ELSEVIER

Artificial Intelligence 118 (2000) 115–161

Artificial Intelligence

www.elsevier.com/locate/artint

Flexible and scalable cost-based query planning in mediators: A transformational approach

José Luis Ambite *, Craig A. Knoblock

Information Sciences Institute and Department of Computer Science, University of Southern California, 4676 Admiralty Way, Marina del Rey, CA 90292, USA

Received 30 October 1998; received in revised form 11 October 1999

Abstract

The Internet provides access to a wealth of information. For any given topic or application domain there are a variety of available information sources. However, current systems, such as search engines or topic directories in the World Wide Web, offer only very limited capabilities for locating, combining, and organizing information. Mediators, systems that provide integrated access and database-like query capabilities to information distributed over heterogeneous sources, are critical to realize the full potential of meaningful access to networked information.

Query planning, the task of generating a cost-efficient plan that computes a user query from the relevant information sources, is central to mediator systems. However, query planning is a computationally hard problem due to the large number of possible sources and possible orderings on the operations to process the data. Moreover, the choice of sources, data processing operations, and their ordering, strongly affects the plan cost.

In this paper, we present an approach to query planning in mediators based on a general planning paradigm called Planning by Rewriting (PbR) (Ambite and Knoblock, 1997). Our work yields several contributions. First, our PbR-based query planner combines both the selection of the sources and the ordering of the operations into a single search space in which to optimize the plan quality. Second, by using local search techniques our planner explores the combined search space efficiently and produces high-quality plans. Third, because our query planner is an instantiation of a domain-independent framework it is very flexible and can be extended in a principled way. Fourth, our planner has an anytime behavior. Finally, we provide empirical results showing that our PbR-based query planner compares favorably on scalability and plan quality over previous approaches, which include both classical AI planning and dynamic-programming query optimization techniques.

Keywords: Query optimization; Planning by Rewriting; Information integration

* Corresponding author. Email: ambite@isi.edu.

0004-3702/00/$ – see front matter
PII: S0004-3702(00)00003-5

1. Introduction

The Internet and the World Wide Web offer an ever-increasing amount of information. Beyond providing mere access to information, a much greater value lies on systems that integrate information from different sources and provide sophisticated query capabilities on the integrated data. Such systems, known as mediators [68], are the key enabling technology for many far-reaching applications such as comparison shopping on the Internet, integrated supply chain management, or integrated medical records.

Query planning lies at the core of mediator systems. Query planning in mediators consists of selecting the relevant information sources for a user query and generating a cost-efficient network of actions that retrieves and processes the required information. In addition to being practically important, this problem constitutes a challenging testbed for planning technology. First, it is a highly combinatorial problem. Query plans not only must incorporate the relevant sources among hundreds of available information sources, but also find an appropriate ordering of the data processing and retrieval operations. Second, finding any valid plan is not enough, plan quality is also critical. The query planner must be able to address complex quality criteria, such as execution time, monetary cost, plan robustness, etc. Third, query plans often have to be produced rapidly. In many cases the utility of a plan decreases with time, thus a trade-off between planning time and plan quality is desirable. 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.

In this paper, we present an approach to query planning in mediators that addresses these challenges. Our query planner is based on a general paradigm for efficient high-quality planning called Planning by Rewriting (PbR) [4]. This planning style uses 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. PbR was designed to address planning efficiency and plan quality, while providing the benefits of flexibility and extensibility that domain-independence affords.

The characteristics of PbR make it especially well-suited for query planning. First, our PbR-based query planner is *scalable*. By using local search techniques, our query planner scales gracefully to a large numbers of sources and large query plans, as demonstrated by the empirical results in Section 6. In spite of the complexity of query planning our system produces high 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 when the cost of executing the current plan is small enough, even if a cheaper plan 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 source selection and traditional cost-based query optimization. Previous work either has not addressed cost-based optimization (e.g., [18]), or has performed these two types of query processing in two sequential stages (e.g., [36]). That is, by first finding all the plans that incorporate the relevant sources, and then optimizing the cost of each of these plans to finally choose the best. Given that both finding the relevant sources and optimizing the cost of a query plan are combinatorial, the two-stage approach 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 Web. Our PbR-based query planner 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.

The application of PbR to query planning in mediators, resulting in a flexible and scalable planner that combines source selection and cost-based optimization, is the contribution of this work. The remainder of the article is organized as follows. First, we introduce the challenging domain of query planning in mediators and present an overview of the architecture of the SIMS mediator [7,8]. Second, we motivate modeling query planning as an instance of classical domain-independent planning and show an encoding of query planning as a classical planning problem (following [41]). Third, we briefly review the general Planning by Rewriting framework. Fourth, we describe in detail how query planning is performed within the Planning by Rewriting framework and the main components of our PbR-based query planner. Fifth, we show the results of several scalability experiments comparing PbR with classical AI planning and dynamic programming optimization techniques. Sixth, we relate our approach with previous work. Finally, we discuss future work and conclude.

2. Integration and query planning in the SIMS mediator

Mediators provide access, in a given application domain, to information that resides in distributed and heterogeneous sources [68]. 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.

Mediators were initially developed to integrate structured information sources, such as databases [8,48,63]. Many 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 either programmed by hand [33,57] or learned automatically [43,52]. Also, semantic mark-up languages such as XML can facilitate considerably the extraction of information from Web sources [15]. For the purposes of this paper we consider the sources to have a well-defined schema, be it because the sources are databases or because they are wrapped Web 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 [37]. 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 this section we present an overview of the integration and query planning architecture of the SIMS mediator. In Sections 3, 4, and 5 we describe the combination of source selection and cost-based optimization using our rewriting approach.

2.1. Integration model and axiom compilation in SIMS

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 (that we will call the *domain model*), 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, sometimes called local-as-view, is to define each source term as a logical formula over terms of the global domain model [18,44,47]. Another approach, correspondingly called global-as-view, is to define each domain term as a formula over source terms [2,32,33]. These two approaches have complementary strengths and weaknesses [64]. The local-as-view 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 hard for many languages [17,46]. Conversely, the global-as-view approach facilitates the query processing, which is reduced to unfolding, that is, the terms in the user query, which are domain terms, are simply substituted by the formula of source terms given in the integration statement. However, adding new sources may involve extensive changes to the mediator definitions.

In our mediators, SIMS [8] and Ariadne [42], we combine the strengths of both approaches by defining source terms as formulas on the global model (local-as-view statements) and *precompiling* a set of inverse formulas (domain terms as a combination of source terms, global-as-view statements) before any query planning starts. [1] We will call our precompiled inverse formulas *integration axioms* for the remainder of the paper. During

[1] However, our local-as-view statements are more restricted than those of [17,46], as we discuss at the end of this subsection.

query planning our system only uses the integration axioms. This allows our system to plan more efficiently by unfolding, but still accept new source definitions without manually restructuring the domain model.

SIMS domain model. In SIMS, the domain model is specified in a subset of Loom [50], which is a KL-ONE style knowledge representation language [10]. KL-ONE style languages, also known as description logics, contain unary relations (*classes*), which represent the classes of the objects in the domain, and binary relations (*attributes*), which describe relationships between objects. Classes are defined using a set of class constructors. We list the constructors supported in the SIMS language below, giving examples using the simple application domain about seaport information shown in Fig. 1.

- **Primitive:** A primitive class is defined as a subclass without specifying the constraints that differentiate it from the parent class.
- **Defined:** A class can be defined using the following class constructors:
  - **Attribute Introduction:** A class C can be defined as having an attribute R relating class C to another class D. For example, in Fig. 1 the class seaport is defined as having the attributes: geographic code (gc), port name (pn), country name (cn), and number of cranes (cr).
  - **Conjunction:** A class can be specified as a conjunction of other classes.
  - **Disjunction (Covering):** A class can be specified as a disjunction of its subclasses. For example, the class seaport in Fig. 1 is the union of its subclasses: large-seaport and small-seaport.
  - **Equality and Order Constraints:** A class can be specialized by conjoining it with order constraints of the type: attribute θ constant, where $\theta \in \{=, <, \leqslant, >, \geqslant\}$. For example, large-seaport in Fig. 1 is defined as a seaport with more than seven cranes.

A set of attributes is associated with each class, and any subclass of a given class, C, inherits all of the attributes of C. For purposes of integration, we also require that every class has at least one defined key. A key is one or more attributes that uniquely identify the objects in a class. Since there may be more than one way to uniquely identify an object, a class may have multiple keys. In the hierarchy of Fig. 1, gc and pn are two alternative keys for seaport and its subclasses.

SIMS source descriptions. In SIMS the domain model is used to describe the information sources. The source description for source S with attributes $S.a_1, \ldots, S.a_n$ is written:

$$S(S.a_1, \ldots, S.a_n) = D_S(a_1, \ldots, a_n).$$

The equation above specifies that the source S provides all instances of the domain class D_S and values for the corresponding attributes. Fig. 1 provides an example of sources and mappings to domain classes. For instance, there are three sources for the class large-seaport: s2 that provides the attribute pn, s3 that provides gc, and s7 that provides both pn and cn. For simplicity, each information source in Fig. 1 is assumed to contain a single table of the same name, but SIMS supports sources with multiple tables.

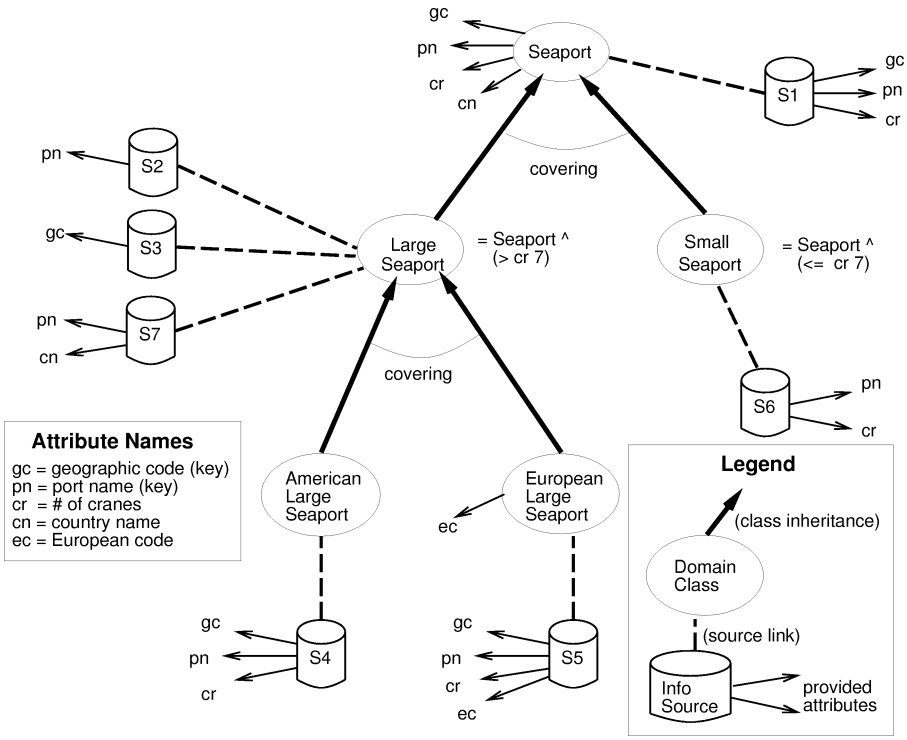


Fig. 1. Example domain model.

In contrast to other approaches to information integration (e.g., [18,48]), which use *containment* to express the relationship between a source and its description in domain terms, SIMS uses *equality*. That is, SIMS assumes that the source description defines exactly the class of information provided by the sources. 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. The use of exact descriptions has two major advantages: it supports complete answers to queries, and, when complete answers are not possible, it allows the system to determine when and in what way the answer is incomplete. For example, the system could signal that there are no sources for the class of information required in the query, but that data for a superclass or a subclass is available.

SIMS axiom compilation. SIMS precompiles the local-as-view source descriptions into a set of maximal global-as-view integration axioms to facilitate query processing. An integration axiom specifies one way of combining a set of sources in order to provide a set of attributes for a given domain class. These axioms are maximal in the sense that they specify the maximum number of attributes for a single class that can be obtained from each combination of sources. The relevant axioms for the classes and attributes requested in a user query can be efficiently computed by instantiating these maximal axioms at run time.

seaport(pn)	$\Leftrightarrow$	s2(pn) $\vee$ s6(pn)	*2.1*
	$\Leftrightarrow$	s6(pn) $\vee$ s7(pn)	*2.2*
seaport(cr pn)	$\Leftrightarrow$	s4(cr pn) $\vee$ s5(cr pn) $\vee$ s6(cr pn)	*2.3*
seaport(cr gc pn)	$\Leftrightarrow$	s1(cr gc pn)	*1.1*
small-seaport(cr pn)	$\Leftrightarrow$	s6(cr pn)	*1.2*
small-seaport(cr gc pn)	$\Leftrightarrow$	s1(cr gc pn) $\wedge$ cr ≤ 7	*3.1*
	$\Leftrightarrow$	s1(cr gc pn) $\wedge$ s6(cr pn)	*4.1*
large-seaport(gc)	$\Leftrightarrow$	s3(gc)	*1.3*
large-seaport(pn)	$\Leftrightarrow$	s2(pn)	*1.4*
large-seaport(cn pn)	$\Leftrightarrow$	s7(cn pn)	*1.5*
large-seaport(cr gc pn)	$\Leftrightarrow$	s4(cr gc pn) $\vee$ s5(cr gc pn)	*2.4*
	$\Leftrightarrow$	s1(cr gc pn) $\wedge$ cr > 7	*3.2*
	$\Leftrightarrow$	s1(cr gc pn) $\wedge$ s3(gc)	*4.2*
	$\Leftrightarrow$	s1(cr gc pn) $\wedge$ s2(pn)	*4.3*
large-seaport(cn cr gc pn)	$\Leftrightarrow$	s1(cr gc pn) $\wedge$ s7(cn pn)	*4.4*
	$\Leftrightarrow$	[s4(cr gc pn) $\wedge$ s7(cn pn)] $\vee$ [s5(cr gc pn) $\wedge$ s7(cn pn)]	*5.1*
american-large-seaport(cr gc pn)	$\Leftrightarrow$	s4(cr gc pn)	*1.6*
american-large-seaport(cn cr gc pn)	$\Leftrightarrow$	s4(cr gc pn) $\wedge$ s7(cn pn)	*4.5*
european-large-seaport(cr ec gc pn)	$\Leftrightarrow$	s5(cr ec gc pn)	*1.7*
european-large-seaport(cn cr ec gc pn)	$\Leftrightarrow$	s5(cr ec gc pn) $\wedge$ s7(cn pn)	*4.6*

Fig. 2. Compiled integration axioms.

