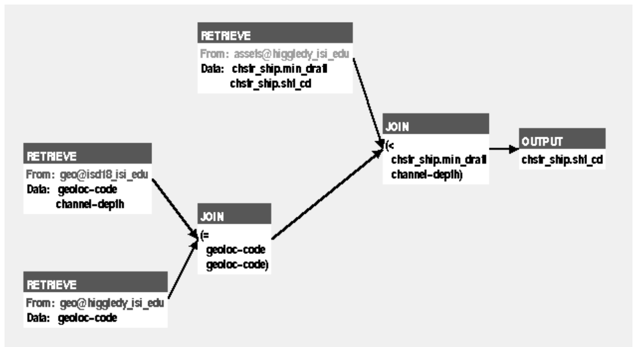

Figure 2: A Sample Query Plan

3.1 Generation of an Initial Query Plan

For PBR 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. These initial plans although correct may be of very low quality.

A (randomly generated) initial plan for the query in Figures 1 and 2 is shown in Figure 3. Note how it introduces an inefficient cross-product and access the same source (`geo@higgledy.isi.edu`) twice, while the plan in Figure 2 maximizes the parallelism among the three databases.

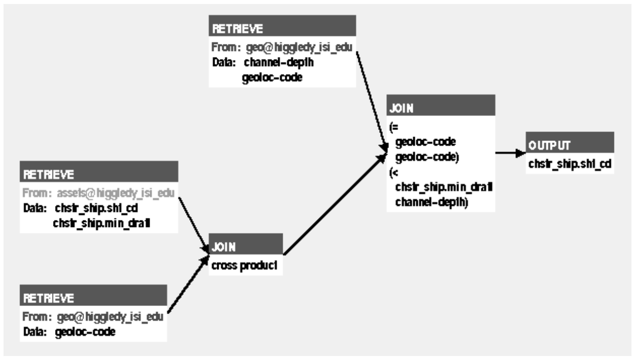

Figure 3: A Suboptimal Initial Query Plan

3.2 Rewriting Rules for Query Planning

In query planning domains the plan rewriting rules can be easily derived from the algebraic properties of the data definition language and the characteristics of the computational environment. In this section we present a description of a representative set of the rules for our distributed query planning domain. The algebraic basis of the rules is shown in Figure 4 and their syntax in the PBR rule language is shown in Figures 5, 6, 7, 8, 9, 10, and 11.

Remote-Join-Eval: This rule is an instance of the fact that when a group of operators can be executed together at a remote information source it is generally more efficient to do so. In particular, if a source has join capabilities, it is more efficient to compute a join at the source than transmit the data over the network and compute the join at the local system. This suggests the rewriting rule in Figure 5, which specifies that if in a plan there exist two retrieve operators from the same remote database which are consequently joined, the system can rewrite the plan into one that contains a unique retrieve operation that pushes the join operation to the remote database.

Remote-Selection-Eval: Similarly to the previous rule, it is generally beneficial to execute selections remotely. Note the need for checking the capabilities of the information sources. We do not assume that sources are full databases. They may have no query processing capabilities (for example, wrapped WWW pages) or support very limited queries (for example WWW forms). The corresponding rewriting rule, shown in Figure 6, specifies that if in a plan there is a `?query2` which is decomposed in a local `?selection` performed on the result of a remote retrieval of `?query1` from `?source` at `?server`, it may be more efficient to evaluate `?query2` remotely at `?source`.

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

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

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

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

Push-In-Double-Selection:
$\sigma_A(Q1 \bowtie Q2) \Rightarrow \sigma_{A'} Q1 \bowtie \sigma_{A''} Q2$

Push-Out-Double-Selection:
$\sigma_{A'} Q1 \bowtie \sigma_{A''} Q2 \Rightarrow \sigma_A(Q1 \bowtie Q2)$

Replicated-Source-Swap:
$retrieve(Q, Src, Svr1) \wedge Svr2 \neq Svr1$
$\Rightarrow retrieve(Q, Src, Svr2)$

