Optimizing Recursive Information Gathering Plans in EMERAC

Subbarao Kambhampati*, Eric Lambrecht,
Ullas Nambiar, Zaiqing Nie, Gnanaprakasam Senthil
Department of Computer Science & Engineering, Arizona State University, Tempe, AZ , USA 85287

Abstract. In this paper we describe two optimization techniques that are specially tailored for information gathering. The first is a greedy minimization algorithm that minimizes an information gathering plan by removing redundant and overlapping information sources without loss of completeness. We then discuss a set of heuristics that guide the greedy minimization algorithm so as to remove costlier information sources first. In contrast to previous work, our approach can handle recursive query plans that arise commonly in the presence of constrained sources. Second, we present a method for ordering the access to sources to reduce the execution cost. This problem differs significantly from the traditional database query optimization problem as sources on the Internet have a variety of access limitations and the execution cost in information gathering is affected both by network traffic and by the connection setup costs. Furthermore, because of the autonomous and decentralized nature of the Web, very little cost statistics about the sources may be available. In this paper, we propose a heuristic algorithm for ordering source calls that takes these constraints into account. Specifically, our algorithm takes both access costs and traffic costs into account, and is able to operate with very coarse statistics about sources (i.e., without depending on full source statistics). Finally, we will discuss implementation and empirical evaluation of these methods in *Emerac*, our prototype information gathering system.

Keywords: Data Integration, Information Gathering, Query Optimization

1. Introduction

The explosive growth and popularity of the world-wide web have resulted in thousands of structured queryable information sources on the Internet, and the promise of unprecedented information-gathering capabilities to lay users. Unfortunately, the promise has not yet been transformed into reality. While there are sources relevant to virtually any user-queries, the morass of sources presents a formidable hurdle to effectively accessing the information. One way of alleviating this problem is to develop *information gatherers* (also called mediators) which take the user's query, and develop and execute an effective *information gathering plan*, that accesses the relevant sources to answer the user's query efficiently.[1] Figure 1 illustrates the typical architecture of such a system for integrating diverse information sources on the

* This research is supported in part by NSF young investigator award (NYI) IRI-9457634, Army AASERT grant DAAH04-96-1-0247, and NSF grant IRI-9801676. We thank Selçuk Candan for many helpful comments. Preliminary versions of parts of this work have been presented at IJCAI [31], and workshops on Intelligent Information Integration [30, 26].

internet. Several first steps have recently been taken towards the development of a theory of such gatherers in both database and artificial intelligence communities.

The information gathering problem is typically modeled by building a virtual global schema for the information that the user is interested in, and describing the accessible information sources as materialized views on the global schema.[2] The user query is posed in terms of the relations of the global schema. Since the global schema is virtual (in that its extensions are not stored explicitly anywhere), computing the query requires rewriting (or "folding" [39]) the query such that all the extensional (EDB) predicates in the rewrite correspond to the materialized view predicates that represent information sources. Several researchers [33, 39, 27] have addressed this rewriting problem. Recent research by Duschka and his co-workers [10, 11] subsumes most of this work, and provides a clean methodology for constructing information gathering plans for user queries posed in terms of a global schema The plans produced by this methodology are "maximally contained" in that any other plan for answering the given query is contained in them.[3]

Generating source complete plans however is only a first step towards efficient information gathering. A crucial next step, which we focus on in this paper, is that of query plan optimization. Maximally contained plans produced by Duschka's methodology are conservative in that they in essence wind up calling any information source that may be remotely relevant to the query. Given the autonomous and decentralized nature of the Internet, sources tend to have significantly overlapping contents (e.g. mirror sources), as well as varying access costs (premium vs. non-premium sources, high-traffic vs. low-traffic sources). Naive execution of maximally contained plans will access all potentially relevant sources and be prohibitively costly, in terms of the network traffic, response time, and access costs (in the case of "premium" sources that charge for access).

At first blush, it would seem that we should be able to directly apply the rich body of work on query optimization in databases [5] to solve this problem. Unfortunately, this does not work because many of the assumptions made in the traditional database query optimization do not hold in information gathering scenarios. To begin with, in traditional databases, redundancy and overlap among different sources is not a major issue, while it is a very crucial issue in information gathering. Similarly, traditional query optimization methods depend on elaborate statistical models (histograms, selectivity indices etc.) of the underlying databases. Such statistical models may not be easily available for sources on the Internet.[4] Finally, even the work on optimizing queries in the presence of materialized views (c.f. [6]) is not directly applicable as in such work materialized views are assumed to be available *in addition* to the main database. In contrast, the global database in information gathering is "virtual" and the only accessible information resides in materi-

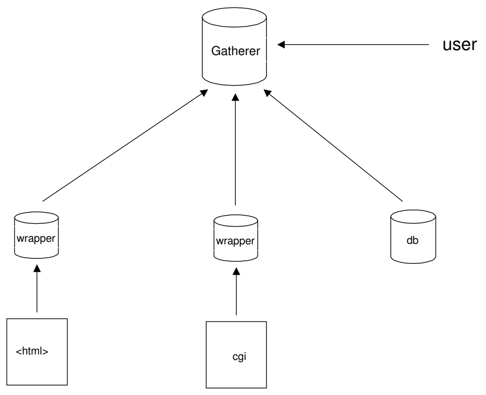


Figure 1. The information gatherer acts as an intermediary between the user and information sources on the Internet.

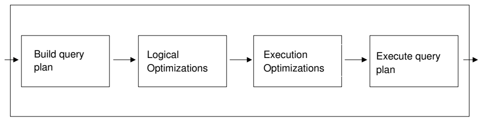


Figure 2. The full process of query planning

alized views whose statistical models are not easily available. For all these reasons, it is now generally recognized (c.f. [17]) that query optimization for information gathering is a very important open problem.

In this paper we describe the query optimization techniques that we have developed in the context of *Emerac*, a prototype information gathering system under development. Figure 2 provides a schematic illustration of the query planning and optimization process in *Emerac*. We start by generating a query plan using the source inversion techniques described by Duschka [10, 11]. This polynomial time process gives us a "maximally contained" query plan which serves as the input for the optimization methods. As in traditional databases, our optimization phase involves two steps: logical optimization and execution optimization. In traditional databases, logical optimization involves rewriting a query plan, using relational algebra equivalences, to make it more efficient; while execution optimization involves steps such as ordering the access to the base relations to make computations of joins cheaper. For *Emerac*, the logical optimization step involves minimizing the maximally contained query plan such that access to redundant sources is removed. Execution optimization involves ordering the access to the information sources in the minimized plan so as to reduce execution cost.

Our contributions: For logical optimization, we present a technique that operates on the recursive plans generated by Duschka's algorithm and greedily minimizes it so as to remove access to costly and redundant information sources, without affecting the completeness of the plan. For this purpose, we use the so-called localized closed world (LCW) statements that characterize the completeness of the contents of a source relative to either the global (virtual) database schema or the other sources. Our techniques are based on an adaptation of Sagiv's [40] method for minimizing datalog programs under uniform equivalence. Although there exists some previous research on minimizing information gathering plans using LCW statements [9, 18], none of it is applicable to minimization of information gathering plans containing recursion. Our ability to handle recursion is significant because recursion appears in virtually all information gathering plans either due to functional dependencies, binding constraints on information sources, or recursive user queries [10]. Additionally, in contrast to existing methods, which do pairwise redundancy checks on source accesses, our approach is capable of exploiting cases where access to one information source is rendered redundant by access to a combination of sources together. Large performance improvements in our prototype information gatherer, *Emerac*, attest to the cost-effectiveness of our minimization approach.

Ultimately plan execution in our context boils down to doing joins between the sources efficiently. When gathering information on the Internet, we typically cannot instruct two sources to join with each other. It is thus necessary to order the access to the sources. The existing methods for subgoal ordering assume that the plan is operating on a single "fully relational" (i.e., no binding restrictions) database, and that the plan execution cost is dominated by the number of tuples transferred. In contrast, sources on the Internet have a variety of access limitations and the execution cost in information gathering is affected significantly by the connection setup costs. We describe a way of representing the access capabilities of sources, and provide a greedy algorithm for ordering source calls that respects source limitations, and takes both access costs and traffic costs into account, without requring full source statistics.

Although there exist other research efforts that address source redundancy elimination and optimization in the presence of sources with limited capabilities, *Emerac* is the first to consider end-to-end issues of redundancy elimination and optimization in recursive information gathering plans. It is also the first system system to consider the source access costs as well as traffic costs together in doing optimization.