The compiled axioms for the sample domain in Fig. 1 are shown in Fig. 2. The integration axioms have two parts. The first, that we call the axiom head, is the domain class and accompanying attributes left of the biconditional. The second, that we call the axiom body, is the logical formula, right of the biconditional, that specifies a combination of source classes that provide the attributes for the domain class in the axiom head. The interpretation of the axioms in Fig. 2 is quite intuitive. For example, consider axiom *2.4*. Inspecting the domain in Fig. 1, it can be readily seen that large-seaport is the union of american-large-seaport and european-large-seaport, which have sources s4 and s5, respectively. As these two sources have attributes (cr gc pn) in common, the union of s4 and s5 provides (cr gc pn) for large-seaport (attribute ec only appears in source s5 so we cannot find all values of ec for large-seaport). Another example of compilation is axiom *3.2*. As the model states that the large seaports are those with more than seven cranes and that all seaports can be found in source s1, to find large-seaport(cr gc pn) it is enough to apply the filter cr > 7 to the data in source s1.

The body of the axioms in Fig. 2 are formulas in source terms, but as our source definitions are equivalences, we can also write them in domain terms. For example, the domain level version of axiom *4.2* is:

$$\text{large-seaport(cr gc pn)} \Leftrightarrow \text{seaport(cr gc pn)} \wedge \text{large-seaport(gc)}$$

As we will see in Section 5 this encoding is quite useful when there are many alternative sources for a given domain class and set of attributes since it provides a layer of abstraction over the particular sources chosen to implement an axiom.

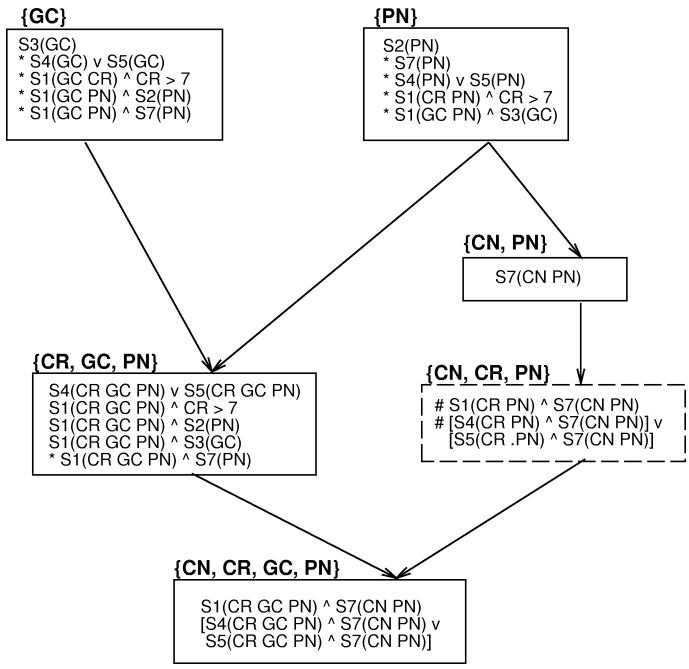


Fig. 3. Axiom lattice for large-seaport.

The axioms for each class are organized in a lattice to facilitate efficient access during query processing. The order in the lattice is set containment of attributes. As an example consider the axiom lattice for the large-seaport class shown in Fig. 3. Note how the axioms for large-seaport in Fig. 2 have been transferred to the lattice. The lattice of Fig. 3 shows two additional sets of axioms, which are derived from the basic axioms in Fig. 2. The first set are the supplementary axioms. For each node in the lattice all possible alternative axioms must be computed. This may involve projecting axioms from nodes providing a larger attribute set. In Fig. 3 the supplementary axioms are marked with an asterisk (*). For example, the supplementary axioms of the node gc are obtained from the axioms in node (cr gc pn) by projecting out the unnecessary attributes, so that "s1(cr gc pn) $\wedge$ cr $>$ 7" turns into "s1(gc cr) $\wedge$ cr $>$ 7" (attribute cr is necessary to perform the selection), and similarly for the rest of the axioms. The second set are the interstitial axioms, which are computed at run-time in response to the demands of queries. For example, the attribute set (cn cr pn) does not appear in the basic axioms for large-seaport of Fig. 2. If a user query requests them, a new node (cn cr pn) is created and populated with interstitial axioms (shown marked with a hash, #, in Fig. 3). A detailed explanation of the algorithm for automatic compilation of the integration axioms and the maintenance of the axiom lattice lies outside the scope of this paper; see [5] for details.

Our integration axioms resemble the inverse rules in [17,18]. However, there are several important differences. First, we are using a description logic formalism as opposed to datalog. Second, our source descriptions are defined with equality, so the analogous case

in Duschka's complexity analysis is that of *closed-world* positive existential queries and views. The complexity of this case is NP-hard, as opposed to the open-world case that is polynomial. Third, our local-as-view source descriptions are more restricted than those in Duschka's analysis. In the SIMS model, one cannot describe a source as an arbitrary join of domain classes. However, a conjunctive domain class can be defined and a source can be linked to this new class. Fourth, our integration axioms involve only sources that correspond to classes in one hierarchy. We assume that each (source and domain) class has a key that identifies each object. This allows us to compile the global-as-view expressions from the local-as-view source descriptions without loss of information. Finally, our integration axioms are a form of partial evaluation of the integration model. Each integration axiom represents an alternative combination of sources that provides a given set of attributes for a *single* domain class. However, user queries may involve a large number of domain classes, so having these integration axioms facilitates, but does not solve the source selection problem for a given user query.

2.2. Overview of query planning in SIMS

Once the integration axioms have been compiled, SIMS is ready to accept user queries expressed over terms of the domain model. Query planning follows three main steps. First, the query is parsed, simplified, and rewritten so that each class mentioned in the query is the most specific according to the definitions in the domain model. For example, if a query against the model in Fig. 1 contains "seaport(cr pn) $\wedge$ cr $>$ 10", that portion of the query will be replaced by "large-seaport(cr pn) $\wedge$ cr $>$ 10", as any seaport with more than 7 cranes is a large-seaport. This ensures that the appropriate set of integration axioms are used. Second, the simplified query is sent to the initial plan generator which constructs an initial query plan. This initial plan is probably suboptimal, but it is generated very efficiently. Finally, the current plan is iteratively transformed using a set of rewriting rules in order to optimize the desired cost metric.

The query planner has a modular, declarative, and extensible architecture. The initial plan generator, the cost metric, the set of rewriting rules, and the strategy used during the rewriting search, are all modules that can be extended or replaced independently. Since our query planner is based on a domain-independent approach to planning, it is extensible in a principled way and very flexible. The specification of both the plan operators and the plan rewriting rules is declarative. In the following sections we describe a particular instantiation of the general Planning by Rewriting approach for query planning in the SIMS mediator.

3. Query planning as a classical planning problem

Classical AI Planning studies the problem 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, such as query planning, logistics, robot control and navigation, etc, 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 takes as *input* a declarative domain

specification (that is, the relevant properties of the state and the actions that change them), so that the planning engine remains the same from domain to domain. Some excellent surveys on classical domain-independent planning are [58,66,67].

Using a general domain-independent planning framework has two main advantages. First, the specification of the planning domain is declarative, so it is easy to understand, to maintain, and to extend. Second, domain-independent planning offers a uniform and principled solution to a host of problems that domain-specific planners tend to address in an ad-hoc manner. Most important for query planning are general search strategies and interleaving of planning and execution. In Section 5, we show how our modular planner can take advantage of a variety of general search strategies. In Section 8, we argue how specific techniques of the database literature can be seen as particular cases of general planning concepts such as conditional planning, replanning, and sensing actions. The declarative and principled design of domain-independent planners yields much more flexible and adaptable systems than domain-specific solutions. The major drawback of domain-independent planning is its efficiency. However, recent advances have produced planners of remarkable scalability [67]. A planner that is particularly well suited for query planning as we discussed in Section 1 is PbR.

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 [41]. The query language of this domain is union of conjunctive queries with arithmetic comparison predicates. In this section, we will describe the information goals (queries), the data processing operators, and some sample query plans.

Information goals. An information goal is represented by the predicate `(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 Fig. 4. This goal asks to send to the output device of the mediator all the names of seaports in Tunisia. The desired query is expressed in the Loom query syntax that the SIMS mediator accepts.[2] Note that the query itself is represented as a complex term within the `available` predicate.

```
(available output (retrieve (?name)
                            (:and (seaport ?port)
                                  (country-name ?port "Tunisia")
                                  (port-name ?port ?name))))
```

Fig. 4. Sample information goal.

[2] The SIMS language is a prefix logical notation. Variables start with a question mark (?). The query of Fig. 4 in SQL would be:

```
select port-name from seaport where country-name = 'Tunisia'.
```

Planning operators. The specification of the operators is shown in Fig. 5.[3] The query language addressed in this planning domain is union of conjunctive queries with interpreted predicates. The operators `join`, `select`, and `union` implement the corresponding relational algebra operations. The `retrieve` operator executes a query in a particular source and transmits the data to the mediator. The `assign` operator computes a new attribute by applying an arbitrary formula to previously obtained attributes. Finally, the `output` operator sends the result of the query to the output device of the SIMS mediator.[4]

All the operators in Fig. 5 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 (which in turn can satisfy the preconditions of other operators). In order to check for properties of the queries, the operators rely on user-defined interpreted predicates. An interpreted predicate is a predicate that is satisfied by evaluating a function on its arguments (possibly producing bindings for some of the arguments), as opposed to being satisfied by the effects of other operators (or the initial state) as a normal state predicate. To illustrate these ideas, we explain the behavior of the `join` and `retrieve` operators.

Consider the `join` operator of Fig. 5 and the sample instantiation of its variables shown in Fig. 6. A planning operator can be understood both in a forward and in a backward reasoning fashion. Reasoning forward (from preconditions to effects), the `join` operator joins two queries that are available locally in the mediator. These queries are represented intensionally by `?query-a` and `?query-b`, and extensionally by `?result-a` and `?result-b` (see Fig. 6 for the bindings of these variables in our example). The effect of this operator is to make available locally the joined query (`?query`), and, when executed, the results of the join (`!result`). Reasoning backwards (from effects to preconditions), the `join` operator states that in order to obtain (`?query`), it must be decomposable as a join of two subqueries (`?query-a` and `?query-b`) and that these subqueries must be in turn obtained. The interpreted predicate `join-query` is in charge of checking that `?query` is indeed conjunctive and of extracting the two subqueries. This is the way a regression planner, such as UCPOP [54] or Sage [40], would use the operator when generating the plan. Of course, plan execution always proceeds in the forward direction.

[3] Note that the predicate `available` actually used in the operators of Fig. 5 is `(available ?source ?host ?query !result)` as opposed to the simpler version of Fig. 4. 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)`.

[4] Given that typically a mediator does not have any control on the execution plans generated at a remote site, the data processing operators defined in Fig. 5, are executed locally at the mediator (`sims`) (although a complex query can be sent to a source for remote execution if capable). Nevertheless, this planning domain can be easily generalized to represent query plans that could execute in a distributed fashion. One would replace the constants `local` and `sims` in the occurrences of the `available` predicate in the operators with variables. During planning these variables would be instantiated to the appropriate locations in the network.

```
(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 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)
                       (union-query ?query ?subquery-a ?subquery-b))
   :effect (available local sims ?query !result))
```

Fig. 5. Operators for query planning.

As another example consider the `retrieve` operator in Fig. 5. This operator executes a query at an information source (`?source`) provided that the source is available at some host machine (`source-available`) and that the source is capable of processing the query (`source-acceptable-query`). The SIMS mediator keeps a model of the capabilities of the sources. The interpreted predicate `source-acceptable-query`

```
?query-a: (retrieve (?pn ?gc1) (:and (seaport ?a) (port-name ?a ?pn)
                                     (geoloc-code ?a ?gc1)))
?query-b: (retrieve (?gc2)
            (:and (location ?l) (geoloc-code ?l ?gc2)
                  (country-name ?l "Tunisia")))
?join-conds: (= ?gc1 ?gc2)
?query: (retrieve (?pn)
          (:and (seaport ?a) (port-name ?a ?pn) (geoloc-code ?a ?gc1)
                (location ?l) (geoloc-code ?l ?gc2)
                (country-name ?l "Tunisia") (= ?gc1 ?gc2)))
?result-a: (("Tunis" "XJCS") ("Long Beach" "NPTU")
            ("Bizerte" "BSRL")  ... )
?result-b: (("XJCS") ("BSRL") ... )
!result:   (("Tunis") ("Bizerte") ... )
```

Fig. 6. Behavior of the join operator.

checks the requirements of the query against the capabilities of the source. For example, the value of `?query` in Fig. 6 could not be accepted by a source that cannot perform a join operation even if both seaport and location were present at such source. Such a query would have to be decomposed first into the two simpler queries shown in Fig. 6 and the join processed locally. After executing the retrieve operator the results are available locally at the SIMS mediator. The remaining operators behave analogously.

Sample plans. Two plans generated using this planning domain specification appear in Figs. 7 and 8. Both plans evaluate the query in Fig. 4 but at a very different cost (cf. Section 5.1). These plans involve three source accesses (retrieves), two joins, one selection, and one output operation. For the particular example in Figs. 7 and 8 we have assumed that the remote sources can only answer very simple queries (i.e., they can only return all tuples and project columns, but cannot perform other operations, such as selections or joins).

The reason why the seemingly simple query in Fig. 4 expands into plans 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. In fact, these plans arise from the integration axiom in Fig. 9 that states that the required attributes for seaport can be obtained from the join of sources for seaport, location, and country. Note that the body of the axiom is expressed in domain terms in order to provide a level of abstraction over the possible sources for a query. For example, seaport(geoloc-code port-name) is obtained from the geoh@higgledy.isi.edu source in the plan of Fig. 7 and from the port@local source in the plan of Fig. 8. The plans of Figs. 7 and 8 are essentially alternative algebraic representations of the axiom in Fig. 9, with the addition of the selection on country-name required in the query. [5]

[5] A subtle point about the generation of the plans in our domain is that the interpreted predicates in the operators consult the integration axioms in the domain when processing a query. In the plans in Figs. 7 and 8, the `join-query` interpreted predicate, will successively partition the user query of Fig. 4 making explicit the two joins present in the axiom of Fig. 9.

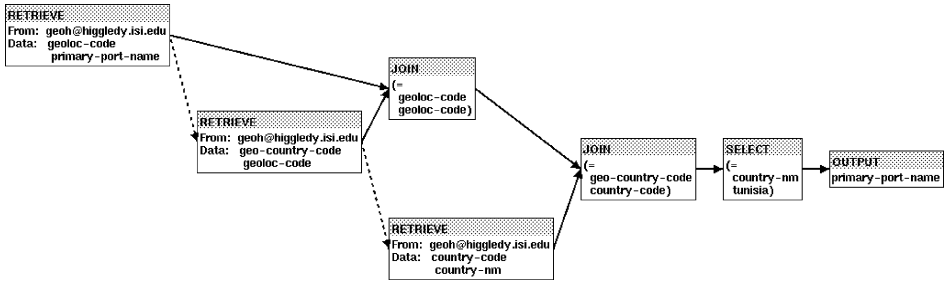


Fig. 7. Suboptimal initial query plan.