Figure 4: Transformations in Query Planning

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

Figure 5: Remote-Join-Eval Rewriting Rule

```
(define-rule :name remote-selection-eval
  :if (:operators
      ((?n1 (retrieve ?query1 ?source ?server))
       (?n2 (select ?query2 ?selection ?query1)))
      :constraints ((capability ?source 'selection)))
  :replace (:operators (?n1 ?n2))
  :with (:operators
         ((?n3 (retrieve ?query2 ?source ?server)))))
```

Figure 6: Remote-Selection-Eval Rewriting Rule

Join-Swap: This rule specifies the reordering of two consecutive joins operators (Figure 7). It is derived from the associative and commutative properties of the relational algebra join operation. This rule allows the system to explore the space of join trees. In the query planning domain of [10] queries are expressed as complex terms. The PBR rules use the interpreted predicates in the `constraints` field to manipulate such query expressions. 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) and produces as output the description of the two reordered join operations (as bindings for the last eight variables). If two subqueries do not share any attributes, the join degenerates into a cross-product. Although a cross-product is inefficient, such rewritings are allowed for completeness.

```
(define-rule :name join-swap
  :if (:operators ((?n1 (join ?q1 ?jc11 ?sq1a ?sq1b))
                   (?n2 (join ?q2 ?jc2 ?sq2a ?sq2b)))
       :links (?n1 ?n2)
       :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)))))
```

Figure 7: Join-Swap Rewriting Rule

Single-Selection-Swap: This rule moves bidirectionally a selection operator through a join when the selection conditions are applicable to only one of the subqueries of the join (Figure 8). Note that a selection can involve attributes present in both subqueries on a join. This case in handled by the next two rules. The reason for having three rules to cover the rewritings involving the selection and join operators stems from the choice of our rule rewriting language. In order to keep rule matching efficient we decided to allow only conjunctive expressions in the antecedent and consequent of the rules. As a result currently the rules cannot express a variable number of operators to match in the antecedent (or being generated in the consequent) which would require a universal quantification capability. This is the case in attempting to push a selection through a join in which it is not know a priori if the selection will appear in one or both subqueries of the join. Our solution is to specify one rule for each case.

```
(define-rule :name single-selection-swap
  :if (:operators ((?n1 (join ?q1 ?jc ?sq1a ?sq1b))
                   (?n2 (select ?q2 ?sc ?sq2)))
      :constraints
      (selection-swappable ?q1 ?jc ?sq1a ?sq1b ; in
                           ?q2 ?sc ?sq2        ; in
                           ?q3 ?sq3a ?sq3b     ; out
                           ?q4 ?sq4))          ; out
  :replace (:operators (?n1 ?n2))
  :with (:operators ((?n3 (join ?jc ?q3 ?sq3a ?sq3b))
                     (?n4 (select ?sc ?sq4 ?q4)))))
```

Figure 8: Single-Selection-Swap Rewriting Rule

Push-In-Double-Selection: This rule moves a selection operator into both subqueries of the join if appropriate (Figure 9). Note that it will split the selection conditions into the attributes present in each subquery. This transformation is based on the fact that is generally beneficial to push the selections as deeply into the query (that is, as closer to the base relations) as possible in order to produce the least amount of data for latter stages of processing.

```
(define-rule :name push-in-double-selection
  :if (:operators ((?n1 (join ?q1 ?jc ?sq1a ?sq1b))
                   (?n2 (select ?q2 ?sel ?q1))))
      :constraints
      (pushable-in-2sel ?q1 ?jc ?sq1a ?sq1b ; in
                        ?q2 ?sel            ; in
                        ?sq2a ?sel-2a       ; out
                        ?sq2b ?sel-2b))     ; out
  :replace (:operators (?n1 ?n2))
  :with (:operators ((?n3 (join ?q2 ?jc ?sq2a ?sq2b))
                     (?n4 (select ?sq2a ?sel-2a ?sq1a))
                     (?n5 (select ?sq2b ?sel-2b ?sq1b)))))
```

Figure 9: Push-In-Double-Selection Rewriting Rule

Push-Out-Double-Selection: This rule moves two selection operators that occur before a join and condenses them into a unique selection operator to be performed after the join (Figure 10). We add this rule for completeness despite that in most cases the previous rule, its inverse, leads more often to cheaper plans.