Organization: The rest of the paper is organized as follows. Section 2 provides the background on information integration and describes the motivation for our work. Section 3, we review the work on integrating diverse information sources by modeling them as materialized views on a virtual database. We pay special attention to the work of Duschka [10, 11], which forms the basis for our own work. Section 4 briefly reviews the use of LCW statements and

Sagiv's algorithm for datalog program minimization under uniform equivalence. Section 5 presents our greedy minimization algorithm that adapts Sagiv's algorithm to check for source redundancy in the context of the given LCW statements. We also explain how the inter-source subsumption relations can be exploited in addition to LCW statements (Section 5.1). We then discuss the complexity of the minimization and present heuristics for biasing the greedy minimization strategy. Section 6.1 describes our algorithm for ordering source accesses during execution. Section 7 describes the architecture of *Emerac*, our prototype information gatherer, and presents an empirical evaluation of the effectiveness of our optimization techniques. Section 8 discusses related work. Section 9 presents our conclusions, and outlines our current research directions.

2. Background and Motivation

2.1. Background

In order to place the work on *Emerac* in the proper context, it is important to provide a broad classification of the prior work on information integration. Information integration (aka data integration) has received a significant amount of attention in the recent years, and several systems have been developed. These include InfoMaster [21], Information Manifold [33], Garlic [22], TSIMMIS [20], HERMES [3], and DISCO [1]. The similarities and differences among these systems can be broadly understood in terms of (1) the approach used to relate the mediator and source schemas and (2) the type of application scenario considered by the system.

The application scenarios considered to date can be usefully classified into two categories:

Authorized integration of databases: Integrating a set of heterogeneous database systems owned by a given corporation/enterprise.

Information Gathering: Integrating a set of information sources that export information related to some specific application area (e.g. comparison shopping of books, integration of multiple bibliography sources etc.).

In the first case, we would expect that the set of data sources are relatively stable, and that the mediation is "authorized" (in that the data sources are aware that the they are being integrated). In the second, "information gathering" scenario, the set of data sources may be changing, and more often than not, the mediation may not have been explicitly authorized by the sources. Systems such as Garlic [22], TSIMMIS [20], HERMES [3], and DISCO [1] can be characterized as aiming at authorized database integration, while InfoMaster [21], Information Manifold [33], Occam [27], Razor [18] as well as the *Emerac* system presented in this paper address the information gathering scenario.

Although related in many ways, authorized database integration and information gathering systems do differ in two important ways–the way source and mediator schemas are modeled, and the types of approaches used for query optimization.

There are two broad approaches for modeling source and mediator schemas [23]. The "global as view" (GAV) approach involves modeling the mediator schema as a view on the (union of) source schemas. The "local as view" (LAV) approach involves modeling the source schemas as views on the mediated schema. The GAV approaches make query planning relatively easy as a query posed on the mediated schema can directly be rewritten in terms of sources. The LAV approach, in contrast, would require a more complex rewriting phase to convert the query posed on the mediator schema to a query on the sources.

The advantage of LAV approach however is that adding new sources to a mediator involves just modeling them as views on the mediated schema. In contrast, in the GAV approach, the mediated schema has to be rewritten every time a new source is added. Systems that address authorized integration of known databases, such as Garlic, Disco, TSIMMIS and HERMES use the GAV approach as they can be certain of a relatively stable set of sources. In contrast, systems aimed at information gathering, such as the Information Manifold [33] and InfoMaster [21] use the LAV approach.

The other main difference among information integration systems is the types of approaches used for query optimization. Systems addressing authorized integration of known databases can count on the availability of statistics on the databases (sources) being integrated. Thus Garlic [22], TSIMMIS [8], HERMES [3] and DISCO [41] systems attempt to use cost-based optimization algorithms for query planning. Systems addressing information gathering scenarios, on the other hand, cannot count on having access to statistics about the information sources. Thus, either the mediator has to learn the statistics it needs, or will have to resort to optimization algorithms that are not dependent on complete statistics about sources. Both Infomaster and Information Manifold system use heuristic techniques for query optimization.

Within the above classification, *Emerac* is aimed at information gathering. Thus, it is most closely related to systems such as InfoMaster [21], Information Manifold [33], Occam [27] and Razor [18]. Like these other systems, *Emerac* too uses the LAV approach to model source and mediator schemas, and uses heuristic techniques for query optimization. The specific contributions of *Emerac* over the other systems are:

– A systematic approach for handling minimization of (recursive) datalog query plans using the LCW information.

– A heuristic optimization technique for query plans that takes into account both the access and transfer costs.

3. Building Query Plans: Background

Suppose our global schema contains the world relation $advisor(S, A)$, where A is the advisor of S. Furthermore, suppose we have an information source ADDB, such that for every tuple (S, A) returned by it, A is the advisor of S. This can be represented as a materialized view on the global schema as follows:

ADDB*(S,A)* $\rightarrow$ *advisor(S, A)*

We make the "open world assumption," (OWA) on the sources [2], meaning that the ADDB source has some but not necessarily all of the tuples satisfying the *advisor* relation.

Suppose we want to retrieve all the students advised by Weld. We can represent our goal by the query $\mathcal{Q}$:

query(S, A) :- *advisor(S, A)* $\wedge$ *A = "Weld"*

Dushcka et. al. [10, 11] show how we can generate an information gathering plan that is "maximally contained" in that it returns every query-satisfying tuple that is stored in any of the accessible information sources. This method works by *inverting* all source (materialized view) definitions, and adding them to the query. The inverse, v^{-1}, of the materialized view definition with head *v(*$X_1, \ldots, X_m$*)* is a set of logic rules in which the body of each new rule is the head of the original view, and the head of each new rule is a relation from the body of the original view. When we invert our definition above, we get:

advisor(S,A) :- ADDB*(S,A)*

When this rule is added to the original query $\mathcal{Q}$, we effectively create a datalog[5] program whose execution produces all the tuples satisfying the query.

Note that we are modeling sources as "conjunctive views" on the mediated schema. The complexity of finding the maximally contained plan depends on the expressiveness of the language used to describe sources. Duschka and Abiteboul [2] show that as long as sources are described as conjunctive views on the mediated schema, and we use the open world assumption on the sources, maximally contained plans can be found in polynomial time. The complexity becomes NP-hard when the sources are written as disjunctive, and undecidable when the sources are written as recursive views on the mediated schema.

Constrained sources & Recursion: The materialized view inversion algorithm can be modified in order to model databases that have binding pattern requirements. Suppose we have a second information source, CONDB that requires the student argument to be bound, and returns the advisor of that given student. We denote this in its view as follows:

CONDB*($S, A)* $\rightarrow$ *advisor(S, A)*

The '$' notation denotes that *S* must be bound for any query sent to CONDB. A straightforward inversion of this source will get us a rule of the form:

advisor(S,A) :- CONDB*($S, A)*

which is unexecutable as S is not bound. This is handled by making up a new relation called dom whose extension is made to correspond to all possible constants that can be substituted for S. In our example, assuming that we have both the ADDB source and the CONDB source, the complete plan for the query, which we shall refer to as $\mathcal{P}$, is:

r1: query(S, A)	:-	*advisor(S , A)* $\wedge$ *A="Weld"*
r2: advisor(S, A)	:-	ADDB*(S, A)*
r3: advisor(S, A)	:-	*dom(S)* $\wedge$ CONDB*(S, A)*
r4: dom(S)	:-	ADDB*(S, A)*
r5: dom(A)	:-	ADDB*(S, A)*
r6: dom(A)	:-	*dom(S)* $\wedge$ CONDB*(S, A)*

Notice that all extensional (EDB) predicates in the progam correspond to source predicates (materialized views). Notice also the presence of $dom(S)$ relation in the rule $r3$. Rules $r4, r5$ and $r6$ define the extension of dom by collecting all possible constants that can be derived from source calls. Finally, note that rule $r6$ is recursive, which makes the overall plan recursive, *even though* the original query as well as the source description are non-recursive. Given the ubiquitousness of constrained sources on the Internet, it is thus important that we know how to handle recursive information gathering plans.

It is worth noting that the complexity of finding maximally contained plans remains polynomial when we have sources with access constraints. The only change is that the query plan itself is going to be a recursive datalog program. This change can in turn significantly increase the execution cost of the plans. Consequently, we focus on using any information about the source overlap to minimize the query plan and remove the recursion as much as possible.

4. Plan minimization preliminaries

The plan $\mathcal{P}$ above accesses two different advisor databases to answer the query. It would be useful to try and cut down redundant accesses, as this would improve the execution cost of the plan. To do this however, we need more information about the sources. While the materialized view characterizations of sources explicate the world relations that are respected by each tuple returned by the source, there is no guarantee that all tuples satisfying those properties are going to be returned by that source.