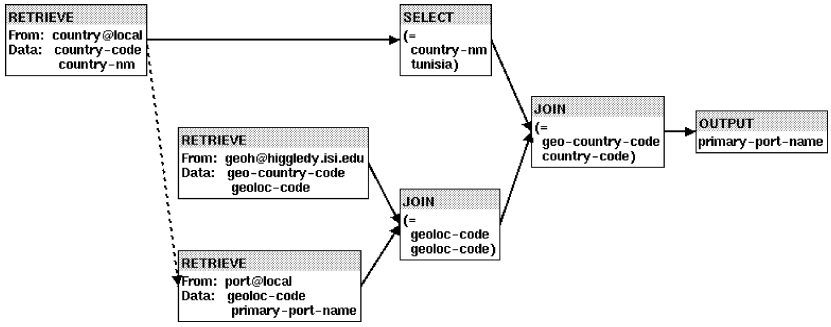


Fig. 8. Optimized query plan.

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

Fig. 9. Sample integration axiom.

4. Review of planning by rewriting

Planning by Rewriting [4] 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 declarative 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 [54]. The nodes are domain actions. The edges specify a temporal ordering relation among nodes, 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. Figs. 11, 13, 14, 15, and 16 in the next section are examples of plan rewriting rules in the query planning domain.

The following is a list of the main issues in Planning by Rewriting viewed as an instantiation of the local search idea to domain-independent planning. A detailed description of the general Planning by Rewriting paradigm is given in [3,4]. The next section discusses these issues in the context of query planning in mediators.

- *Efficient generation of an initial solution plan*. In many domains a possibly suboptimal initial plan can be constructed efficiently.
- *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 of possibly better quality.
- *Plan quality measure*. This is the plan cost function of the application domain that should be optimized during the planning process.
- *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.

5. 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. We will explain the approach following the main issues of local search as we did in the previous section. 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. The reason for this decomposition is two-fold: it simplifies the exposition, but, more importantly, it reflects the modular design of PbR. Cost estimation, initial plan generation, rewriting rules, and search strategy are all components of our system that can be changed independently. This modularity is another factor that contributes to the flexibility of our PbR-based query planner.

5.1. Query plan quality

There are many factors 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. In PbR the designer has the freedom to specify what are the important features in plan quality. In this respect, an advantage of PbR is that a complete plan is always available during the planning process, so that the designer can specify arbitrarily complex plan quality metrics. In generative planners only partial plans are available during optimization, so that simpler (additive) metrics must be used.

Estimating the execution time 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 gets compounded repeatedly through the intermediate results [35]. Moreover, obtaining the basic statistics for cost estimation from Web sources is a significant challenge. However, techniques that learn the statistics from the execution of queries such as [1,2,71] are promising. As there is a wealth of research on statistics gathering and cost estimation, in this paper we assume that the basic statistics are already available and our system uses the cost estimation detailed below. In any case, the fact that even with the most sophisticated estimation methods there always remains some degree of error supports using local search methods as in PbR. The computational complexity of finding the global optimum may not be warranted given that it is only an estimate of the real value. In practice, it is enough to find a good quality local optimum.

For the query planning domain in this paper 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. Our system estimates the execution cost of a plan 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 of 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 (as describe in [59]). 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 $R(x\ y) \wedge y = 7$ is 5 (100/20). If the query were $R(x\ y) \wedge x = 7$, since x is a key we would expect 1 tuple. The cost of a plan is the sum of the costs of each operation (estimated as the number of tuples it produces as described above), but in order to prefer more parallel plans, when a subplan has two parallel branches the cost of the subplan is the maximum of the cost of each of the parallel branches. Admittedly, this treatment of parallelism does not account for the local contention of resources, such as memory, that are going to be needed by both branches, but it is often appropriate in a Web environment where sites can be accessed in parallel and the transmission speed is more limiting than the size of the data.

As an example of plan quality, consider again the two alternative evaluation plans for the query in Fig. 4 shown in Figs. 7 and 8. Fig. 7 shows a randomly generated initial plan that has a much lower quality than the optimized plan in Fig. 8. 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.

5.2. 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. In this section, we first describe a completely random initial plan generator, then we discuss heuristics that improve its behavior to arrive at the generator that we used in the experiments in Section 6.

An initial query evaluation plan can be efficiently constructed as a random depth-first parse of the given query. For example, consider how the initial plan in Fig. 7 is obtained from the query in Fig. 4 by choosing randomly among the alternatives at each parsing and expansion point. The generation of this plan is best understood by reasoning backward from the `output` step (cf. Section 3). The first choice when analyzing the query in Fig. 4 is either to extract the selection country-name = "Tunisia" from the query (equivalently, introduce a `select` operator in the query plan) or expand the query using the integration axiom of Fig. 9. In Fig. 7 the `select` operator is chosen. Because the remaining query contains only one domain class (seaport) but no source can provide all the required attributes, the next decision is fixed: the seaport class must be expanded using an integration axiom. Assume the axiom in Fig. 9 is chosen. This results in a new query (the body of the axiom) that is the conjunction of three classes. Thus, next, two `join` operators must be inserted in the query plan. Again, the order in which to perform the two joins is chosen randomly. Finally, the three remaining queries, which contain one class each, are implemented by corresponding `retrieve` operators since there are sources in which they can be executed. If there are several alternative sources for the same source class, this constitutes another choice point. For the initial plan of Fig. 7 the same source geoh@higgledy.isi.edu happens to be selected for all three `retrieve`s. This kind of completely random initial plans are correct and efficient to produce, but they may be of very low quality.

The designer 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 initial plans than a random generator. For example, as 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. In general, starting from a higher quality initial plan does not guarantee that we find the optimal any sooner. In fact, sometimes good local minima can be reached more easily from a random distribution of initial plans. 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 the experiments in Section 6 randomly chooses the sources, the integration axioms, and the order of join, union, and the assignment operators, but it places selections as early as possible as this is an excellent heuristic for this domain.

5.3. 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. The rule set is extensible and specified declaratively. We defined a set of rewriting rules that address both source selection and cost-based optimization, facilitating the efficient search of the combined optimization space. The rules are derived from properties of the distributed environment, the relational algebra, and the integration axioms in the application domain. We describe each of these classes in the following sections.

5.3.1. 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 Fig. 10. 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 arise from the fact that it is often 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 mediator. Thus, `Remote-Join-Eval` pushes a query that contains a join for execution at a remote source, `Remote-Selection-Eval` pushes a selection, `Remote-Assignment-Eval` pushes a query that contain an assignment formula, and `Remote-Union-Eval` pushes unions. Note

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

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

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

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

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

Fig. 10. Transformations (distributed environment).

```
(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)))))
```

Fig. 11. Remote-join-eval rewriting rule.

the need for checking the capabilities of the information sources in the transformations in Fig. 10, as we do not assume that sources are full databases. The sources may have no query processing capabilities (for example, wrapped Web pages) or support very limited types of queries (for example, Web forms).

Fig. 11 shows the `Remote-Join-Eval` rule in the declarative input syntax accepted by the PbR planner. PbR matches the subgraph specified in the antecedent, the `:if` field, against the current plan, deletes the subset of the matched subplan identified in the `:replace` field, and replaces it by the subplan in the `:with` field. A full specification of this syntax appears in [3]. The `Remote-Join-Eval` rule states that if in a plan there exist two retrieve operators (nodes `?n1` and `?n2`) on the same remote source which are consequently joined (node `?n3`), and the remote source is capable of performing joins, the system can rewrite the plan into one that contains a single retrieve operation (node `?n4`) that pushes the join operation to the remote database (recall from the definition of the `join` operator in Section 3 that `?query` is the join of `?query1` and `?query2`). A graphical example of the application of this rule during query planning is shown in Fig. 17. Similarly, the three remaining rules cover the other algebraic operators that the SIMS language supports. These rewriting rules apply iteratively to ensure that as much of a complex query is evaluated remotely whenever it is cost-efficient to do so.

Although the transformations in Fig. 10 are shown applying in only one direction, the other direction is also valid. The designer has the choice of specifying rules that correspond to both directions or only one. Implementing both will cover the entire solution space. However, some direction may be much preferred. For example, the direction that pushes operations to the remote sources is generally much more useful, thus this is the direction that we included in the system. This results in a loss of accessibility to some optima, but the savings in planning time, by reducing the number of rules and the plans generated, outweighs reaching some rarely occurring optima. This is another version of the utility problem [51].

5.3.2. 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 Fig. 12. The `Join-Swap` rule combines the associate and commutative properties of the relational join. The `Selection-Join-Swap` and `Selection-Union-Swap` reflect the distributive property of selection with respect to join and union. The `Assignment-Swap` rule is the distributive property of the assignment operator

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: $Q1 \bowtie (Q2 \cup Q3) \Leftrightarrow (Q1 \bowtie Q2) \cup (Q1 \bowtie Q3)$

Fig. 12. Transformations (relational algebra).

with respect to join. And finally the `Join-Union-Distributive` is the distributive property of join and union.

Although the transformations in Fig. 12 are shown in a logical form, the reader should also view them as transformations on query plan trees (graphs), as this is exactly how PbR uses them. Consider the `Join-Swap` rule shown in its PbR syntax in Fig. 13(a). This rule specifies how two consecutive joins operators can be reordered and allows the planner to explore the space of join trees. An example of the application of this rule appears in Fig. 13(b). The left plan in Fig. 13(b) shows how query `ABC`, which is a conjunction of three subqueries `A`, `B`, and `C`, is implemented using two join operators. Applying the `Join-Swap` rule to this plan generates two rewritings: one plan with the positions of `A` and `C` exchanged, and the other with the positions of `B` and `C` exchanged. Note how the variables in the rule specification in Fig. 13(a) are bound to the operators and queries in the plans detailed in Fig. 13(b). Another example of the application of this rule appears in Fig. 17.

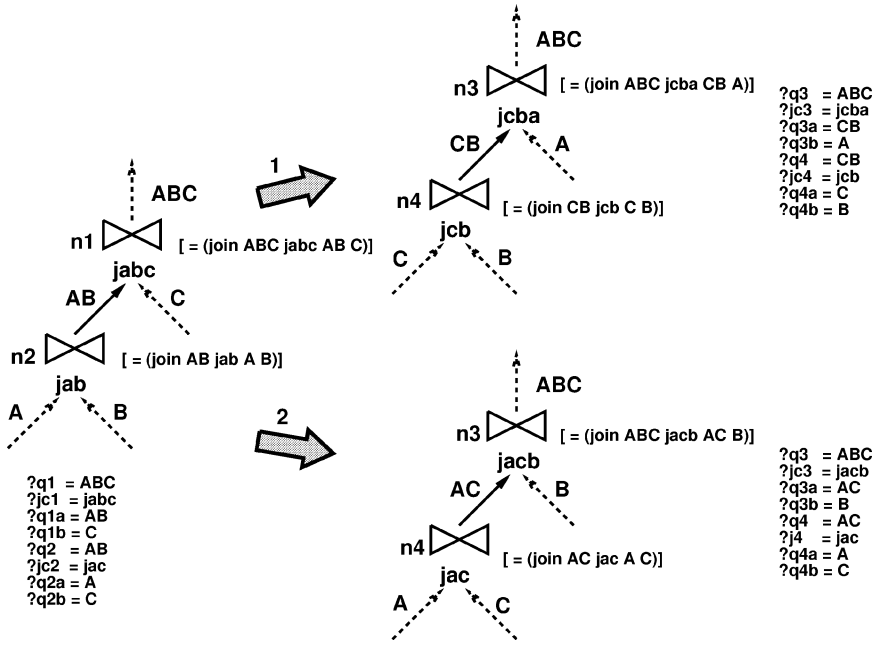
As described in Section 3 queries are expressed as complex terms. The PbR rules use the interpreted predicates in the `:constraints` field to manipulate such query expressions (in a manner analogous to the interpreted predicates in the operators). For example, the `join-swappable` predicate in Fig. 13(a) checks if the two join operators have queries that can be exchanged. It 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). Note that the `join` operators in the rule antecedent do not share any variables among themselves nor with the operators in the consequent. This enables the rule to match both left trees and right trees.

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 Fig. 13, we could have defined a join associativity rule more directly as in Fig. 14(a). This is a simpler rule but it also makes more commitments than `join-swap`.

The `join-associativity` rule enforces the matching join structure to be a left tree and produces a right join tree as shown in Fig. 14(b). Therefore, we would also need another rule for join commutatitivy as the one shown in Fig. 15, 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.

```
(define-rule :name join-swap
  :if (:operators ((?n1 (join ?q1 ?jc1 ?q1a ?q1b))
                   (?n2 (join ?q2 ?jc2 ?q2a ?q2b)))
       :links (?n2 ?n1)
       :constraints
        (join-swappable ?q1 ?jc1 ?q1a ?q1b ?q2 ?jc2 ?q2a ?q2b   ;;in
                        ?q3 ?jc3 ?q3a ?q3b ?q4 ?jc4 ?q4a ?q4b)) ;;out
  :replace (:operators (?n1 ?n2))
  :with (:operators ((?n3 (join ?q3 ?jc3 ?q3a ?q3b))
                     (?n4 (join ?q4 ?jc4 ?q4a ?q4b)))
         :links (?n4 ?n3)))
```

(a)



(b)

Fig. 13. Join-swap rewriting rule. (a) Rule in PbR syntax. (b) Example of rewriting.

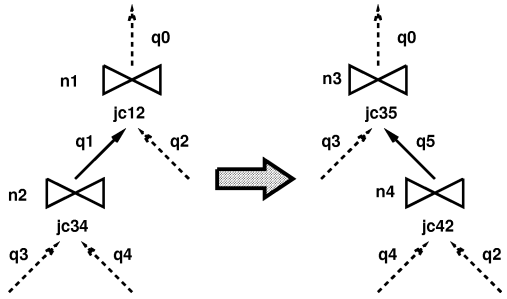
A greater number of rules decreases the performance of matching. More importantly, rules that are discriminative in the cost surface are more useful. In our system we used `join-swap` because it conveniently combines all four cases into one rule and produces rewritings of very different cost, which helps to move through the cost space

5.3.3. Rewriting rules from the integration axioms

The third set of rewriting rules arises from the heterogeneous nature of the environment. As we explained in Section 2, our mediator reconciles the semantic differences among the

```
(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 ?q1 ?q2 ;; in
                                      ?jc34 ?q3 ?q4     ;; in
                                      ?jc42 ?q5 ?jc35)) ;; out
  :replace (:operators (?n1 ?n2))
  :with (:operators ((?n3 (join ?jc35 ?q0 ?q3 ?q5))
                     (?n4 (join ?jc42 ?q5 ?q4 ?q2)))))
```

(a)



(b)

Fig. 14. Join-associativity rewriting rule. (a) Rule in PbR syntax. (b) Schematic.