Replicated-Source-Swap: This rule covers the case in which there are several replicated information sources that can satisfy some query (Figure 11). Each source may have different retrieval or transmission costs associated. This rule allows the

```
(define-rule :name push-out-double-selection
  :if (:operators ((?n1 (select ?q1 ?sel-1 ?sq1))
                   (?n2 (select ?q2 ?sel-2 ?sq2))
                   (?n3 (join ?q3 ?jc ?q1 ?q2)))
       :links ((?n1 ?n3) (?n2 ?n3))
       :constraints
       (pushable-out-2sel ?q3 ?jc ?q1 ?q2  ; in
                          ?sel-1 ?sq1      ; in
                          ?sel-2 ?sq2      ; in
                          ?sel-4 ?q4))     ; out
  :replace (:operators (?n1 ?n2))
  :with (:operators ((?n4 (join ?q4 ?jc ?sq1 ?sq2))
                     (?n5 (select ?q3 ?sel-4 ?q4)))))
```

Figure 10: Push-Out-Double-Selection Rewriting Rule

system to explore the choice of such alternative sources. Note that 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 could repair the plan and complete the execution without replanning and re-executing from scratch.

```
(define-rule :name replicated-source-swap
  :if (:operators
       ((?n1 (retrieve ?query ?source ?server1)))
       :constraints ((server-available ?source ?server2)
                     (:neq ?server1 ?server2)))
  :replace (:operators (?n1))
  :with (:operators
         ((?n2 (retrieve ?query ?source ?server2)))))
```

Figure 11: Replicated-Source-Swap Rewriting Rule

The modular and declarative rule specification yields a very flexible planner. The capabilities of the planner can be extended incrementally by specifying more rules. For example, the rules above deal with conjunctive queries. Similar rules can be specified to handle relationally complete queries, queries with binding patters, or different capabilities of the sources.

Although it may be easy to add new rules, this comes at the cost of a more expensive match and an increased search space. This is a version of the utility problem. Defining all the rules necessary to completely explore the space of rewritings may not be cost effective. As an optimization, the system could keep the subset of the rules that is most useful in leading the system towards high-quality plans.

3.3 Query Plan Quality

In Planning by Rewriting the user has to provide a measure of plan quality suitable for the application domain. For our query planning domain the quality of a plan is 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 between the remote sources and 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 number of tuples in a relation, the number of distinct values for each attribute, and the maximum and minimum values for numeric attributes. More sophisticated estimation models are surveyed in [17].

3.4 Search Strategy

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. 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 these cases simulated annealing may be more appropriate than strict gradient descent.

We have explored gradient descent techniques, such as first improvement and steepest descent. In *first improvement*, the next plan to consider is the first rewriting that improves the cost. This has the advantage that the neighborhood is generated only up to the point such a plan is found, but the improvement may not be the best that could be achieved in that neighborhood. In *steepest descent*, the minimum cost plan within the neighborhood is chosen. This guarantees the maximum improvement in cost in each iteration, but it requires the whole neighborhood to be searched.

In general, the space of rewritings and the cost functions are not convex, thus our gradient descent techniques can get caught in local minima. To move towards the optimum escaping low-quality local minima, we used two techniques: restart and random walk. In the first one, the system restarts the rewriting process a fixed number of times from a different initial plan. This technique requires an initial plan generator that is able to provide several different/random initial plans. The second technique is applied when the local minima are not strict and consists of a random walk of a fixed length along the plateau.

4 Initial Results in Query planning

We used a simplified domain for the query planner (Sage) of the SIMS mediator. We compare the performance and quality of the Sage planner and PBR for this domain, where the query plans are trees of join operations.

Sage performs a best-first search with a heuristic commonly used in query optimization that explores only the space of left join trees (Sage-BFS). For PBR, we defined the `join-swap` rule of Figure 7. The initial plans were random depth-first search parses of the query (Sage-DFS). PBR performs a steepest descent search. To escape local minima, PBR generates and rewrites three random initial plans and picks the best rewriting.