One way to support minimization is to augment the source descriptions with statements about their relative coverage, using the so-called localized closed world (LCW) statements [14]. An LCW statement attempts to characterize what information (tuples) the source is *guaranteed* to contain in terms of the global schema. Suppose, we happen to know that the source ADDB is guaranteed to contain all the students advised by Weld and Hanks. We can represent this information by the statement (note the direction of the arrow):

$$\text{ADDB}(S, A) \leftarrow advisor(S, A) \wedge A = \text{“Weld”}$$
$$\text{ADDB}(S, A) \leftarrow advisor(S, A) \wedge A = \text{“Hanks”}$$

Pair-wise rule subsumption: Given the LCW statement above, intuitively it is obvious that we can get all the tuples satisfying the query $\mathcal{Q}$ by accessing just $adDB$. We now need to provide an automated way of making these determinations. Suppose we have two datalog rules, each of which has one or more materialized view predicates in its body that also have LCW statements, and we wish to determine if one rule subsumes the other. The obvious way of checking the subsumption is to replace the source predicates from the first rule with the bodies of their view description statements, and the source predicates from the second rule with the bodies of the LCW statements corresponding to those predicates. We now have the transformed first rule providing a “liberal” bound on the tuples returned by that rule, while the transformed second rule gives a “conservative” bound. If the conservative bound subsumes the liberal bound, i.e., if the transformed second rule “contains” (entails) the transformed first rule, we know that second rule subsumes the first rule. Duschka [9] shows that this check, while sufficient, is not a necessary condition for subsumption. He proposes a modified version that involves replacing each source predicate s with $s \wedge v$ in the first rule, and with $s \vee l$ in the second rule, where v is the view description of s, and l is the conjunction of LCW statements of s. If after this transformation, the second rule contains the first, then the first rule is subsumed by it.[6]

Minimization under uniform equivalence: Pair-wise rule subsumption checks alone are enough to detect redundancy in non-recursive plans [32, 18], but are inadequate for minimizing recursive plans. Specifically, recursive plans correspond to infinite union of conjunctive queries and checking if a particular rule of the recursive plan is redundant will involve trying to see if that part is subsumed by any of these infinite conjuncts [42, pp. 908]. We instead base our minimization process on the notion of "uniform containment" for datalog programs, presented in [40]. To minimize a datalog program, we might try removing one rule at a time, and checking if the new program is equivalent to the original program. Two datalog programs are equivalent if they produce the same result for all possible assignments of EDB predicates [40]. Checking equivalence is known to be undecidable. Two datalog programs are uniformly equivalent if they produce the same result for all possible assignments of EDB *and IDB* predicates. Uniform equivalence is decidable, and implies equivalence. Sagiv [40] offers a method for minimizing a datalog program under uniform equivalence that we illustrate by an example (and later adapt for our information gathering plan minimization). Suppose that we have the following datalog program:

$$
\begin{array}{lll}
r1: p(X) & :- & p(Y) \wedge j(X, Y) \\
r2: p(X) & :- & s(Y) \wedge j(X, Y) \\
r3: s(X) & :- & p(X)
\end{array}
$$

We can check to see if *r1* is redundant by removing it from the program, then instantiating its body to see if the remaining rules can derive the instantiation of the head of this rule through a simple bottom-up evaluation [42]. Our initial assignment of relations is *p("Y"), j("X", "Y")* . If the remaining rules in the datalog program can derive *p("X")* from the assignment above, then we can safely leave rule *r1* out of the datalog program. This is indeed the case. Given *p("Y")* we can assert *s("Y")* via rule *r3*. Then, given *s("Y")* and *j("X", "Y")*, we can assert *p("X")* from rule *r2*. Thus the above program will produce the same results without rule *r1* in it.

5. Greedy Minimization of Recursive plans

We now adapt the algorithm for minimizing datalog programs under uniform equivalence to remove redundant sources and unnecessary recursion from the information gathering plans. Our first step is to transform the query plan such that the query predicate is directly related to the source calls. This is done by removing global schema predicates, and replacing them with bodies of source inversion rules that define those predicates (see [42, Sec. 13.4]).[7]

```
Replace all global schema predicates in $\mathcal{P}$
  with bodies of their inversion rules.
repeat
    let $r$ be a rule in $\mathcal{P}$ that has not yet been considered
    let $\hat{\mathcal{P}}$ be the program obtained by deleting rule $r$ from $\mathcal{P}$
    and simplifying it by deleting any unreachable rules.
    let $\hat{\mathcal{P}}'$ be $\hat{\mathcal{P}}[s \mapsto s \vee l]$
    let $r'$ be $r[s \mapsto s \wedge v]$
    if there is a rule, $r_i$ in $r'$,
    such that $r_i$ is uniformly contained by $\hat{\mathcal{P}}'$
      then replace $\mathcal{P}$ with $\hat{\mathcal{P}}$
    until each rule in $\mathcal{P}$ has been considered once
```

Figure 3. The greedy plan minimization algorithm

Our example plan $\mathcal{P}$, from Section 3, after this transformation with the LCW statements in Section 4 looks as follows:

r2: query(S, A)	:-	*adDB(S , A)* $\wedge$ *A=“Weld”*
r3: query(S, A)	:-	*dom(S)* $\wedge$ CONDB*(S, A)* $\wedge$ *A=“Weld”*
r4: dom(S)	:-	ADDB*(S, A)*
r5: dom(A)	:-	ADDB*(S, A)*
r6: dom(A)	:-	*dom(S)* $\wedge$ CONDB*(S, A)*

We are now ready to consider minimization. Our basic idea is to iteratively try to remove each rule from the information gathering plan. At each iteration, we use the method of replacing information source relations with their views or LCW’s as in the rule subsumption check (see previous section) to transform the removed rule into a representation of what could possibly be gathered by the information sources in it, and transform the remaining rules into a representation of what is guaranteed to be gathered by the information sources in them. Then, we instantiate the body of the transformed removed rule and see if the transformed remaining rules can derive its head. If so, we can leave the extracted rule out of the information gathering plan, because the information sources in the remaining rules guarantee to gather at least as much information as the rule that was removed. The full algorithm is shown in Figure 3.

For our example plan above, we will try to prove that rule *r3*, containing an access to the source $conDB$, is unnecessary. First we remove *r3* from our plan, then transform it and the remaining rules so they represent the information gatherered by the information sources in them. For the removed rule, we want to replace each information source in it with a representation of all the possible information that the information source could return. Specifically, we want to transform it to $r[s \mapsto s \wedge v]$. This produces:

query(S,A) :- *dom(S)* $\wedge$ CONDB*(S, A)*
$\wedge$ *advisor(S, A)* $\wedge$ *A="Weld"*

For the remaining rules, $\mathcal{P} - r_3$, we transform them into $\mathcal{P}' = (\mathcal{P} - r_3)[s \mapsto s \vee l]$, which represents the information guaranteed to be produced by the information sources in the rules. For our example, we produce:

r21: query(S, A) :- ADDB*(S, A)* $\wedge$ *A="Weld"*
r22: query(S, A) :- *advisor(S, A)* $\wedge$ *A="Weld"*
r23: query(S, A) :- *advisor(S, A)* $\wedge$ *A="Hanks"*
dom(S) :- ADDB*(S, A)*
dom(S) :- *advisor(S, A)*
dom(A) :- ADDB*(S, A)*
dom(A) :- *advisor(S, A)*
dom(A) :- *dom(S)* $\wedge$ CONDB*(S, A)*
dom(A) :- *dom(S)* $\wedge$ *advisor(S, A)*

When we instantiate the body of the transformed removed rule $r3$, we get the ground terms: *dom("S"), conDB("S", "A"), A="Weld", advisor("S", "A").* After evaluating $\mathcal{P}'$ the remaining rules given with these constants, we find that we can derive *query("S", "A")*, using the rule $r22$, which means we can safely leave out the rule $r3$ that we've removed from our information gathering program.

If we continue with the algorithm on our example problem, we will not be able to remove any more rules. The remaining *dom* rules can be removed if we do a simple reachability test from the user's query, as they are not referenced by any rules reachable from the query.

5.1. HANDLING INTER-SOURCE SUBSUMPTION RELATIONS

The algorithm above only makes use of LCW statements that describe sources in terms of the global schema. It is possible to incorporate inter-source subsumption statements into the minimization algorithm. Specifically, suppose we are considering the removal of a rule r containing a source relation s from the plan P. Let U be the set of inter-source subsumption statements that have s in the tail, and $U^{\leftarrow \mapsto :-}$ be the statements of U with the $\leftarrow$ notation replaced by $:-$ notation (so U is a set of datalog rules). We have to check if $r[s \mapsto s \wedge v]$ is uniformly contained in $(P - r + U^{\leftarrow \mapsto :-})[s \mapsto s \vee l]$ If so, then we can remove r.