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

Fig. 15. Join-commutativity rewriting rule.

sources using a set of integration axioms. 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. Only rewriting rules from axioms that are guaranteed to be relevant to the given query are used in the rewriting process. An axiom is relevant if it provides the attributes that a class in the query requires. This determination is efficient because the integration axioms are precompiled and cached in the lattice structure introduced in Section 2. For example, assume that a user query requests large-seaport(gc pn cr) against the model of Fig. 2 and that the relevant axioms are:

(a) large-seaport(cr gc pn) $\Leftrightarrow$ seaport(cr gc pn) $\wedge$ large-seaport(pn)

(b) large-seaport(cr gc pn) $\Leftrightarrow$ american-large-seaport(cr gc pn) $\vee$
european-large-seaport(cr gc pn)

Axioms (a) and (b) correspond to axioms *4.3* and *5.1* in Fig. 2. The only difference is that the body of the axiom is written with domain terms. In the integration model in Fig. 1 sources s1, s2, s4, and s5 map to seaport, large-seaport, american-large-seaport, and european-large-seaport, respectively. The reason for using the axioms at the domain level is that it provides a layer of abstraction over the sources chosen to implement a particular domain class and facilitates the rewriting.

The rewriting rules corresponding to these integration axioms are shown in Fig. 16. Rule (a) corresponds to the axiom (a). This rule states that if in a plan there exist a set of operators (`?nodes`) that obtain attributes cr, gc, and pn of the large-seaport class, the planner could alternatively obtain this information using the axiom (a) 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 operators in the rule consequent. Similarly, rule (b) specifies the alternative of using axiom (b) to obtain large-seaport(cr gc pn). Note that the consequents of these rules are just the algebraic versions of the bodies of the axioms using the operators in Fig. 5.

This type of rewriting rule is used to exchange alternative integration axioms in a plan. There are two subtle points to this rule specification. First, for n alternative integration axioms for the same domain class and attributes, our system only writes n rules as opposed to the $O(n^2)$ direct exchanges. A direct exchange rule would specify one axiom (in its algebraic/graph form) in the antecedent and an alternative axiom (in its algebraic/graph form) in the consequent. Therefore n alternative axioms would generate $n^2 - n$ rules (all pairs of axioms, except those rules that would have the same axiom in antecedent and consequent). Instead, our system only generates one rule per alternative axiom, because it uses the axiom head as an abstraction of the set of alternative axioms. For example, note how in the antecedents of the rules in Fig. 16 only the axiom head, large-seaport(gc pn cr), is mentioned. Each rule only describes an axiom (the one that is introduced by the consequent). Second, the interpreted predicate `identify-axiom-steps` finds all the operators that implement the body of an axiom in a very simple way, it just looks for an operator in the query plan that produces a query that matches the axiom head. The subtree rooted at such operator must be an implementation of the axiom body. This fact follows from the encoding of the query planning domain with the operators in Fig. 5.

5.4. Searching the space of query plans

The space of rewritings for query planning in mediators is too large for complete search methods to be scalable. In addition, the fact that in query planning, as in many other domains, 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. In order to explore the space of query plans our planner currently uses variations of gradient descent. *First improvement* gradient descent generates the rewritings incrementally and selects the first plan of better cost than the current one. *Best improvement* gradient descent generates all the plans produced by

```
(define-rule :name (<=> (large-seaport cr gc pn)
                        (:and (seaport cr gc pn) (large-seaport pn)))
  :if (:constraints (identify-axiom-steps (large-seaport cr gc pn)))
  :replace (:operators ?nodes)
  :with (:operators ((?n1 (retrieve s1 (seaport cr gc pn)))
                     (?n2 (retrieve s2 (large-seaport pn)))
                     (?n3 (join (large-seaport cr gc pn)
                                ((= pn pn.2))
                                (seaport cr gc pn)
                                (large-seaport pn.2))))))

                          (a)


(define-rule :name (<=> (large-seaport cr gc pn)
                        (:or (american-large-seaport cr gc pn)
                             (european-large-seaport cr gc pn)))
  :if (:constraints (identify-axiom-steps (large-seaport cr gc pn)))
  :replace (:operators ?nodes)
  :with (:operators
          ((?n1 (retrieve s4 (american-large-seaport cr gc pn)))
           (?n2 (retrieve s5 (european-large-seaport cr gc pn)))
           (?n3 (union (large-seaport cr gc pn)
                       (american-large-seaport cr gc pn)
                       (european-large-seaport cr gc pn))))))

                          (b)
```

Fig. 16. Rewriting rules for integration axioms.

(one application of) the rewriting rules and selects the best rewriting among them. In order to escape low-quality local minima, our system uses random restart and a fixed-length random walk to traverse plateaus.

Fig. 17 shows an example of the local search through the space of query plan rewritings. We use plans derived from the axiom in Fig. 9. Assume that the data for the classes seaport and location resides at the source geo-db, and the data for class country resides at the source assets-db. The figure shows alternative query evaluation plans for a conjunctive query that asks for the port and country names of seaports. The leftmost plan is the initial plan. This plan first retrieves the location class at geo-db and the country class at assets-db, and joins them on the attribute country-code. Then, the plan retrieves the seaport class from geo-db source and joins it on geoloc-code 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 an expensive operation, so the system, following a gradient descent search strategy, prefers the other plan. Finally, the planner applies the `remote-join-eval` rule that generates a new rewritten plan that evaluates the join between the location and seaport classes remotely at the geo-db database, which is a more cost-effective plan.

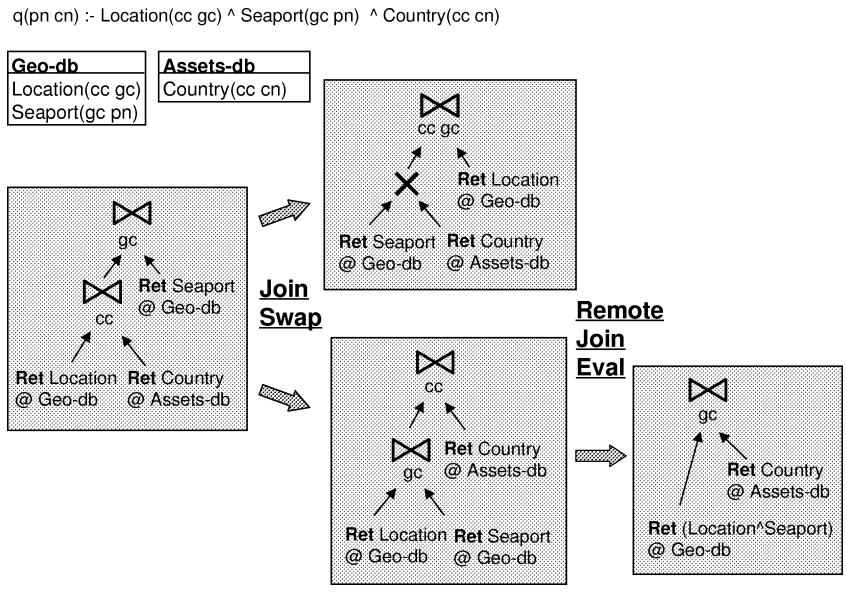


Fig. 17. Search process.

6. Experimental results

The Planning by Rewriting approach has been fully implemented and applied in several application domains [3]. We have designed a series of controlled experiments that test our PbR-based query planner along several factors that contribute to the complexity of query planning in mediators and compare the planning efficiency and plan quality of PbR with other approaches to query planning. We present results for four query planners:

Sage. This is the original query planner [40,41] 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 [39] that generates optimal left-tree query plans.

DP. This is our implementation of a dynamic-programming bottom-up enumeration of query plans (in the style of [53]) adapted for query planning in mediators. The algorithm proceeds in stages. First, it generates all alternative translations of a domain query to source queries using the integration axioms. Then, the optimal plan for each source query is obtained using a dynamic programming bottom-up enumeration. Finally, the best plan among all the optimized source plans is chosen. Since in the Web domain we can execute subqueries in parallel and the cost function reflects that preference, our algorithm considers bushy join orders. However, to improve its planning time, DP applies the heuristic of avoiding cross-products during the enumeration of join orders. Thus, in some rare cases this version of DP may not produce the optimal plan.

Initial. This is the initial plan generator for PbR. It generates plans according to a random depth-first search parse of the query as described in Section 5.2. The only non-random choice is that it places selections as soon as they can be executed. It is the fastest planner but may produce very low quality plans.

PbR. This is an instantiation of the PbR framework for query planning in mediators as described in this paper. We explored several combinations of the rewriting rules introduced in Section 5.3 and two gradient descent search strategies: first-improvement and steepest descent.

We performed two scalability experiments. In the first experiment we compare the behavior of Initial, PbR, DP, and Sage in a distributed query planning domain as the size of the queries increases. In the second experiment we compare the scalability of PbR and DP in the presence of complex integration axioms, showing the effects of increasing the size of the queries, the size of the integration axioms, and the number of alternative axioms.

6.1. Scaling the query size

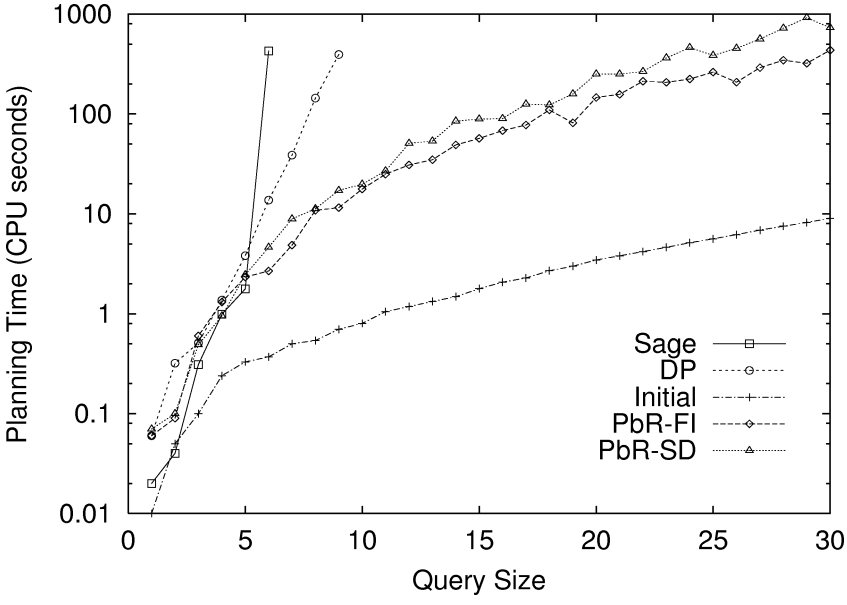
For the first experiment we generated a synthetic domain for the SIMS mediator and defined a set of conjunctive chain queries involving from 1 to 30 domain classes. The queries have one selection on an attribute of each class. The structure of the queries is shown in Fig. 18. There is only one source 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`, etc.) to the sources. For PbR we defined the `Join-Swap` and the `Remote-Join-Eval` rules, defined in Figs. 13 and 11, that are sufficient to optimize the set of test queries. Rules involving selections are not used because the initial planner already pushes the selections to the sources, which is a very good heuristic. During the best-first search of Sage, plans with selections placed closer to the source are also considered first. To improve planning time, DP avoids cross-products during the dynamic-programming enumeration of join orders. We tested two gradient-descent search strategies for PbR: first improvement with four random restarts (PbR-FI), and steepest descent with three random restarts (PbR-SD).

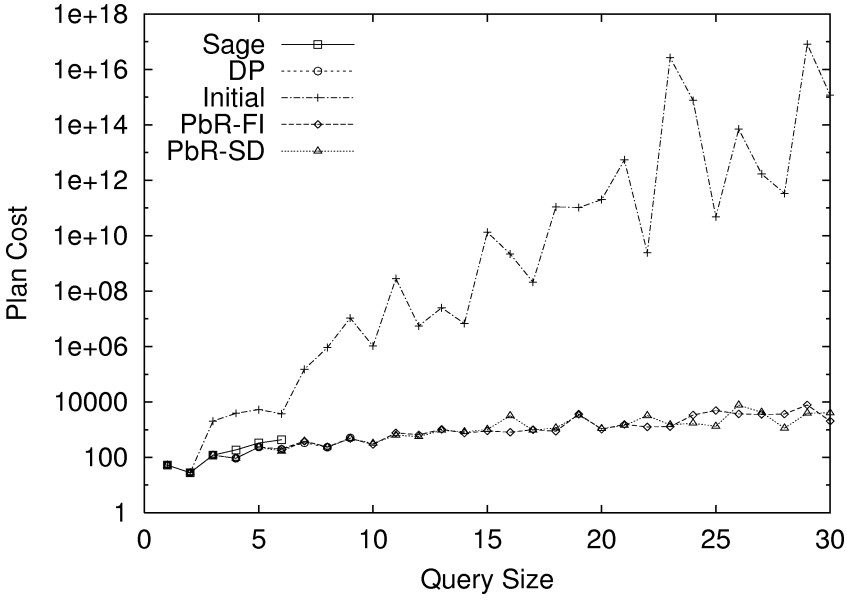
The results of this experiment are shown in Fig. 19. Fig. 19(a) shows the planning time, in a logarithmic scale, for Sage, DP, Initial, PbR-FI, and PbR-SD as the query size grows. The times for PbR include both the generation of all the random initial plans and their rewriting. The times for Initial are the average of the initial plan construction across all the

```
q(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 ...
```

Fig. 18. Query pattern.