We generated a synthetic integration domain for the SIMS mediator and tested the three planners with conjunctive queries involving from 1 to 15 relations. The results are shown in Figures 12 and 13. Figure 12 shows the planning time in a logarithmic scale. The times for PBR includes both the generation of the three random initial plans and their rewriting. The times for Sage-DFS are the average of the three random initial plans. Sage-BFS is able to solve queries involving up to 6 relations, but larger queries are intractable for it within the limit of 200,000 nodes. PBR scaled better and solved all queries with low cost plans.[1]

Figure 13 shows the quality of the query plans for Sage-BFS, Sage-DFS and PBR. A logarithmic scale is used because the large absolute values of the plan costs. The graph shows that Sage-DFS produces very poor quality initial plans. However PBR rewrites them into plans of quality close to those produced by Sage-BFS (often better because PBR searches the larger space of bushy trees) in the range tractable for Sage-BFS and beyond that range it scales gracefully.

[1]Although PBR scaled better than Sage its cost increases considerably in the larger queries. We believe this is an artifact of the current implementation more than a problem of the framework. PBR is implemented on top of UCPOP and reuses much of its code. In particular to check plan consistency after each rule application we used the same routines as UCPOP which are optimized for a causal-link partial-order refinement planner, not a rewriting planner. We expect that a more appropriate implementation would provide significantly better results.

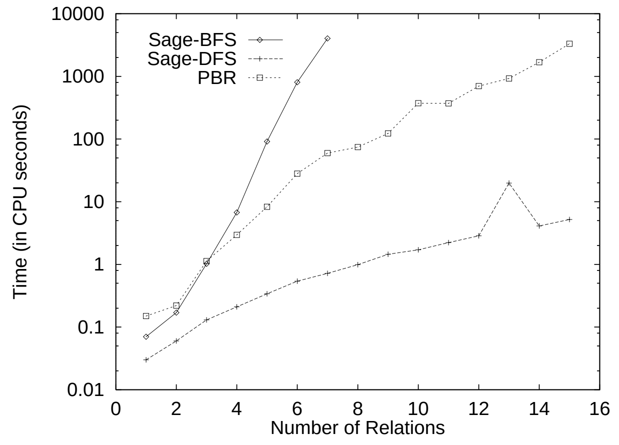

Figure 12: Planning Time Comparisons

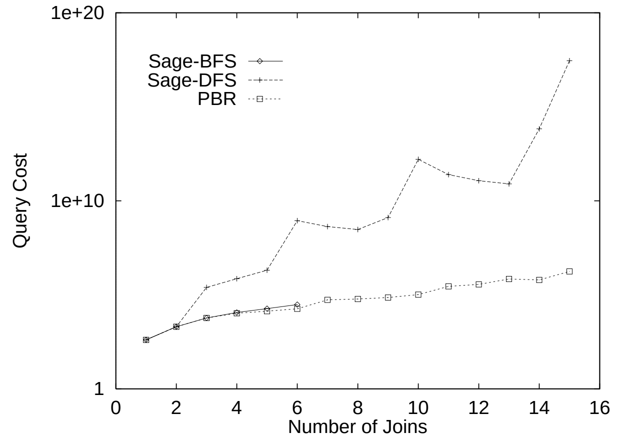

Figure 13: Plan Quality Comparisons

5 Related work

In the database literature, query optimization has been extensively studied [9]. Query optimizers need both to find the most efficient algebraic form of a query and to choose specific methods to implement each data processing operation [7]. For example, a join can be performed by a variety of algorithms, such as nested loops, merge scan, hash join, etc. In the present paper 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 remote databases, and so far the size of data the mediator needs to manipulate has not required very sophisticated consideration of implementation algorithms.

There are two types of research on query optimization that are most relevant to our approach. The first one are query optimizer generators, some of which accept declarative specifications. Exodus [6] is a query optimizer generator that compiles a query optimizer out of a set of operators, transformation rules and the code for the methods that implement each operator. 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 focuses more in implementation methods and it operates on centralized databases. Volcano [5], a successor of Exodus, provides a general implementation of data processing operations based on iterators, but do not offer more generality on algebraic query optimization. Interestingly, it provides a simple form of contingency planning in which two alternative implementation methods are included in the plan and which one is chosen depends on the value of a parameter at execution time. We expect that PBR would be able to provide more general interleaving of planning and execution [11]. The second type of work is in efficient search algorithms for query planners [21, 8]. Our approach being based in a domain-independent planner is more flexible than previous research allowing the analysis of different domains (operators, rewriting rules) and search algorithms in an uniform and easily extensible framework.