As an example, suppose we know that $s1$ and $s2$ are defined by the views:

s1(x) $\rightarrow$ *r(x)*
s2(x) $\rightarrow$ *r(x)*

Suppose we know that s1 contains all tuples that $s2$ contains. This corresponds to the statement

s1(x) $\leftarrow$ *s2(x)*

Suppose we have the query:

Q(x) :- *r(x)*

The corresponding maximally contained plan P will be:

Q(x) :- *r(x)*
r(x) :- *s1(x)*
r(x) :- *s2(x)*

To recognize that we can remove third rule from this plan because of the source subsumption statements, we check if that rule is uniformly contained in the program $(P - r + \text{"}s1(x)\text{: } -\mathrm{s2(x)})''$, which is:

Q(x) :- *r(x)*
r(x) :- *s1(x)*
s1(x) :- *s2(x)*

The uniform containment holds here since if we add the tuple $s2(A)$ to the program, bottom up evaluation allows us to derive $s1(A)$ and subsequently $r(A)$, thus deriving the head of the removed rule.

5.2. Heuristics for ordering rules for removal

The final information gathering plan that we end up with after executing the minimization algorithm will depend on the order in which we remove the rules from the original plan. In the example given in Section 5.1, suppose we had another LCW statement:

CONDB*(S, A)* $\leftarrow$ *advisor(S, A)*

In such a case, we could have removed *r2* from the original information gathering plan $\mathcal{P}$, instead of removing *r3*. Since both rules will lead to the generation of the same information, the removal would succeed. Once *r2* is removed however, we can no longer remove *r3*. This is significant, since in this case, a plan with rule *r3* in it is much costlier to execute than the one with rule *r2* in it. The presence of *r3* triggers the *dom* recursion through rules *r4* $\cdots$ *r6*, which would have been eliminated otherwise. Recursion greatly increases the execution cost of the plan, as it can generate potentially boundless number of accesses to remote sources (see Section 7). We thus consider for elimination rules containing non-recursive predicates before those containing recursive predicates (such as *dom* terms). Beyond this, we also consider any gathered statistics about the access costs of the sources (such as contact time, response time, probability of access etc.) to break ties [29].

Complexity of Minimization: The complexity of the minimization algorithm in Figure 3 is dominated by the cost of uniform containment checks. As Sagiv [40] points out, the running time of the uniform containment check is in the worst case exponential in the size of the query plan being minimized. However, things are brighter in practice since the exponential part of the complexity comes from the "evaluation" of the datalog program. The evaluation here is done with respect to a "small" database – consisting of the grounded literals of the tail of the rule being considered for removal. Nevertheless, the exponential complexity justifies our greedy approach for minimization, as finding a globally minimal plan would require considering all possible rule-removal orders.

6. Plan Execution

Once the datalog query plan has been minimized to remove any redundant source accesses, *Emerac* attempts to execute the minimized plan. In this section, we describe how the techniques for datalog plan execution (c.f. [42]) are adapted to the information gathering scenario to efficiently execute *Emerac*'s information gathering plans.

Two efficient approaches for executing datalog programs are (1) top-down relational evaluation and (2) bottom-up evaluation with magic sets transformation [42]a. In *Emerac* we use the top-down relational evaluation. The top-down relational evaluation scheme attempts to avoid the inefficiencies of the top-down tuple-by-tuple evaluation scheme by directly manipulating relations (c.f. [42, Algorithm 12.17]. The standard version of this scheme involves generating a rule/goal graph for the datalog program and evaluating the graph until fix point. To make this evaluation feasible as well as more efficient, a "conjunct ordering" algorithm is used to re-order the conjuncts [35].

In order to adapt the top-down relational evaluation to information gathering, we make the following extensions to it:

– We provide a framework for modeling the access restrictions on the source relations. These restrictions include attributes that must be bound in order to access the relation, as well as those attributes whose bindings will be ignored by the source.

– We describe a novel conjunct ordering approach that takes into consideration the access restrictions, and qualitative costs in reordering the rules (and thereby the rule/goal graphs).

In the rest of this section, we elaborate on these two extensions. We should mention here that in addition to these main changes, we also make another minor but important change to the evaluation of the rule goal graph. The plan is executed by traversing the relational operator graph. When ever a *union* node is encountered during traversal of the rule/goal graph, new threads of execution are created to traverse the children of the node in parallel. Use of separate threads allows us to reduce the response time as well as return answers to the user asynchronously.

6.1. Ordering source calls during Execution

A crucial practical choice we have to make during the evaluation of datalog programs is the order in which predicates are evaluated. Our objective is to reduce the "cost" of execution, where cost is a function of the access cost (including connection time), traffic costs (the number of tuples transferred), and processing cost (the time involved in processing the data). Typically, traffic and processing costs are closely correlated.

In our cost model, we assume that the access cost dominates the other terms. This is a reasonable assumption given the large connection setup delays involved in accessing sources on the Internet. While the traffic costs can also be significant, this is offset to some extent by the fact that many data sources on the Internet do tend to have smaller extractable tables.[8]

Although the source call ordering problem is similar to the "join ordering" phase in the traditional database optimization algorithms [5], there are several reasons why the traditional as well as distributed-database techniques are not suitable:

– Join ordering algorithms assume that all sources are relational databases. The sources on the Internet are rarely fully relational and tend to support limited types of queries. These limitations need to be represented and respected by the join ordering algorithm.

– Join ordering algorithms in distributed databases typically assume that the cost of query execution is dominated by the number of tuples transferred during execution. Thus, the so-called "bound-is-easier" assumption makes good sense. In the Internet information gathering scenario, the cost of accessing sources tends to dominate the execution cost. Consequently, we cannot rely solely on the bound-is-easier assumption and would need to consider the number of source calls.

– Typically, join ordering algorithms use statistics about the sizes of the various predicates to compute an optimal order of joining. These techniques are not applicable for us as our predicates correspond to source relations, about which we typically do not have complete statistics.

– The fact that source latencies make up a significant portion of the cost of execution argues for parallel (or "bushy") join trees instead of the "left-linear" join trees considered by the conventional algorithms [5].

6.1.1. *Representing source access capabilities*

As we mentioned, in the information gathering scenarios, the assumption that information sources are fully relational databases is not valid. An information source may now be a wrapped web page, a form interfaced database, or a fully relational database. A wrapped web page is a WWW document interfaced through a wrapper program to make it appear as a relational database. The wrapper retrieves the web page, extracts the relational information from it, and then answers relational queries. A form-interfaced database refers to a database with an HTML form interface on the web which only answers selection queries over a subset of the attributes in the database. A WWW airline database that accepts two cities and two dates and returns flight listings is an example of a form interfaced database.

In our system, we use a simple way to inform the gatherer as to what types of queries an information source would accept.[9] We use the "$" annotation to identify variables that must be bound, and "%" annotation to identify unselectable attributes (i.e., those attributes whose bindings cannot be pushed to the source to narrow down the selection). Thus a fully relational source would be adorned $source(X, Y)$, a form interfaced web-page that only accepts bindings for its first argument would be adorned $source(X, \%Y)$, while a wrapped web-page source would have all its attributes marked unselectable, represented as $source(\%X, \%Y)$. Finally, a form interfaced web-page that requires bindings for its first argument, and is able to do selections only on the second argument would be adorned as $source(\$X, Y, \%Z)$.

Often times, a single source might support multiple binding patterns. These are supported by listing all the feasible binding patterns for that source. For example, if source S_1 has two binding patterns: $S_1(\$X, Y, \%Z)$, $S_1(X, \$Y, \%Z)$, it means that S_1 can be accessed either with X bound or with Y bound. In either case, the attribute Z cannot be selected at the source (and must be filtered locally).

The "$" and "%" annotations are used to identify feasible binding patterns for queries on a source, to establish generality relations between two binding patterns, and to ensure that soundness is preserved in pushing variable selection constraints (such as "$Y = 7$") into source calls. Given a source with annotations $S_1(\$X, \%Y, Z)$, only the binding patterns of the form S_1^{b--} are feasible (where "$-$" stands for either $b$ound or $f$ree argument). Similarly, we are not allowed to push selection constraints on Y to the source S_1 (they must be filtered locally). Thus the call S_1^{bbf} is modeled as S_1^{bff} filtered locally with the binding on Y.