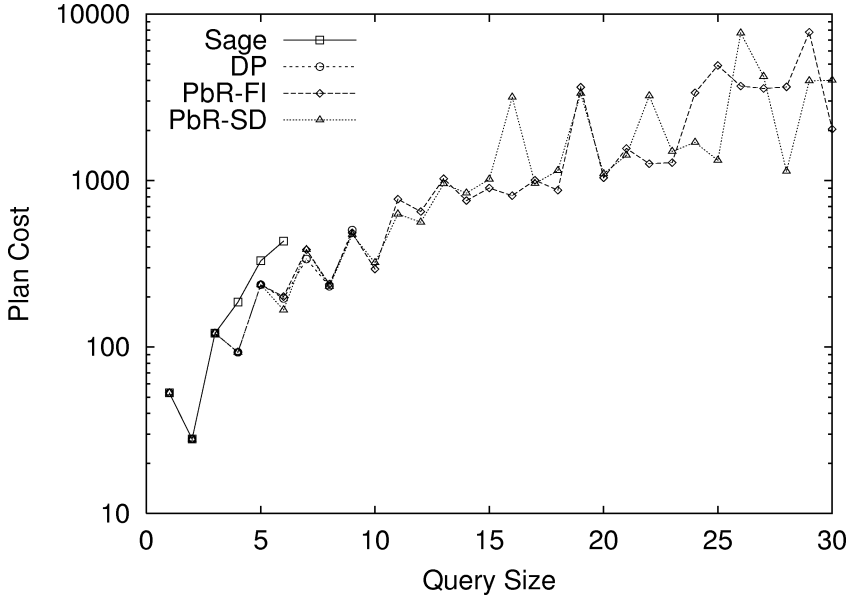


(a)

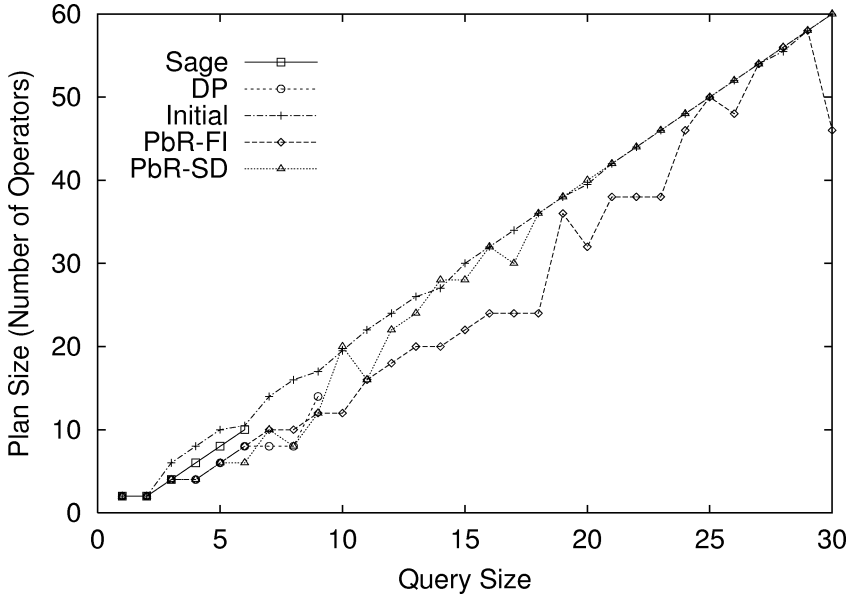


(b)

Fig. 19. Scaling the query size. (a) Planning time. (b) Plan cost.



(c)



(d)

Fig. 19. Scaling the query size (continued). (c) Plan cost (detail). (d) Plan size.

restarts of each query. Sage is able to solve queries involving up to 6 classes, but larger queries cannot be solved within its search limit of 200,000 partial-plan nodes. DP scales better than Sage, but cannot solve queries of more than 9 classes in the 1000 second time limit. Both configurations of PbR scale better than Sage and DP. The first-improvement search strategy of PbR-FI clearly dominates the steepest descent of PbR-SD.

Fig. 19(b) shows the cost of the query plans for the five planners. Fig. 19(c) compares in detail the cost of the plans produced by Sage, DP, and the two configurations of PbR. [6] A logarithmic scale is used because of the increasingly larger absolute values of the plan costs for our conjunctive chain queries. The plan cost is an estimate of the query execution time as described in Section 5.1. However, the optimizing behavior of PbR is largely independent of the particular cost metric used, so we would expect similar results for other cost metrics. The cost for Initial is the average of all the initial plans. Fig. 19(b) shows how PbR rewrites the very poor quality plans generated by Initial into high-quality plans. PbR produces plans of quality comparable to DP for its tractable range and beyond that range PbR scales gracefully. In fact, as it can be seen in Fig. 19(c), PbR sometimes produces a better plan than DP because PbR considers plans with cross products and our version of DP, to improve its planning time, does not. Both PbR and DP produce better plans than Sage (in the range tractable for Sage) for this experiment. This happens because they are searching the larger space of bushy query trees and can take greater advantage of parallel execution plans. The two configurations of PbR produce plans of similar cost, though PbR-FI needed less planning time than PbR-SD as shown in Fig. 19(a).

Finally, Fig. 19(d) shows the number of operators in each plan. The plans generated by PbR and DP use fewer operators 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 in PbR collapses three operators (two retrieves and join) into one (retrieve) when it is cost efficient to do so.

In summary, this experiment shows that PbR scales better than both traditional AI planning approaches such as Sage and database query optimization techniques such as Dynamic Programming, while producing high quality plans, for a distributed query planning domain. In the next section we show that PbR also performs well when the complexity of the optimization problem does not only come from query size, but also from increasing the number and the size of the integration axioms.

6.2. Scaling the integration axioms

This collection of experiments analyzes the effect of complex integration axioms in the efficiency and quality of planning in mediators. We compare the performance of PbR and the dynamic-programming planner DP. We also compared against Sage, but DP outperforms Sage as in the previous experiment, so we only report results for DP.

[6] The reason for the jagged shape of Fig. 19(c) is twofold. Recall that in this experiment each information source contains two source classes, thus in the even query sizes there are more opportunities for remote execution. This is particularly noticeable for small query sizes (up to 12 relations) where the cost decreases exactly on the even query sizes. For larger queries, such effect diminishes and the random nature of PbR dominates which local optima are obtained.

We designed a set of parameterized integration models in which we scale the size of the integration axioms as well as the number of alternative integration axioms for each domain class.

For this experiment we use a set of conjunctive integration axioms following the pattern shown in Fig. 20. In this figure domain classes have a `dc` suffix (e.g., `Adc`, `Bdc`, ...), source classes contains `sc` (e.g., `Asc01`, `Bsc11`, ...), and attributes containing `k` are keys. This axiom structure allows us to vary two of the main dimensions that affect the complexity of query planning in mediators. The first dimension is the axiom size, measured as the number of conjuncts in the axiom body. In this axiom structure the length of the axioms relevant to a user query is controlled, indirectly, by the attributes that the query requests. For example, consider the first axiom for domain class `Adc` in Fig. 20 that is a conjunction of source classes (`Asc00`, `Asc01`, `Asc02`, ...) joining in the key attribute `ak0`. Each source class uniquely produces an attribute (e.g., `Asc00` is the only source class to produce `a0`, only `Asc01` produces `a1`, etc.). Thus, we control the relevant source classes from an axiom, the effective axiom size, by just requesting the corresponding attributes in a query.

The second dimension is the number of alternative axioms for each domain class. For example, the second axiom for `Adc` in Fig. 20 is structurally identical to the first except that the axiom body joins on the key `ak1` as opposed to `ak0`. Note that except for the keys (`ak0`, `ak1`, ...) the returned attributes of both axioms are the same. Therefore for any query that requests any combination of attributes not involving the keys `aki` all the axioms for a given class are relevant. In other words, each of the axioms for a domain class is a valid alternative to answer such query.

In order to measure the optimization abilities of the planners, the sources involved in each integration axiom have different access costs. In particular, the cost of accessing the sources for each axiom is increased along each source in the axiom body, on each alternative axioms for a given domain class, and across domain classes. For example, in

```
Adc(ak0 rc a0 a1 a2 ...) ⇔ Asc00(ak0 a0 rc) ∧ Asc01(ak0 a1) ∧
                           Asc02(ak0 a2) ∧ ...
Adc(ak1 rc a0 a1 a2 ...) ⇔ Asc10(ak1 a0 rc) ∧ Asc11(ak1 a1) ∧
                           Asc12(ak1 a2) ∧ ...
⋮
Bdc(bk0 rc b0 b1 b2 ...) ⇔ Bsc00(bk0 b0 rc) ∧ Bsc01(bk0 b1) ∧
                           Bsc02(bk0 b2) ∧ ...
Bdc(bk1 rc b0 b1 b2 ...) ⇔ Bsc10(bk1 b0 rc) ∧ Bsc11(bk1 b1) ∧
                           Bsc12(bk1 b2) ∧ ...
⋮
```

Fig. 20. Parameterized integration axioms.

```
q(rc a0 a1 a2 b0 b1 b2) :- Adc(rc a0 a1 a2) ∧ Bdc(rc b0 b1 b2)
```

(a) Domain Query

```
q(rc a0 a1 a2 b0 b1 b2) :-
      Asc00(ak0 a0 rc) ∧ Asc01(ak0 a1) ∧ Asc02(ak0 a2) ∧
      Bsc10(bk1 b0 rc) ∧ Bsc11(bk1 b1) ∧ Bsc12(bk1 b2)
```

(b) Source Query Expansion

Fig. 21. Sample domain query and its source expansion.

the first axiom in Fig. 20 (joining on `k0`) accessing `Asc00` is cheaper than accessing `Asc01`, and so on. Similarly, the sources for second axiom increase with respect to the first. Analogously, the sources for axioms of `Bdc` have a higher cost than for `Adc`.

In this experiment we tested conjunctive star queries (which are hard to optimize [49]). The queries in the test set involve an increasing number of domain classes, which are joined on a single non-key attribute (`rc`). For example, consider the test query in Fig. 21(a) that is a join of two domain classes: `Adc` and `Bdc`. Since the attributes required from domain class `Adc` are `a0`, `a1`, and `a2`, the relevant integration axioms from Fig. 20 must have 3 conjuncts. Fig. 21(a) shows the corresponding source-level expansion using the first axiom for `Adc` and the second for `Bdc` from Fig. 20.

We scaled along the three dimensions that contribute to the complexity of query planning in mediators, namely, the query size, the size of the integration axioms, and the number of alternative axioms. We tested conjunctive queries involving from 1 to 5 domain classes, we scaled the length of the body of each individual axiom from 1 to 5 source classes, and we scaled the number of alternative axioms *per* domain class from 1 to 5. For the largest problem there is a total of 125 sources (5 domain classes in the query times 5 alternative axioms per class times 5 sources per axiom). The effective query size from the perspective of the optimizer 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. The number of alternative axioms determines the number of alternative source queries to which a given domain query can be translated. For example, for a query involving 5 domain classes and 5 alternative axioms for each class, there are 3125 (5^5) alternative source queries. We generated a data cube of 125 points along these three dimensions on which we measured planning time and plan cost for DP and PbR.

PbR used the rules derived from the relevant integration axioms for each query (cf. Section 5.3.3) in addition to the `Join-Swap` rule defined in Fig. 13. The number of relevant integration axioms (and thus rules) is the number of domain classes in the query times the number of alternative integration axioms. For example, for a query with 5 domain classes and 5 alternative integration axioms per class the system generates 25 rules. Thus, the system would apply 26 rules (25 from the integration axioms plus `Join-Swap`) at each rewriting cycle. We have assumed that the sources do not have remote execution capabilities as it is often the case in the Web. PbR used a first improvement search strategy

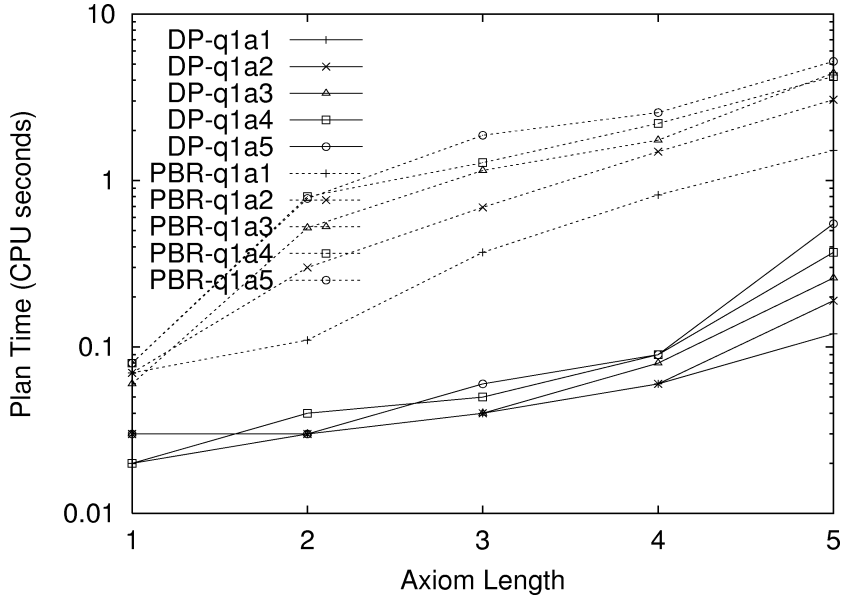
with four restarts. PbR considers all bushy join orders, but DP to improve its running time, avoids cross products.

The results of this experiment appear in Figs. 22 and 23. In these graphs, PbR data is shown with dashed lines and DP data with solid lines. In the graphs DP and PbR use the same dot marker to denote the same experiment parameters. The experiments had a time limit of 900 CPU seconds. PbR solved all problems under the time limit, but DP could not solve many queries as we scaled the complexity of the query planning problem.

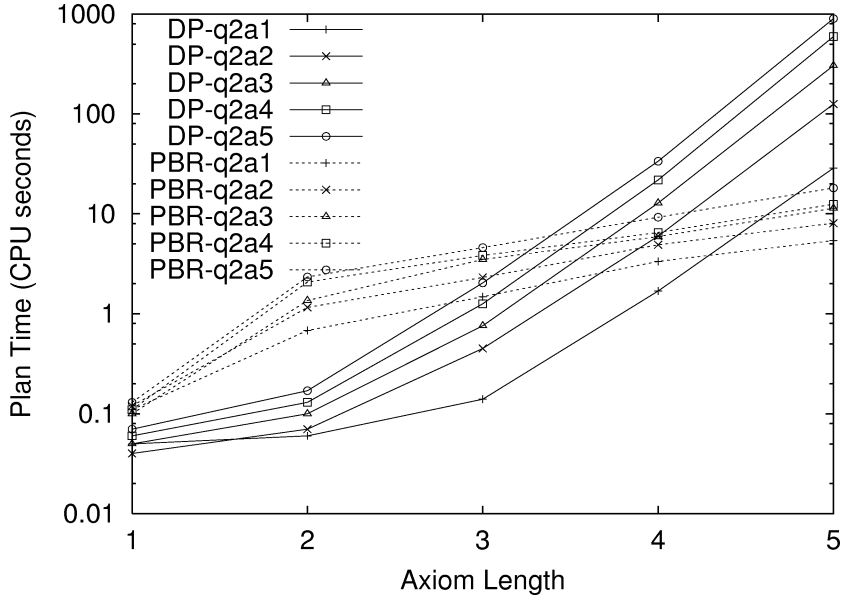
Fig. 22 shows the evolution of the planning time for PbR and DP along three dimensions that affect the complexity of query planning. The first dimension is query size. Each of the five graphs (Figs. 22(a)–22(e)) represents the results for a fixed domain query size from 1 to 5 domain classes, that correspond to source query plans involving from 5 to 25 sources classes, respectively (recall that a domain query expands into a much larger source query). 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 in the y-axis (using a logarithmic scale). The third dimension is the number of alternative axioms. Each graph shows 5 lines per planner parameterized by the number of alternative axioms. For example, the line labeled DP-q3a4 (in Fig. 22(c)) shows the result for the DP planner on a query involving 3 domain classes with 4 alternative axioms per class. The corresponding result for PbR is line PBR-q3a4 (note how both use the same square dot marker).