Despite the importance of query planning, there has been little work in the planning literature. Occam [15] is a planner for information gathering, but it does not specifically address plan quality. Sage [12] considers plan quality and supports interleaving of planning and execution. PBR does not currently interleaves currently planning and execution, but it has the capability of being as general as Sage with better scaling properties.

The framework of Planning by Rewriting is related to several pieces of previous work in AI planning. Most significantly it is a generalization of plan merging [4] and it follows on iterative repair ideas such as those in [18] and [23]. The reader is referred to [1] for a more detailed discussion of the relationship with AI planning.

Finally, PBR can be understood as an instantiation of the local search idea, which has a long tradition in combinatorial optimization [19]. However, instead of hard-coding a specific algorithm for each problem, PBR provides a general framework in which a problem is cast as a declarative planning specification and a set of declarative rewriting rules, while the bulk of the program consist of the rewriting and search engine.

6 Future Work

Query planning is an excellent domain to test the limits of our Planning by Rewriting framework. We plan to extend the capabilities of PBR in several dimensions: more sophisticated query planning domains, interleaving of planning and execution, insertion of information gathering actions, and new search methods. More complex query planning domains will serve both to test the expressiveness of our rule definition language and to compare with query optimizers in the database literature. Interleaving planning and execution is necessary in order to deal effectively with unexpected situations in the environment such as database or network failures. It also enables the planner to perform dynamic query optimization, in which plans depend on run-time conditions, and the insertion of information gathering actions [14].

We plan to explore a variety of search techniques for query planning, for example, variable depth rewriting. In variable depth search a sequence of rewritings is applied atomically. This allows the planner to overcome initial cost increases that eventually would lead to strong cost reductions. This idea leads to the creation of rule programs which is particularly appealing in the query planning domain in which sequences of rewritings are natural. For example, a sequence of `join-swap` transformations may put two joins in the same database together in the query tree and then `remote-join-eval` collapse the explicit join operators into one retrieval of a remote join.

7 Conclusions

We have presented a scalable and flexible query planner for distributed environments based on a general planning framework: Planning by Rewriting. PBR efficiently produces high-quality plans by transforming an easy-to-generate, but possibly suboptimal, initial plan into a low-cost plan, using plan rewriting rules and local search techniques. This efficient search makes PBR scalable to large queries. Being based on a declarative domain-independent planner, PBR is very flexible and extensible: new query planning domains (operators and rewriting rules) can be easily specified, new search mechanisms can be incorporated, and novel features such as the interleaving of planning and execution dealt with in a uniform and general manner.

Acknowledgments

The research reported here was supported in part by Rome Laboratory of the Air Force Systems Command and the Advanced Research Projects Agency under contract no. F30602-94-C-0210, and in part by the National Science Foundation under grant number IRI-9313993.

References

[1] 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.

[2] 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.

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

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

[5] G. Graefe, R. L. Cole, D. L. Davison, W. J. McKenna, and R. 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*, page 305. Morgan Kaufmann, San Francisco, CA, 1994.

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

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

[8] 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.

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

[10] Craig A. Knoblock. Applying a general-purpose planner to the problem of query access planning. In *Proceedings of the AAAI Fall Symposium on Planning and Learning: On to Real Applications*, New Orleans, La, 1994.

[11] 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.

[12] 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.

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

[14] Craig A. Knoblock and Alon Levy. Exploiting runtime information for efficient processing of queries. In *Working Notes of the AAAI Spring Symposium on Information Gathering in Heterogeneous, Distributed Environments*, Palo Alto, CA, 1995.

[15] Chung T. Kwok and Daniel S. Weld. Planning to gather information. Technical Report UW-CSE-96-01-04, Department of Computer Science and Engineering, University of Washington, 1996.

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

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

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

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

[20] 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.

[21] Arun N. 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, OR, May 1989.

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

[23] 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.