A binding pattern S^p is more general than S^q (written $S^p \succ_g S^q$, if every selectable (non "%"-annotated) variable that is free in q is also free in p, but not *vice versa*. Thus, for the source S_1 above, the binding pattern S_1^{bbf} is more general than the binding pattern S_1^{bfb} (while such a relation would not have held without "%" annotations). Intuitively, the more general a binding pattern, the higher the number of tuples returned by the source when called with that binding pattern. We ignore binding status of "%"-annotated variables since by definition they will not have any effect on the amount of data transferred. Finally, we define $\#(\alpha)$ as the number of bound variables in α that are not %-annotated. Notice that "$\succ_g$" holds only between binding patterns of the same source while "$\#(.)$" can be used to relate binding patterns of different sources.

6.1.2. *Plans and Costs*

In Emerac, we support simple select/project/join queries on the information sources. Given the access restrictions on the sources, plans for such queries involve computing "dependent joins" (c.f. [7]; see below) over the source relations. Moreover, it may not always be feasible to push selections to the sources. As an example, consider two data sources $S_1(\%X, Y), S_2(\$Y, Z)$ that export the relations $R_1(X, Y)$ and $R_2(Y, Z)$ respectively. Suppose we have a query

$$Q(X, Z) : -R_1(X, Y), R_2(Y, Z), X ='' a''$$

The query rewriting phase in Emerac will convert this to

$$Q(X, Z) : -S_1(X, Y), S_2(Y, Z), X ='' a''$$

It would seem that a good execution plan for the query would be to push the selection over X to S_1 and do join between the two sources (in any order). However, since S_1 has an unselectable attribute restriction on X, it is not possible to do a selection at the source. Further, since S_2 requires Y to be bound, we need to do a dependent join between S_1 and S_2. A feasible plan for this query would thus be:

$$\sigma_{X=''a''}(S_1^{ff}(X, Y)) \overset{\overset{Y}{\rightarrow}}{\bowtie} S_2^{bf}(Y, Z)$$

Here source S_1 is being called with the binding pattern ff, and the results are processed (at the mediator) with a selection on X = "a". Next, the source S_2 is called with the binding pattern bf, where calls are issued once for each unique value of Y from the left sub-tree. (This form of passing bindings from the left subtree of a join to the right subtree is called a "dependent join," and has been studied in the literature in connection with optimization in the presence of foreign functions [7].)

In computing the plans, we are thus interested in deciding not only the order in which the joins are carried out, as is the case in traditional system-R style query optimization [5], but also about what specific binding patterns are used in source calls, and how far it is feasible to push selections into query plans. We note that the execution cost is a function of the access cost (including connection time), traffic costs (the number of tuples transferred), and processing cost (the time involved in processing the data). Thus, optimal plans will need to minimize:

$$\sum_{s}(C_a^s * n_s + C_t^s * D_s)$$

where n_s is the number of times a source s has been accessed during the plan and C_a^s is the cost per access, and C_t^s is the per tuple transfer cost for source s, and D_s is the number of tuples transferred by s. We note that this cost metric imposes a tension between the desire to reduce network traffic, and the desire to reduce access costs. To elaborate, reducing the network traffic involves accessing sources with less general binding patterns. This in turn typically increases the number of separate calls made to a source, and leads to increased access cost. To illustrate this further, consider the subgoals:

$$S_1(X,Y) \wedge S_2(Y,Z)$$

Suppose that the query provides bindings for X. How should we access the sources? The conventional wisdom says that we should access S_1 first since it has more bound arguments. As a result of this access, we will have bindings for Y which can then be fed into calls to S_2. The motivation here is to reduce the costs due to network traffic. However, calling S_1 and using its outputs to bind the arguments of S_2 may also lead to a potentially large number of separate calls to S_2 (one per each of the distinct Y values returned by S_1)[10], and this can lead to a significant connection setup costs, thus worsening the overall cost. On the other hand, calling S_2 without propagating bindings from S_1 would reduce the source calls to two. We need to thus consider both the access costs and the traffic costs to optimize the ordering of the sources.

Since we often do not have the requisite statistics to do the full cost-based optimization, we propose an approach that is less dependent on full statistics. Our algorithm does not make use of a quantitative measure of access cost or transfer costs, but rather a qualitative measure of high and low cost of access to a source. We describe this approach in the next section.

6.1.3. *An algorithm for ordering source calls*

We make two important assumptions in designing our source-call ordering algorithm:

- Exact optimization of the execution cost requires access to source selectivity statistics. While such statistics may be available for intra-corporation information integration scenarios (c.f. GARLIC [22]), they are harder to get in the case of autonomous and decentralized sources on the Internet.

– We assume that by default source access costs (rather than network traffic) are the dominating cost of a query plan. This becomes reasonable given the large connection setup delays involved in accessing sources on the Internet. Many Internet sources tend to have small extractable tables which help offset the traffic costs that at times can be proportional to or greater than access cost.[11]

If our assumptions about the secondary importance of network traffic costs were always true, then we can issue calls to any source as soon as its binding constraints are met (i.e., all the variables requiring bindings have bindings available). Furthermore, we need only access the source with the most general feasible binding pattern (since this will reduce the number of accesses to the source). We do provide an escape clause for this assumption (see below), as sometimes sources can transfer arbitrarily large amounts of data for calls with sufficiently general binding patterns.

High-traffic Binding Patterns: To ensure that we don't get penalized excessively for concentrating primarily on access costs, we also maintain a table, called HTBP, of least general (w.r.t. "$\succ_g$") source binding patterns that are still known to be high-traffic producing. The general idea is to postpone calling a source as long as all the feasible binding patterns for that source supported by the currently bound variables are equal to or more general than a binding pattern listed in HTBP.

An underlying assumption of this approach is that while full source statistics are rarely available, one can easily gain partial information on the types of binding patterns that cause excessive traffic. For example, given a source that exports the relation

$$Book(Author, Title, ISBN, Subject, Price, Pages)$$

we might know that calls that do not bind at least one of the first four attributes tend to generate high traffic. The information as to which binding patterns generate high traffic could come either from source modeling phase, or could be learned with rudimentary probing techniques. The latter approach involves probing the sources with a set of sample queries, and logging for each source the binding patterns and the cardinalities of generated result sets, identifying the HTBP patterns using a threshold on the result set cardinality.

There are two useful internal consistency conditions on HTBP. First, if S^α is listed in HTBP (where α is a binding pattern on S), then every S^β where $\beta \succ_g \alpha$ is also implicitly in HTBP. Similarly, if S^α is in HTBP, then it cannot be the case that the only free variables in α are all "%"-annotated variables.

```
Inputs: FBP: table of forbidden binding patterns
  HTBP: table of high traffic binding patterns
  V := all variables bound by the head; $V'$ := $\emptyset$
  $C[1 \cdots m]$:=
  Array where $C[i]$ lists sources chosen at $i^{th}$ stage;
  $P[1 \cdots m]$:=
  Array where $P[i]$ lists sources postponed at $i^{th}$ stage
for $i$ := 1 to $m$ (where $m$ is the number of subgoals)
do begin
  $C[i]$ := $\emptyset$; $P[i]$ := $\emptyset$; $V := V \cup V'$
  for each unchosen subgoal $S$
  do begin
      $\mathcal{B}$ := All feasible binding patterns for $S$ w.r.t. $V$
            and FBP sorted using "$\succ_g$" relation.
      for each $\beta \in \mathcal{B}$
      do begin
          if $\nexists_{\beta' \in HTBP}$ s.t.$(\beta = \beta') \vee (\beta \succ_g \beta')$
          then begin
              Push $S$ with binding pattern $\beta$ into $C[i]$;
              Mark $S$ as "chosen";
              add to $V'$ all variables appearing in $S$;
          end
  end
  if $\mathcal{B} \neq \emptyset$ and $S$ is not chosen
      then Push $S^{\gamma}$ into $P[i]$, where
          $\gamma \in \mathcal{B}$ has the maximum $\#(.)$ value;
  end
  if $C[i] = \emptyset$ and $P[i] \neq \emptyset$
      then begin
          Take the source $S^{\beta} \in P[i]$ with maximum $\#(.)$
          value and push it into $C[i]$;
          add to $V$ all variables appearing in $S$;
      else fail
  end
  Return the array $C[1..i]$.
```

Figure 4. A greedy source call ordering algorithm that considers both access costs and traffic costs.

A greedy algorithm to order source calls based on these ideas appears in Figure 4. It is along the lines of "bound-is-easier" type ordering procedures [35, 33]. By default, it attempts to access each source with the most general feasible binding pattern. This default is reasonable given our assumption that access costs dominate transfer costs. The default is overridden if a binding pattern is known to produce too much traffic–by being present in HTBP (or being more general than a pattern present in HTBP).