PbR scales much better than DP with query and axiom size. For example, consider Fig. 22(c) which shows the results for queries with 3 domain classes (which generate plans involving up to 15 source classes). The graph shows that PbR is significantly faster than DP as we increase the size of the integration axioms. PbR solves all queries, but DP cannot solve the queries over an axiom length of 4 source classes. Even at axiom length of 4, DP cannot explore all alternative source queries (it times out) when there are more than 2 alternative axioms per domain class. Similarly, increasing the number of alternative axioms affects DP quite strongly as shown by the large difference in planning time among the set of DP lines, while PbR shows much better scaling. For example, consider the points at axiom length 3 in Fig. 22(c). From a single integration axiom (line DP-q3a1) to five alternative axioms (line DPq3a5) there is a difference of about two orders of magnitude in planning time. The corresponding case for PbR shows a significantly smaller variance. Consider now Fig. 22 as a whole. As we increase the domain query size from 1 to 5, graphs (a) to (e), DP is able to solve fewer and fewer queries, disappearing from the graphs until it only solves the queries involving axioms with two conjuncts at domain query size 5. Meanwhile, PbR is able to solve all queries and its planning time scales well with the increasing query size, axiom size, and number of alternative axioms.

Fig. 23 shows the evolution of plan quality along the same three 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 high-quality plans. The quality of PbR is comparable to that of DP in the range of problems solvable by DP and beyond it scales gracefully. For some points DP does not produce the optimum. For example, lines DP-q3a2, DP-q3a3, DP-q3a4, and DP-q3a5 for an axiom length of 4 sources in Fig. 23(c). The reason is that our version DP for mediators computes first all query translations from domain to source terms and then optimizes each

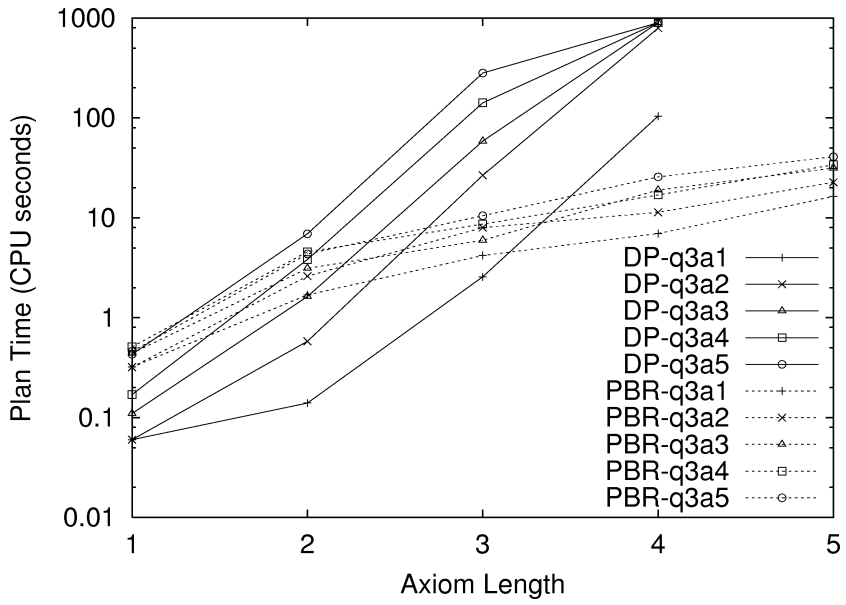


(a)

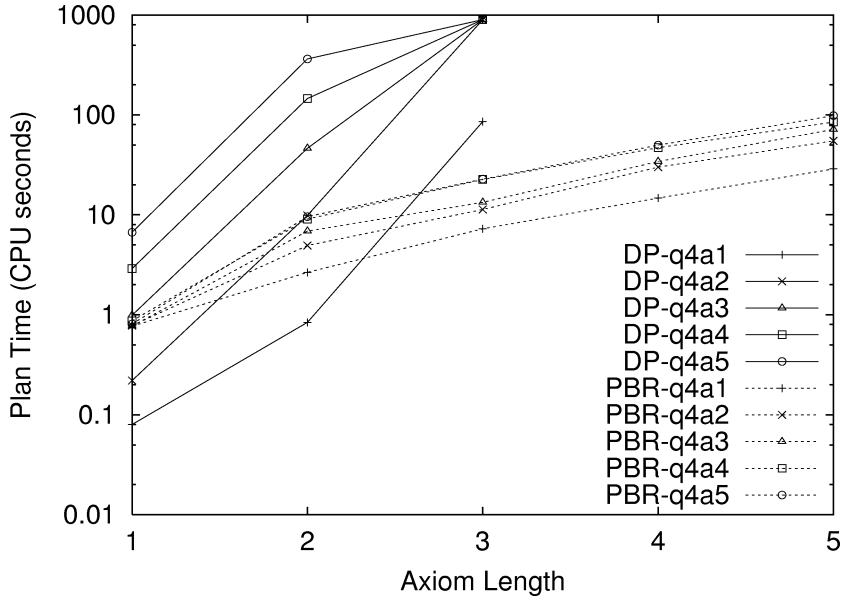


(b)

Fig. 22. Scaling the integration axioms: Planning time. (a) Domain query size = 1 classes. Source query size = up to 5 classes. (b) Domain query size = 2 classes. Source query size = up to 10 classes.

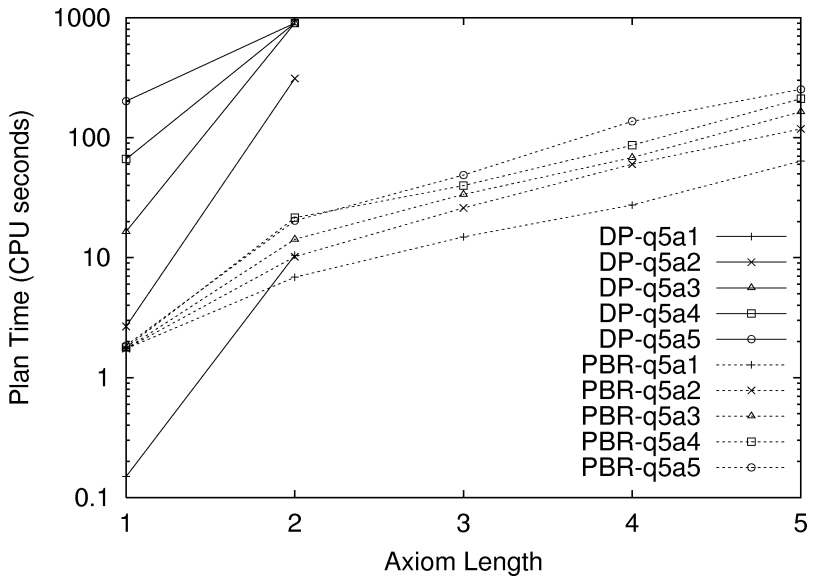


(c)



(d)

Fig. 22. Scaling the integration axioms: Planning time (continued). (c) Domain query size = 3 classes. Source query size = up to 15 classes. (d) Domain query size = 4 classes. Source query size = up to 20 classes.



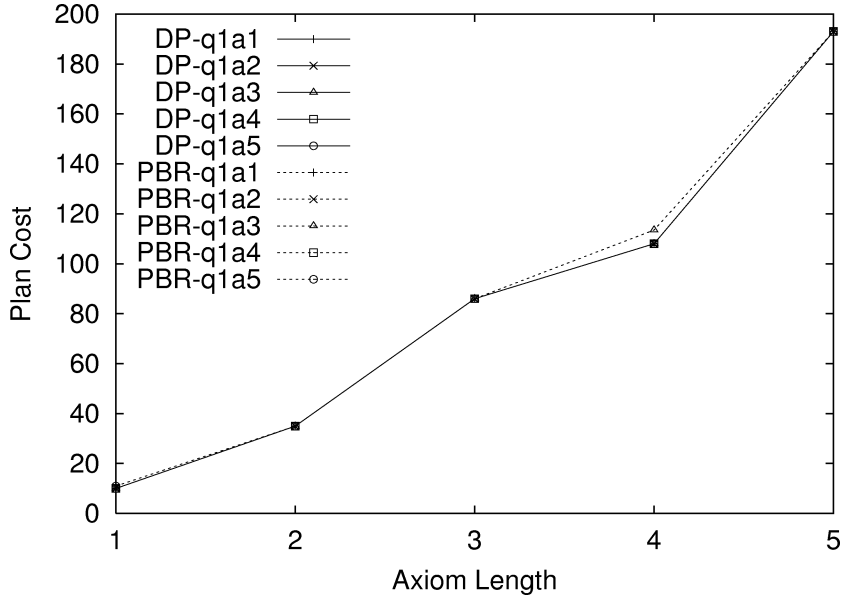
(e)

Fig. 22. Scaling the integration axioms: Planning time (continued). (e) Domain query size = 5 classes. Source query size = up to 25 classes.

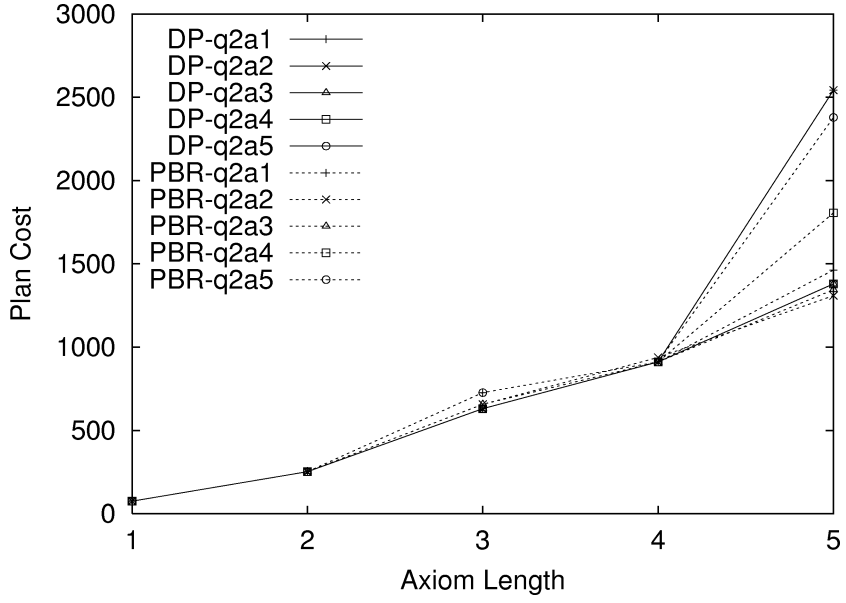
source query sequentially. Upon reaching the time limit, DP outputs the best complete plan found so far, but the optimum may lie in an alternative source query not yet optimized. In the case of Fig. 23(c), DP timed out before finding the source query that would lead to the optimal plan.

Finally, Fig. 24 compares the planning time and plan cost of PbR and Initial. We show the results for domain queries of size 3, corresponding to the graphs in Figs. 22(c) and 23(c), that are representative of the behavior of these planners. The planning time for PbR include the four calls to Initial. The plan cost of Initial is the average of the four initial plans from which PbR started its search. Although Initial is faster than PbR, the quality of the plans it generates is very poor. For this example the cost of the initial plans is up to an order of magnitude worse than the PbR plans. PbR significantly improves the quality of these initial plans bringing them close to the optimal.

In summary, these experiments have shown the good scalability and cost optimization properties of our local-search transformational approach for query planning in mediators embodied in our PbR-based query planner. We have analyzed empirically the performance of PbR scaling along the three dimensions that contribute to the complexity of query planning in mediators, namely, the query size, the size of the integration axioms, and the number of alternative axioms. We have shown that PbR compares favorably to both classical AI planning techniques such as Sage and database query optimization techniques such as Dynamic Programming. PbR is scalable and produces high-quality plans.

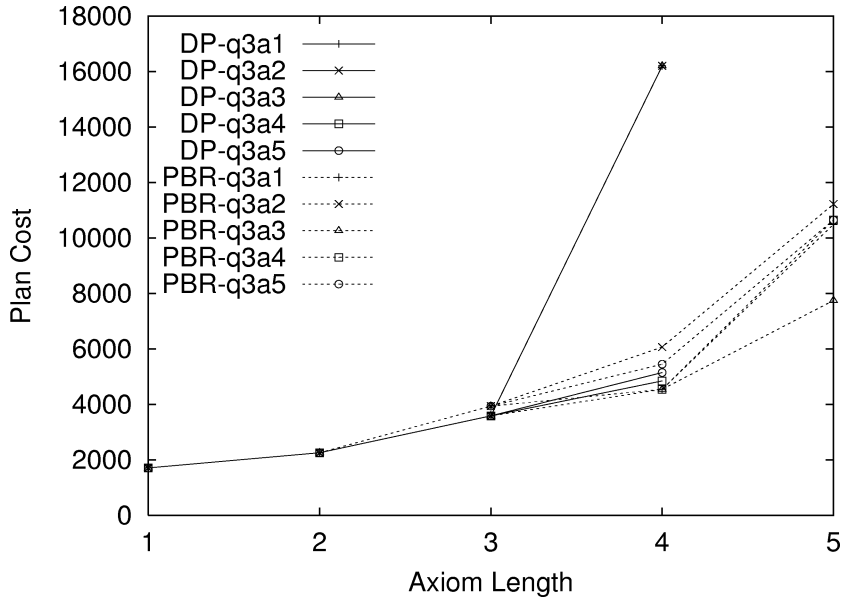


(a)

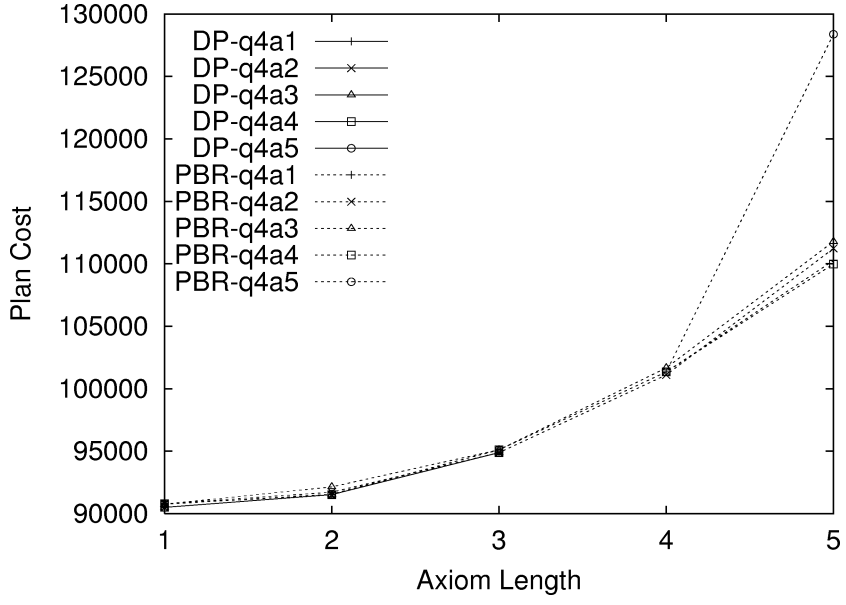


(b)

Fig. 23. Scaling the integration axioms: Plan cost. (a) Domain query size = 1 class. Source query size = up to 5 classes. (b) Domain query size = 2 classes. Source query size = up to 10 classes.