The procedure takes as input a rule with m subgoals and a given binding pattern for its head. The input also includes FBP, a table of forbidden binding patterns for each source (constructed from the "\$" annotations), and the table HTBP, which contains all source binding patterns that are known to be high-traffic producing. At each level i, the algorithm considers the feasible binding patterns of each unchosen source from most general to least general, until one is found that is not in HTBP. If such a binding pattern is found, that source, along with that binding pattern, is added to the set of selected sources at that level (maintained in the array of sets $C[i]$). If not, the source, along with the least general feasible binding pattern (given the "\$" restrictions as well as the currently available bound variables), is temporarily postponed (by placing it in $P[i]$). If at the end of considering all unchosen sources at level i, we have not chosen at least one source (i.e., all of them have only high-traffic inducing binding patterns), then one of the sources from the list of postponed sources ($P[i]$) is chosen and placed in $C[i]$. This choice is made with the "bound-is-easier" assumption by selecting the source with the least general binding pattern (in terms of $\#(.)$).

Anytime a source is placed in $C[i]$, the set V of variables that currently have available bindings is updated.[12] This ensures that at the next stage more sources will have feasible, as well as less general, binding patterns available. This allows the algorithm to progress since less general binding patterns are also less likely to be present in the HTBP table. Specifically, when $C[i]$ is empty and $P[i]$ is non-empty, we push only one source from $P[i]$ into $C[i]$ since it is hoped that the updated V will then support non-high-traffic binding patterns at the later stages. (By consulting HTBP, and the set of unchosen sources, the selection of the source from $P[i]$ can be done more intelligently to ensure that the likelihood of this occurrence is increased).

Notice that each element of C is a (possibly non-singleton) set of source calls with associated binding patterns (rather than a single source call)–thus supporting parallel source calls which reduce the time spent on connection delays. Thus, $C[i]$ may be non-empty for only a prefix of values from 1 to m. The complexity of our ordering algorithm is $O(n^2)$ where n is the length of the rule. Note that HTBP table determines the behavior of our algorithm. An empty HTBP table makes our algorithm to focus on reducing source accesses(similar to [46]), whereas presence of all source binding patterns in HTBP table makes the algorithm focus on reducing network traffic by using a

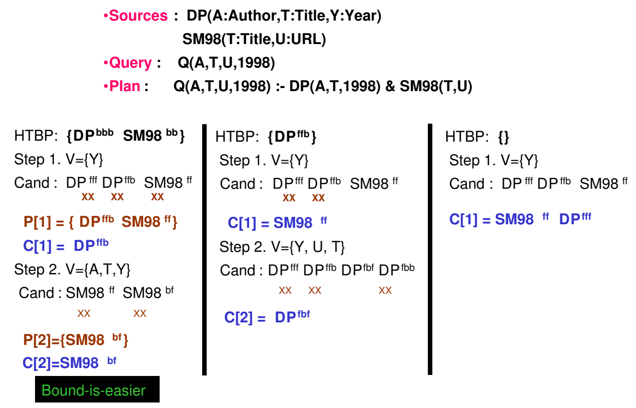


Figure 5. Example illustrating the operation of the source-call ordering procedure. Note that based on the contents of the HTBP table, the procedure can degenerate into bound-is-easier type ordering.

variant of bound-is-easier [35]. When some source patterns are present in the HTBP table our algorithm attempts to reduce both access and transfer costs, as appropriate.

6.1.4. *Example*

Figure 5 shows an example illustrating the operation of the source-call ordering procedure. We have two sources in the query plan: DP(A,T,Y), which is a source of all database papers, and SM98(T,U), which is a source of all papers from SIGMOD-98. The query binds the year to 1998, and wants the author, title and URL tuples. We will illustrate the algorithm under three different scenarios.

Case 1: In the first case, shown on the left, HTBP contains DP^{bbb}, and SM98^{bb}. Notice that this means that every possible call to these sources is considered to be high-traffic. Given that the query binds one variable, Y, the only possible source call bindings we have are: DP^{fff}, DP^{ffb}, and SM98^{ff}. Among these, the algorithm finds no possible feasible source call that is not in HTBP–all of them are more general than the source call patterns stored in HTBP. Thus, it winds up picking up DP^{ffb} since this is the one with most bound variables. At this, point, the second iteration starts with one remaining source SM98, and bindings for three variables, A, T, Y (where A and T are supplied by the first call). The two possible source calls are SM98^{ff} and SM98^{bf}, both of which are again in the HTBP. So, the algorithm picks SM98^{bf}. The query plan thus involves doing a dependent join [7] between DP^{ffb} and SM98^{bf}, with the unique titles retrieved by the DP tuples being used to invoke SM98 $(DP^{ffb}(A,T,Y) \overset{\overrightarrow{T}}{\bowtie} SM98^{bf}(T,U))$

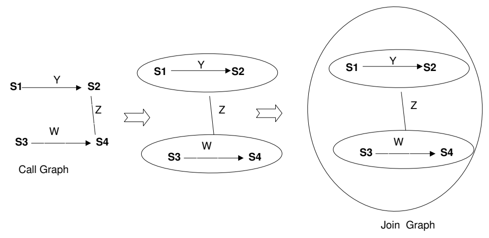


Figure 6. Converting a call graph into a join graph

Case 2: In the second case, only the call DP^{ffb} is in the HTBP. Thus, in the first iteration, neither of the DP calls are feasible, but the SM98 call is. So, SM98^{ff} is chosen. In the second iteration, we have four DP calls, of which three are in HTBP. The call DP^{fbf} is however not in HTBP, and is thus feasible. The query plan thus involves a dependent join between SM98 and DP with the URL values from SM98 call being passed to DP ($SM98^{ff}(T,U) \overset{\overrightarrow{T}}{\bowtie} DP^{fbf}(A,T,Y)$).
Case 3: In the third case, HTBP is empty. Thus, we simply pick the most general feasible source calls in the very first iteration–leading to calls SM98^{ff} and DP^{fff}. The query plan is thus a (non-dependent) join between the sources SM98 and DP ($DP^{fff}(A,T,Y) \bowtie SM98^{ff}(T,U)$).

6.1.5. *Converting the Call Graph to a Join Tree*
When the algorithm in Figure 4 terminates,the array C specifies which sources are to be called at each stage, and what binding patterns are to be used in those calls - specifically, all the source calls in $C[i]$ are issued in *parallel* before those in $C[i+1]$. There is still the matter of what is the exact *join tree* that is to be executed at the mediator to derive answers from these source calls.

Consider running the algorithm on the query with the given binding restrictions

$$Q(X,Y,W,Z) : -S_1^{ff}(X,Y) \wedge S_2^{bf}(Y,Z) \wedge S_3^{ff}(T,W) \wedge S_4^{bf}(W,Z)$$

We also assume that HTBP is empty.[13] Our algorithm will end with $C[1] = \{S_1^{ff}, S_3^{ff}\}$ and $C[2] = \{S_2^{bf}, S_4^{bf}\}$. Analyzing the common variables among the sources, it is easy to represent this as a call graph as shown in Figure 6–which has both directed and undirected arcs. Arcs correspond to shared variables between source calls. A directed arc from $S1$ to $S2$ states that a shared variable needs to be bound by $S1$ before reaching $S2$.

The join tree can be greedily derived from the call graph by combining vertices in the graph till it becomes a single node. The vertex combination is done by traversing the graph in a topologically sorted order. All the vertices that have directed arcs between them are first combined (by doing a dependent join [7] between the corresponding sources). Then the undirected arcs are processed by converting them to joins. Finally, if we are left with a disconnected graph, the corresponding subtrees are joined by a cartesian product. Accordingly converting the example in Figure 6 gives us the following plan:

$$\left(S_1^{ff}(X,Y) \overset{Y}{\overrightarrow{\bowtie}} S_2^{bf}(Y,Z)\right) \bowtie \left(S_3^{ff}(T,W) \overset{Z}{\overrightarrow{\bowtie}} S_4^{bf}(W,Z)\right)$$

As the example above shows, our approach supports bushy join trees (instead of sticking merely to left-linear joint trees). As is evidenced by this example (and pointed out first in [16]), bushy join trees allow us to avoid cartesian products in more situations than left linear join trees alone would–when there are binding restrictions on sources.