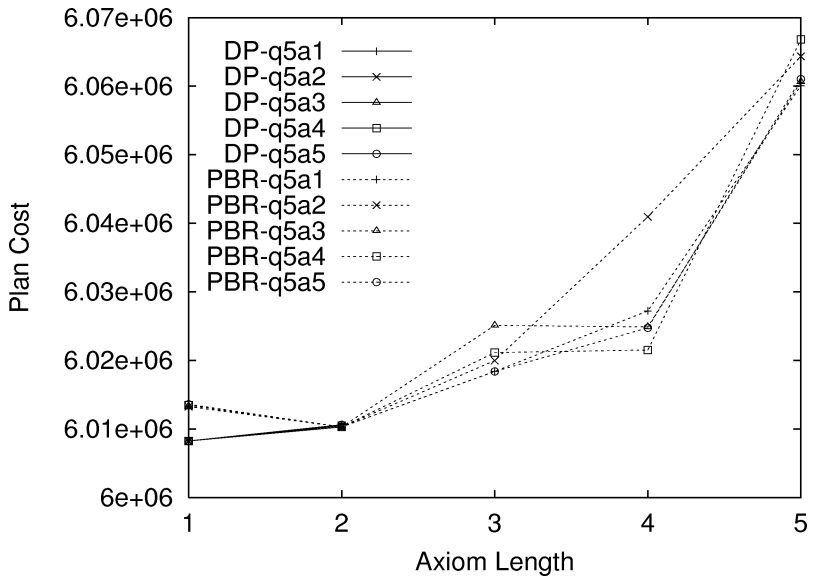


(c)



(d)

Fig. 23. Scaling the integration axioms: Plan cost (continued). (c) Domain query size = 3 classes. Source query size = up to 15 classes. (d) Domain query size = 4 classes. Source query size = up to 20 classes.



(e)

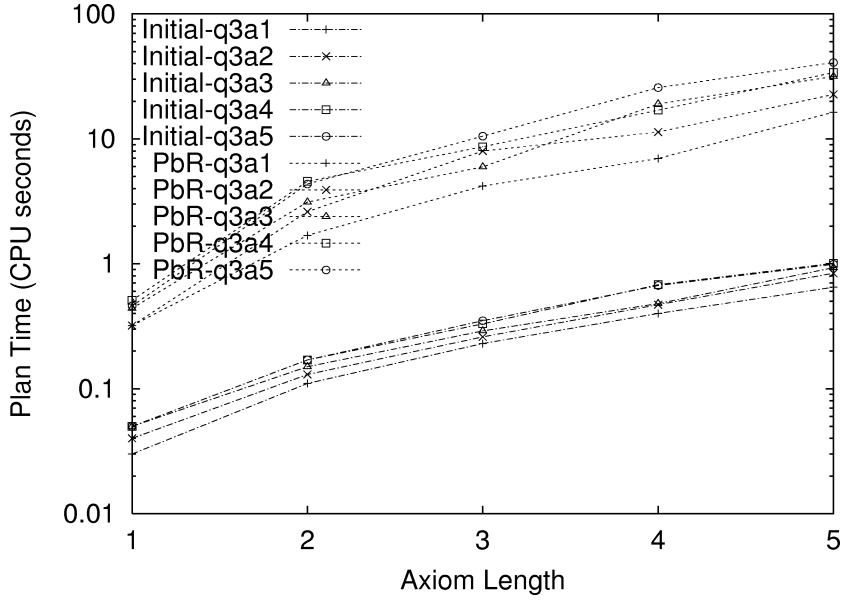
Fig. 23. Scaling the integration axioms: Plan cost (continued). (e) Domain query size = 5 classes. Source query size = up to 25 classes.

7. Related work

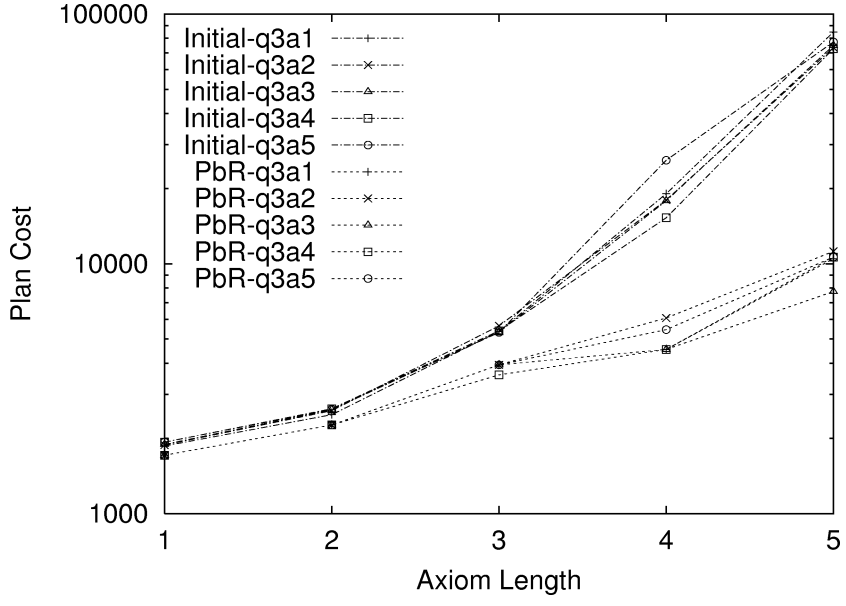
In this section, we discuss related approaches for query planning both in traditional query optimization and in query planning in mediators.

7.1. Traditional query optimization

In the database literature, query optimization has been extensively studied [24,26,37]. Query optimizers search for the most efficient algebraic form of a query and choose specific methods to implement each data processing operation [23]. For example, a join can be performed by a variety of algorithms, such as nested loops, merge-join, 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, since PbR is a declarative framework our query planning domain can be refined without much effort. For example, the `join operator` could be extended as shown in Fig. 25, where the implementation methods for the join are modeled by an additional variable and a supporting predicate (*method-appropriate*). Additionally, properties of the queries can also be modeled explicitly. For example, Fig. 25 also shows how we can model the fact that a merge-join leaves the resulting query sorted. Rewriting rules can be used



(a)



(b)

Fig. 24. Scaling the integration axioms: PbR and initial. (a) Planning time. (b) Plan cost.

```
(define (operator join)
  :parameters (?join-method ?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)
          (method-appropriate ?query ?join-method))
  :effect (and (available local sims ?query !result)
               (when (:eq ?join-method 'merge-join)
                     (sorted ?query !result))))
```

Fig. 25. Refining operators (join).

```
(define-rule :name join-by-merge-join
  :if (:operators ((?n1 (join 'algebra ?query ?jc ?query1 ?query2)))
       :constraints ((unsorted ?query1) (unsorted ?query2)))
  :replace (:operators (?n1))
  :with (:operators
         ((?n2 (sort ?query3 ?query1))
          (?n3 (sort ?query4 ?query2))
          (?n4 (join 'merge-join ?query ?jc ?query3 ?query4)))))
```

Fig. 26. Replacing an algebraic operator by implementation operators.

to substitute purely algebraic operators by implementation methods, possibly inserting additional operators. For example, in Fig. 26 a join operator without an specified join method is substituted by a merge-join preceded by two sort operators on the input tables. Similarly, the rewriting rules can exchange implementation operators. For example, Fig. 27 shows a rule that replaces a nested-loops implementation of a join operator by a merge-sort implementation when the input queries are sorted. In summary, although previous systems have a greater coverage of implementation operators and query language features, the declarativeness in our system facilitates its extension. For example, we could add to our system without much effort the rewriting rules for distributed query optimization in [13], or the rules for optimization of aggregation operators in [69,70]. A greater number and more detailed specification of implementation methods would contribute to the complexity of the query planning problem. However, our local-search approach is likely to scale better with this increase in complexity than competing approaches such as dynamic programming that perform a more exhaustive search.

Our system follows the transformational approach to query optimization in the style of Cascades [25], Volcano [27,29], and Exodus [28], in contrast to the generative (dynamic-programming) enumeration style exemplified by Starburst [31,49]. Closest to our approach is the Cascades optimizer that allows declarative rules, mixing logical and physical operators in a query plan, and transformations that replace multiple operators [25]. However, they use a heuristic search strategy as opposed to the randomized local search

```
(define-rule :name nested-loops-by-merge-join-when-sorted
  :if (:operators
        ((?n1 (join 'nested-loops ?query ?jc ?query1 ?query2)))
       :constraints ((sorted ?query1) (sorted ?query2)))
  :replace (:operators (?n1))
  :with (:operators
          ((?n4 (join 'merge-join ?query ?jc ?query1 ?query4)))))
```

Fig. 27. Exchanging implementation operators.

that we used in PbR. We have not attempted to produced a heuristically-guided search method for the query planning domain in PbR, but nothing in the framework prevents it. Another search module would fit naturally in the architecture.

Local search techniques have been applied in query optimization of centralized databases. In [34] iterative improvement, simulated annealing, and a combination of the two, called two phase optimization are applied to large join queries. 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 [60,61] 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 easily incorporated.

7.2. Query planning in mediators

A number of projects have focussed on query planning for mediators. For example, the Information Manifold [47], Infomaster [19], TSIMMIS [33], HERMES [2], Garlic [32, 56,57], and Tukwila [36]. TSIMMIS, Infomaster, 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 stages. First, these systems would find a set of retrievable plans. Second, each of these plans would be optimized independently. Finally, the best plan in this set would be selected. However, such an algorithm would not scale for our integration and query language. Because our integration axioms use equality, the number of alternative rewritings would cause a combinatorial explosion in the first stage.

HERMES does consider 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 on an extensible or declarative query optimization framework, such as our PbR-based query planner.

The Garlic system [32,56,57] is the most complete implementation of cost optimization for mediators. In addition to traditional operators, their system considers remote evaluation at the information sources by adding the `PushDown` operator, which is similar to our `retrieve` operator, and higher-level expansion rules (STARs) such as `RepoJoin` that

reflects the capability of a source to perform joins and serves a function similar to our `remote-join-eval` (cf. [32]). Their rules are implemented procedurally, which is flexible and efficient, but this requires more effort to develop, maintain, and extend than our declarative rewriting rules. In the same way that Garlic builds upon the dynamic-programming plan enumeration approach of Starburst and DB2 [22] and extends it to mediator systems, our PbR-based optimizer applies the transformational approach of Volcano (combined with local search) to cost optimization in mediators.

The Tukwila mediator [36] has a sophisticated execution engine that incorporates run-time query re-optimization and operators optimized for integration tasks. Their architecture separates the translation from a domain query to a retrievable plan [7] from the optimization of the plan. But as they use containment in their integration language, their system produces a single maximally-contained retrievable plan. This plan can possibly be a very large union of conjunctive queries, but they have developed operators and evaluation techniques tailored to such queries. In this paper we have focussed on a different type of optimizations. Since the integration language of the SIMS mediator is based on equivalence, a domain query can typically be translated into several *alternative* retrievable plans. In our system the search for the relevant sources (out of the many alternatives), and the choice of data processing operations and their order is performed in a single space, as opposed to the two-stage approach of Tukwila.

Despite the practical importance of query planning, there has not been much work in the AI planning literature either in traditional query planning or in query planning for mediators. Occam [44], its successor Razor [21], and Emerac [45] are planners for information gathering in distributed and heterogeneous domains that focus on the source selection problem. Our work combines both source selection and traditional cost-based query optimization. Sage [41] also addresses source selection and cost-based optimization, but PbR is more scalable as shown in Section 6.

Our integration axioms and the rules derived from them are related to Hierarchical-task Network (HTN) planning [20,62]. 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 as transformations applied during the local search as opposed to generative planning. Interestingly, the plan optimization grammar rules of Starburst [49] are also related to HTN planning (although, to our knowledge, this has not been noticed neither in the planning nor the database communities). The STARs are equivalent to task expansion definitions. The difference being that Starburst is computing the possible plans bottom-up using dynamic programming (with pruning based on cost and plan features), as opposed to the top-down task expansion typical in HTN planners.

8. Discussion

We have presented the application of the Planning by Rewriting framework to the challenging domain of query planning in mediators. As a result we have developed a

[7] A retrievable plan is a plan that only mentions source predicates.

novel combination of traditional query optimization and source selection. PbR explores this integrated optimization space using efficient local search methods that make the planner scalable to large queries and domains with complex axioms and large numbers of alternative sources. Moreover, our approach is very flexible and extensible because it is based on a general domain-independent planning paradigm that uses declarative specifications for the plan operators and rewriting rules, and it follows a modular design.

We plan to pursue several areas of future work. First, many advanced techniques in the local search literature can be adapted and expanded 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. For example, Fig. 17 hints at a situation in which a complex transformation can be specified as a program of simple rewriting rules. A sequence of `Join-Swap` transformations may place two retrieve operators on the same database together in a query tree and then the `Remote-Join-Eval` rule 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 rewriting-rule programs are presented in the query optimizer for object-oriented languages in [11,12].

Second, another area for further research is the interplay of query planning and plan execution. There are several types of interplay that have been explored in the database literature, but we believe that a general planning framework such as PbR will provide a more principled and general interleaving of planning and execution. The first type of interplay of run-time information and query planning is dynamic query optimization [14, 30]. Dynamic query evaluation plans include several alternative subplans which are chosen for execution depending on run-time conditions. From the planning perspective, this is a simple form of contingency planning [16,55]. A second type is query scrambling [36,38, 65]. As subqueries are answered during the execution of a query plan, the system can refine the cost 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. A rewriting-based planner as PbR is perfectly suited for this type of tasks. Third, interleaved planning and execution is also necessary in order to deal effectively with unexpected situations in the environment such as database or network failures. Approaches based on domain-independent planning provide a principled way of responding to such events [6,40]. Finally, a system like PbR allows for the inclusion of information gathering actions in the query plans [9].

Third, we plan to address more sophisticated models of the capabilities of the sources. In fact, we are in the process of adding support for binding 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 need as input the ticker symbol of a company. Our algorithm for compilation of integration axioms already supports binding patterns (see [5]). We have implemented a planning domain that incorporates an operator similar to the `bind-join` of [32], that the Sage planner routinely uses for accessing Web sources. However, we have not yet defined the rewriting rules for plans with binding patterns. In particular we need to modify the rules

that manipulate the joins, such as `join-swap` to be aware of the binding constraints and bind-join operators.