7. Implementation and Evaluation

We will start by describing the architecture and implementation of *Emerac*(Section 7.1). Next, we will discuss a variety of experiments we conducted to evaluate its effectiveness. Section 7.2 evaluates the effectiveness of the plan minimization routines described in Section 5 in terms of their costs and benefits. Section 7.3 describes the experiments we conducted to evaluate the techniques for improving the plan execution that we presented in Section 6. This section starts with an empirical validation of our assumptions about the domination of source access costs (Section 7.3.1). The source call ordering scheme is evaluated over simulated sources in Section 7.3.2, and over sources on the Internet in Section 7.3.3.

7.1. ARCHITECTURE OF *Emerac*

Emerac is written in the Java programming language, and is intended to be a library used by applications that need a uniform interface to multiple information sources. Full details of *Emerac* system are available in [28]. *Emerac* presents a simple interface for posing queries and defining a global schema. *Emerac* is internally split into two parts: the query planner and the plan executor. The default planner uses algorithms discussed in this paper, but it can be replaced with alternate planners. The plan executor can likewise be replaced, and the current implementation attempts to execute an information gathering plan in parallel after transforming it into a relational operator graph.

The query planner accepts and parses datalog rules, materialized view definitions of sources, and LCW statements about sources. Given a query, the query planner builds a source complete information gathering plan (using the method from [10]) and attempts to minimize it using the minimization algorithm presented in Section 5.

The optimized plan is passed to the plan executor, which transforms the plan into a relational operator graph. The plan executor makes use of "\$" and "%"-adornments to determine the order to access each information source in a join of multiple sources, as described in this paper. The plan is executed by traversing the relational operator graph. When a *union* node is encountered during traversal, new threads of execution are created to traverse the children of the node in parallel. Use of separate threads also allows us to return answers to the user asynchronously, facilitating return of first tuples faster.

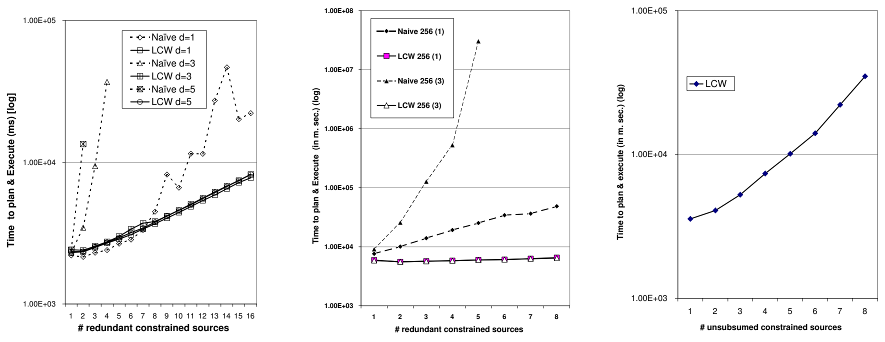
Handling Recursion during execution: Since information gathering plans can contain recursion, handling recursive plans during execution becomes an important issue. Since each recursive call to a node in the r/g graph [42] can potentially generate an access call to a remote source, evaluating a program until it reaches fix point can get prohibitively expensive. Currently, we take a practical solution to this problem involving depth-limited recursion. Specifically, we keep a counter on each node in the r/g graph to record how many times the node has been executed. When the counter reaches a pre-specified depth-limit, the node would not be executed, and an empty set will be returned to represent the result of executing the node. Since the recursion induced by the binding restrictions does not involve any negation in the tail of the rules, this strategy remains sound– i.e., will produce only correct answers.

Wrapper Interface: *Emerac* assumes that all information sources contain tuples of information with a fixed set of attributes, and can only answer simple *select* queries. To interface an information source with *Emerac*, a Java class needs to be developed that implements a simple standard interface for accessing it. The information source is able to identify itself so as to provide a mapping between references to it in materialized view and LCW definitions and its code.

In order to facilitate construction of wrappers for web pages, a tool was created to convert the finite state machine based wrappers created by SoftMealy [24] into Java source code that can be compiled into information sources usable by *Emerac*. We have successfully adapted 28 computer science faculty listing web pages wrapped with SoftMealy into information sources usable by *Emerac*.

7.2. Evaluating the Effectiveness of Plan Minimization

We used the prototype implementation of *Emerac* to evaluate the effectiveness of the optimization techniques proposed in this paper. We used two sets of experimental data. The first were a set of small artificial sources containing 5 tuples each. Our second data set was derived from the University of Trier's Database and Logic Programming (DBLP) online database, which contains bibliographical information on database-related publications. Indi-



(a) Cumulative costs of LCW vs. Naive (artificial sources)

(b) Cumulative costs of LCW vs. Naive (artificial sources)

(c) Effect of unsubsumed constrained sources on LCW

Figure 7. Results characterizing utility of minimization algorithm.

vidual sources used in the experiments corresponded to different subsets of DBLP data (ranging from 128 to 2048 tuples). In each case, some of the sources are unconstrained, while others have binding restrictions (leading to recursive plans). To normalize for differences caused by individual source implementations, we extracted the data into tables which we stored on disk as Java serialized data. All experiments were conducted using a simple wrapper (written in compiled Java) to return the contents of the serialized tables.

The sources delay answering each query for a set period of time in order to simulate actual latency on the Internet. In all our experiments, this delay was set to 2 seconds, which is quite reasonable in the context of current day Internet sources.

Utility of minimization: To see how the planner and executor performed with and without minimization, we varied the number of duplicate information sources available and relevant to the query, and compared the total time taken for optimization (if any) and execution. Given that the minimization step involves an exponential "uniform containment" check, it is important to ensure that the time spent in minimization is made up in improved execution cost. Notice that we are looking at only the execution time, and ignoring other costs (such as access cost for premium sources), which also can be reduced significantly with the minimization step. The naive method simply builds and executes source complete plans. The "LCW" method builds source complete plans, then applies the minimization algorithm described in Section 5 before executing the plans. For both methods, we support fully parallel execution at the union nodes in the r/g graph. Since in practice, recursive plans are handled with depth bounded recursion, we experimented with a variety of depth limits (i.e., the number of times a node is executed in the rule-goal graph), starting from 1 (which in essence prunes the recursion completely).

The plots in Figure 7 show the results of our experiments. Plot a is for the artificial sources, and shows the relative time performances of LCW against the naive algorithm when the number of redundant constrained sources is increased. In this set of experiments, LCW statements allow us to prove all constrained sources to be redundant, and the minimization algorithm prunes them. The y-axis shows the cumulative time taken for minimization and execution. We note that the time taken by the LCW algorithm remains fairly independent of recursion depth as well as number of constrained sources. The naive algorithm, in contrast, worsens exponentially with increasing number of constrained sources. The degradation is more pronounced for higher recursion depths, with the LCW method outperforming the naive one when there are two or more redundant constrained sources. Plot b repeats the same experiment, but with the sources derived from the DBLP data. The sources are such that the experimental query returns upto 256 tuples. The experiment is conducted for recursion depth limits 1 and 3. We note once again, that LCW method remains fairly unaffected by the presence of redundant constrained sources, while the naive method degrades exponentially. Plot c considers DBLP data sources in a scenario where some constrained sources are left unsubsumed after the minimization. As expected, LCW performance degrades gracefully with increased number of constrained sources. Naive algorithm would not have shown such graceful degradation as no sources would be removed through subsumption.

7.3. Evaluating the Effectiveness of Source Call Ordering

In this section, we report on a set of experiments conducted on both artificial and real Internet sources to evaluate the effectiveness of our algorithm (referred to as HT). We compare the performance with two other greedy approaches for join ordering namely Bound-is-easier (BE) and Reduced Access (RA). For sources where HTBP information cannot be ascertained and hence is not available, HT will work as RA. The aim of our experiments is to demonstrate that our algorithm outperforms algorithms that concentrate on reducing only tuple transfer cost (as in BE) or only source access cost (as in RA). BE corresponds roughly to the algorithm used in Information Manifold [33] while RA is similar to the algorithm proposed in [46] for reducing source access costs. The experiments also indirectly show that the kind of coarse statistics that we assume in our algorithm are likely to be available easily in practice.