Applying PbR to more complex query planning domains and to other applications will surely provide new challenges and the possibility of discovering and transferring general planning techniques from one domain to another.

Acknowledgements

This work was supported in part by the United States Air Force under contract number F49620-98-1-0046, in part by the National Science Foundation under grant numbers IRI-9313993 and IRI-9610014, 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, and F30602-98-2-0109, in part by the Integrated Media Systems Center, a NSF Engineering Research Center, and in part by research grants from NCR and General Dynamics Information Systems. José Luis Ambite was also supported in part by a Fulbright/Ministerio of Educación y Ciencia of Spain scholarship, The views and conclusions contained in this article are the authors' and should not be interpreted as representing the official opinion or policy of any of the above organizations or any person connected with them.

References

[1] A. Aboulnaga, S. Chaudhuri, Self-tuning histograms: Building histograms without looking at data, in: A. Delis, C. Faloutsos, S. Ghandeharizadeh (Eds.), Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD-99), SIGMOD Record, Vol. 28 (2), ACM Press, New York, 1999, pp. 181–192.

[2] S. Adali, K. Selcuk Candan, Y. Papkonstantinou, V.S. Subrahmanian, Query caching and optimization in distributed mediator systems, SIGMOD Record (ACM Special Interest Group on Management of Data) 25 (2) (1996) 137–148.

[3] J.L. Ambite, Planning by Rewriting, Ph.D. Thesis, University of Southern California, Marina del Rey, CA, 1998.

[4] J.L. Ambite, C.A. Knoblock, Planning by rewriting: Efficiently generating high-quality plans, in: Proc. AAAI-97, Providence, RI, 1997.

[5] J.L. Ambite, C.A. Knoblock, I. Muslea, A. Philpot, Compiling source descriptions for efficient and flexible information integration, J. Intelligent Information Systems (to appear).

[6] J. Ambros-Ingerson, IPEM: Integrated planning, execution, and monitoring, Ph.D. Thesis, Department of Computer Science, University of Essex, 1987.

[7] Y. Arens, C.Y. Chee, C.-N. Hsu, C.A. Knoblock, Retrieving and integrating data from multiple information sources, Internat. J. Intelligent and Cooperative Information Systems 2 (2) (1993) 127–158.

[8] Y. Arens, C.A. Knoblock, W.-M. Shen, Query reformulation for dynamic information integration, J. Intelligent Information Systems (Special Issue on Intelligent Information Integration) 6 (2–3) (1996) 99–130.

[9] N. Ashish, C.A. Knoblock, A. Levy, Information gathering plans with sensing actions, in: S. Steel, R. Alami (Eds.), Recent Advances in AI Planning: 4th European Conference on Planning, ECP'97, Springer, New York, 1997.

[10] R. Brachman, J. Schmolze, An overview of the KL-ONE knowledge representation system, Cognitive Sci. 9 (2) (1985) 171–216.

[11] M. Cherniack, S.B. Zdonik, Rule languages and internal algebras for rule-based optimizers, SIGMOD Record (ACM Special Interest Group on Management of Data) 25 (2) (1996) 401–412.
[12] M. Cherniack, S.B. Zdonik, Changing the rules: Transformations for rule-based optimizers, in: Proc. ACM SIGMOD International Conference on Management of Data, Seattle, WA, 1998, pp. 61–72.
[13] W.W. Chu, P. Hurley, Optimal query processing for distributed database systems, IEEE Trans. Comput. 31 (9) (1982) 835–850.
[14] R.L. Cole, G. Graefe, Optimization of dynamic query evaluation plans, SIGMOD Record (ACM Special Interest Group on Management of Data) 23 (2) (1994) 150–160.
[15] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Maier, D. Suciu, Querying XML data, Bull. Technical Committee on Data Engineering 22 (3) (1999) 27–34.
[16] D. Draper, S. Hanks, D. Weld, Probabilistic planning with information gathering and contingent execution, in: Proc. 2nd International Conference on Artificial Intelligence Planning Systems, Chicago, IL, 1994, pp. 31–36.
[17] O.M. Duschka, Query planning and optimization in information integration, Ph.D. Thesis, Stanford University, 1997.
[18] O.M. Duschka, M.R. Genesereth, Answering recursive queries using views, in: Proc. 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, AZ, 1997.
[19] O.M. Duschka, M.R. Genesereth, Infomaster—An information integration tool, in: Proc. Internat. Workshop on Intelligent Information Integration, Freiburg, Germany, 1997.
[20] K. Erol, D. Nau, J. Hendler, UMCP: A sound and complete planning procedure for hierarchical task-network planning, in: Proc. 2nd International Conference on Artificial Intelligence Planning Systems, Chicago, IL, 1994, pp. 249–254.
[21] M. Friedman, D.S. Weld, Efficiently executing information-gathering plans, in: Proc. IJCAI-97, Nagoya, Japan, 1997, pp. 785–791.
[22] P. Gassner, G. Lohman, K.B. Schierfer, L. Wang, Query optimization in the IBM DB2 family, Bulletin of the Technical Committee on Data Engineering (Special Issue on Query Processing in Commercial Database Systems) 16 (4) (1993) 4–18.
[23] G. Graefe, Query evaluation techniques for large databases ACM Computing Surveys 25 (2) (1993) 73–170.
[24] G. Graefe (Ed.), Special Issue on Query Processing in Commercial Database Systems, Bulletin of the Technical Committee on Data Engineering 16 (4) (1993).
[25] G. Graefe, The Cascades framework for query optimization, Bulletin of the Technical Committee on Data Engineering (Special Issue on Database Query Processing) 18 (3) (1995) 19–29.
[26] G. Graefe (Ed.), Special Issue on Database Query Processing, Bulletin of the Technical Committee on Data Engineering 18 (3) (1995).
[27] G. Graefe, R.L. Cole, D.L. Davison, W.J. McKenna, R.H. Wolniewicz, Extensible query optimization and parallel execution in Volcano, in: J.C. Freytag, G. Vossen, D. Maier (Eds.), Query Processing for Advanced Database Applications, Morgan Kaufmann, San Francisco, CA, 1994, pp. 305–381.
[28] G. Graefe, D.J. DeWitt, The EXODUS optimizer generator, in: Proc. 1987 ACM SIGMOD International Conference on Management of Data, SIGMOD Record 16 (3) (1987) 160–172.
[29] G. Graefe, W.J. McKenna, The volcano optimizer generator: Extensibility and efficient search, in: Proc. IEEE International Conference on Data Engineering, Vienna, Austria, 1993, pp. 209–218.
[30] G. Graefe, K. Ward, Dynamic query optimization plans, ACM SIGMOD Record 18 (2) (1989). Also published in/as: 19th ACM SIGMOD Conference on the Management of Data, Portland, OR, May–June 1989.
[31] L.M. Haas, J.C. Freytag, G.M. Lohman, H. Pirahesh, Extensible query processing in Starburst, in: J. Clifford, B.G. Lindsay, D. Maier (Eds.), Proc. 1989 ACM SIGMOD International Conference on Management of Data, Portland, OR, ACM Press, New York, 1989, pp. 377–388.
[32] L.M. Haas, D. Kossmann, E.L. Wimmers, J. Yang, Optimizing queries across diverse data sources, in: Proc. 23rd International Conference on Very Large Data Bases (VLDB-97), 1997, pp. 276–285.
[33] J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom, Information translation, mediation, and Mosaic-based browsing in the TSIMMIS system, in: Proc. ACM SIGMOD International Conference on Management of Data, San Jose, CA, 1995.
[34] Y. Ioannidis, Y.C. Kang, Randomized algorithms for optimizing large join queries, in: Proc. ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, 1990, pp. 312–321.

[35] Y.E. Ioannidis, S. Christodoulakis, On the propagation of errors in the size of join results, SIGMOD Record (ACM Special Interest Group on Management of Data) 20 (2) (1991) 268–277.
[36] Z.G. Ives, D. Florescu, M. Friedman, A. Levy, D.S. Weld, An adaptive query execution system for data integration, in: A. Delis, C. Faloutsos, S. Ghandeharizadeh (Eds.), Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD-99), SIGMOD Record 28 (2) (1999) 299–310.
[37] M. Jarke, J. Koch, Query optimization in database systems, ACM Computing Surveys 16 (2) (1984) 111–152.
[38] N. Kabra, D.J. DeWitt, Efficient mid-query re-optimization of sub-optimal query execution plans, in: Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD-98), SIGMOD Record 27 (2) (1998) 106–117.
[39] S. Kambhampati, C.A. Knoblock, Q. Yang, Planning as refinement search: A unified framework for evaluating the design tradeoffs in partial order planning, Artificial Intelligence 76 (1–2) (1995) 167–238.
[40] C.A. Knoblock, Planning, executing, sensing, and replanning for information gathering, in: Proc. IJCAI-95, Montreal, Quebec, 1995.
[41] C.A. Knoblock, Building a planner for information gathering: A report from the trenches, in: Proc. 3rd International Conference on Artificial Intelligence Planning Systems, Edinburgh, Scotland, 1996.
[42] C.A. Knoblock, S. Minton, J.L. Ambite, A.G. Philpot, N. Ashish, P.J. Modi, I. Muslea, S. Tejada, Modeling web sources for information integration, in: Proc. AAAI-98, Madison, WI, 1998.
[43] N. Kushmerick, Wrapper induction for information extraction, Ph.D. Thesis, Department of Computer Science and Engineering, University of Washington, 1997.
[44] C.T. Kwok, D.S. Weld, Planning to gather information, in: Proc. AAAI-96, Portland, OR, 1996.
[45] E. Lambrecht, S. Kambhampati, S. Gnanaprakasam, Optimizing recursive information gathering plans, in: Proc. IJCAI-99, Stockholm, Sweden, 1999.
[46] A.Y. Levy, A.O. Mendelzon, Y. Sagiv, D. Srivastava, Answering queries using views, in: Proc. 14th ACM Symposium on Principles of Database Systems, San Jose, CA, 1995.
[47] A.Y. Levy, A. Rajaraman, J.J. Ordille, Querying heterogeneous information sources using source descriptions, in: Proc. 22th International Conference on Very Large Data Bases, Bombay, India, 1996.
[48] A.Y. Levy, D. Srivastava, T. Kirk, Data model and query evaluation in global information systems, J. Intelligent Information Systems (Special Issue on Networked Information Discovery and Retrieval) 5 (2) (1995) 121–143.
[49] G.M. Lohman, Grammar-like functional rules for representing query optimization alternatives, in: H. Boral, P.-Å. Larson (Eds.), Proc. 1988 ACM SIGMOD International Conference on Management of Data, Chicago, IL, ACM Press, New York, 1988, pp. 18–27.
[50] R. MacGregor, The evolving technology of classification-based knowledge representation systems, in: J. Sowa (Ed.), Principles of Semantic Networks: Explorations in the Representation of Knowledge, Morgan Kaufmann, San Mateo, CA, 1990.
[51] S. Minton, Quantitative results concerning the utility of explanation-based learning, Artificial Intelligence 42 (2–3) (1990) 363–392.
[52] I. Muslea, S. Minton, C.A. Knoblock, Wrapper induction for semistructured, web-based information sources, in: Proc. Conference on Automated Learning and Discovery Workshop on Learning from Text and the Web, Pittsburgh, PA, 1998.
[53] K. Ono, G.M. Lohman, Measuring the complexity of join enumeration in query optimization, in: D. McLeod, R. Sacks-Davis, H.-J. Schek (Eds.), Proc. 16th International Conference on Very Large Data Bases, Brisbane, Queensland, Australia, Morgan Kaufmann, San Mateo, CA, 1990, pp. 314–325.
[54] J.S. Penberthy, D.S. Weld, UCPOP: A sound, complete, partial order planner for ADL, in: Proc. 3rd International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA, 1992, pp. 189–197.
[55] M. Peot, D. Smith, Conditional nonlinear planning, in: J. Hendler (Ed.), Proc. First International Conference on AI Planning Systems, College Park, MD, Morgan Kaufmann, San Mateo, CA, 1992, pp. 189–197.
[56] M.T. Roth, M. Arya, L.M. Haas, M.J. Carey, W. Cody, R. Fagin, P.M. Schwarz, J. Thomas, E.L. Wimmers, The Garlic project, SIGMOD Record (ACM Special Interest Group on Management of Data) 25 (2) (1996) 557–558.
[57] M.T. Roth, P.M. Schwarz, Don't scrap it, wrap it! A wrapper architecture for legacy data sources, in: Proc. 23rd International Conference on Very Large Data Bases (VLDB-97), 1997, pp. 266–275.

[58] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, Englewood Cliffs, NJ, 1995.
[59] A. Silberschatz, H.F. Korth, S. Sudarshan, Database System Concepts, McGraw-Hill, New York, 1997.
[60] A. Swami, Optimization of large join queries: Combining heuristic and combinatorial techniques, in: Proc. ACM SIGMOD International Conference on Management of Data, Portland, OR, 1989, pp. 367–376.
[61] A. Swami, A. Gupta, Optimization of large join queries, SIGMOD Record (ACM Special Interest Group on Management of Data) 17 (3) (1988) 8–17.
[62] A. Tate, Generating project networks, in: Proc. IJCAI-77, Cambridge, MA, 1977, pp. 888–893.
[63] A. Tomasic, L. Rashid, P. Valduriez, Scaling access to heterogeneous data sources with DISCO, IEEE Trans. Knowledge and Data Engineering 10 (5) (1998) 808–823.
[64] J.D. Ullman, Information integration using logical views, in: Proc. 6th International Conference on Database Theory, Delphi, Greece, 1997.
[65] T. Urhan, M.J. Franklin, L. Amsaleg, Cost based query scrambling for initial delays, in: Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD-98), SIGMOD Record 27 (2) (1998) 130–141.
[66] D.S. Weld, An introduction to least commitment planning, AI Magazine 15 (4) (1994).
[67] D.S. Weld, Recent advances in AI planning, AI Magazine 20 (2) (1999).
[68] G. Wiederhold, Mediators in the architecture of future information systems, IEEE Computer (1992).
[69] W.P. Yan, P.-Å. Larson, Performing group-by before join, in: A.K. Elmagarmid, E. Neuhold (Eds.), Proc. 10th International Conference on Data Engineering, Houston, TX, IEEE Computer Society Press, 1994.
[70] W.P. Yan, P.-Å. Larson, Eager aggregation and lazy aggregation, in: D. McLeod, R. Sacks-Davis, H. Schek (Eds.), Proc. 21th International Conference on Very Large Data Bases, Zurich, Switzerland, 1995.
[71] V. Zadorozhny, L. Bright, L. Rashid, T. Urhan, M.E. Vidal, Efficient evaluation of queries in a mediator for web sources, Technical Report, UMIACS, University of Maryland, 1999.