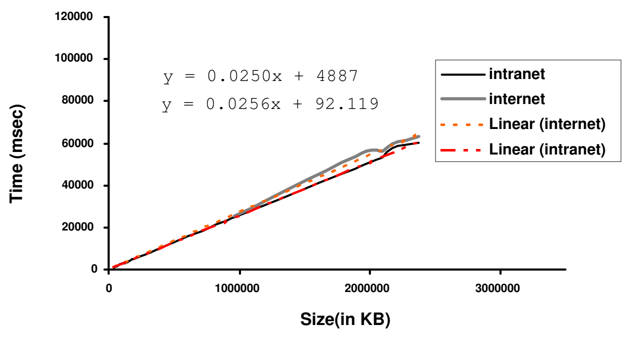


Figure 8. Comparison of Access and Transfer Cost

7.3.1. **Verifying the dominance of Access Cost**

An important assumption we made in developing the execution algorithm is about dominance of source access cost over tuple transfer cost for sources on the Internet. In this section we describe a simple empirical validation of this hypothesis. We consider access cost as the cost incurred in setting up a connection to a source. Transfer cost is incremental cost for each transaction. We considered two sources, an internet source http://dvs1.dvlabs.com/adcritic/ and a local intranet source. We recorded the time taken to download various files with sizes ranging from 10KB to 25MB from both the sources. Each byte transfer is considered as a transaction. Average values were taken after 4 rounds of data transfer for each file.

Though access(a) and transfer time(t) for a source are not known, they can be calculated from a plot of downloading time vs. filesize as given in Figure 8. Source access time (a) is the y-intercept and tuple transfer time (t) is the slope of the graph.

As can be seen from Figure 8 the access cost for internet source (nearly 5 sec) is much higher than that for intranet source (92 msec). Tuple transfer time, on the other hand is nearly same for both types of sources. This validates our assumption that for data sources residing on the Internet, source access cost is considerably higher than tuple transfer cost.

7.3.2. *Source Call Ordering Results for Artificial Sources*

Below we present results from the performance evaluation of HT, BE and RA algorithms for queries over sources mimicking Internet sources. We derived the sources from the relations present in the “Enterprise Schema” used in [13]. The sources are designed as Java servlets accessible from a server on the Intranet. The servlets accept a query and return relevant tuples extracted from data stored in flat files. To model the latency exhibited by Internet sources, we delay the response from the servlets. The delay is proportional to the number of tuples in the resultset. Some of the servlets were designed to also mimic the bursty behaviour (i.e., sending sets of tuples interspersed with delays [43]) shown by some Internet sources. A detailed description of sources in terms of their attributes and their forbidden binding patterns is given below:

$Employee^{ffffffff}(N : Name, S : SSN, B : DateofBirth, A : Address, X : Sex, Y : Salary, U : Manager, D : DeptNo)$

$Dept^{ffff}(Dn : DeptName, D : DeptNo, U : Manager, T : MrgStartDate)$

$Deptloc^{ff}(D : DeptNo, Dc : Location)$

$Workson^{fff}(S : SSN, P : ProjectID, H : Hours)$

$Project^{ffff}(Pn : ProjectName, P : ProjectId, Dc : Location, D : DeptNo).$

The subscripts on source names describe forbidden binding pattern derived from "$" and "%" annotations for the sources. Intuitively, FBP and HTBP will have fewer patterns than the set of feasible binding patterns for a source. Hence we use them to represent the infeasible/costly binding patterns thereby reducing the look up time for our experiments.

Table I. HTBP for Artificial Sources

Employee	Dept	Deptloc	Workson	Project
f,f,f,f,f,f,f,f		b,b	b,f,f	f,f,b,f
f,f,f,f,f,f,f,b		b,f	f,b,f	f,f,b,b
f,f,f,f,f,f,b,b		f,b	f,f,b	

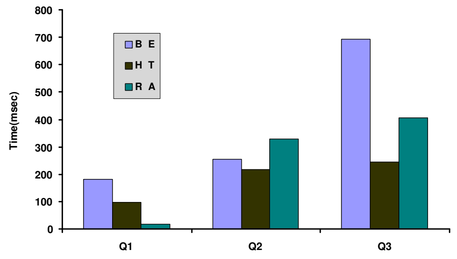


Figure 9. Comparison of algorithms w.r.t Execution Time for Q1,Q2,Q3

The Table I lists source binding patterns that generate large resultsets for most of the values given to the bound attributes. Each column in Table I is an HTBP table. Thus given Deptloc(D,"Houston") and the HTBP table of Deptloc, one can see that pushing the value Location="Houston" to the source Deptloc will result in high traffic. The source Dept has an empty HTBP table. Hence any query with a binding pattern that is not forbidden for Dept can be

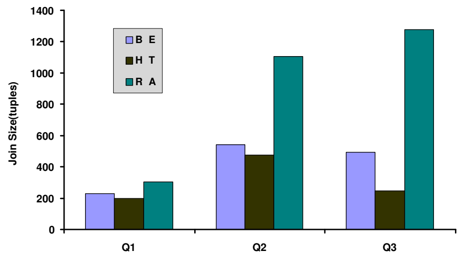


Figure 10. Comparison of algorithms w.r.t Join Size for Q1,Q2,Q3

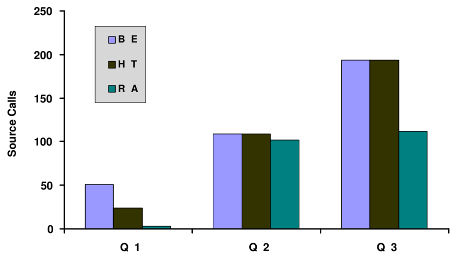


Figure 11. Comparison of algorithms w.r.t # of Source Calls for Q1,Q2,Q3

issued to Dept. But we cannot assume a source with no HTBP to have low cost. Therefore we deliberately modeled Dept to simulate a source with high response time.

The graphs in Figures 9, 10 and 11 compare the average values of execution time, size of result sets and number of source calls for BE, HT and RA. The results were obtained by running queries Q1, Q2 and Q3 using these algorithms. The queries were generated with 3 different binding patterns per source and 5 different binding values per binding pattern. The times for running the source call ordering algorithms themselves were minute in comparison with the execution costs.

$Query1 : Q(N, S, B, A, X, Y, U, D, Dn, U, T, Dc)$
$Plan : Q(N, S, B, A, X, Y, U, D, Dn, U, T, Dc) : -$
Employee(N,S,B,A,X,Y,U,D),Dept(Dn,D,U,T), Deptloc(D,Dc).

$Query2 : Q(D, Dc, S, P, H, Pn)$
$Plan : Q(D, Dc, S, P, H, Pn) : -$
Deptloc(D,Dc),Workson(S,P,H),Project(Pn,P,Dc,D).

$Query3 : Q(Dn, D, U, T, Dc, S, P, H, Pn)$
$Plan : Q(Dn, D, U, T, Dc, S, P, H, Pn) : -$
Dept(Dn,D,U,T),Deptloc(D,Dc),Workson(S,P,H), Project(Pn,P,Dc,D).

From Figure 11, we can see that RA always optimizes the Status: O number of source calls and has the least number of source calls for all 3 queries. BE on the other hand focuses on having smaller result sets and thus to reduce the transfer cost, as is evident from Figure 10. But no approach is a clear winner when the total execution cost is considered (see Figure 9). Our HT algorithm gives lowest execution cost for 2 out of the 3 cases considered. We can see that HT tries to reduce both number of source calls (Figure 11) and/or resultset size (Figure 10) while executing the queries. For Q2 and Q3, HT has source number of source calls equal to BE but smaller result sets compared to BE and RA and hence achieves lower execution cost. HT strikes a middle ground compared to BE and RA and tries to optimize both access cost and transfer cost. Thus HT generates low cost execution plans more often than BE and RA.

The case where RA is better than HT in Figure 9 shows that HTBP is not always perfect. We consider a binding pattern as HTBP if it generates large resultsets for most binding values (without regard to attribute selectivities). It could well be the case that for a specific instantiation of the binding pattern (i.e. for particular value(s) of attribute(s)), a HTBP may not generate high traffic. RA which does not consider HTBP thus makes source calls using this binding pattern and emerges a winner.

7.3.3. *Source Call Ordering Results for Internet Sources*

The next set of tests were done on data sources derived from the DBLP Bibliography of AI and Database papers maintained by Michael Ley at http://dblp.uni-trier.de. We use a simple scenario with the execution plan using only 2 sources:

$Dp1^{fbff,ffff}$ (A:Author, Co:Co-Author, C:Conference, Y:Year)

$Dp2^{ffff}$ (A:Author, T:Title, C:Conference, Y:Year)

We derive two sources Dp1 and Dp2 shown above from the DBLP Bibliography by projecting the corresponding relations from DBLP. Specifically, we developed a wrapper over DBLP that accepts queries over relations of Dp1 and Dp2, forwards them to DBLP and extracts the relevant relation from the resultset given by DBLP. Queries accepted by the wrapper have to satisfy the FBP associated with the projected relation. The subscripts on source names give the forbidden binding patterns (FBP). HTBP for these sources are shown in Table II. These HTBP statements can be determined by rudimentary probing techniques. Specifically, we execute queries on sources with various binding pattern/value combinations and log the resultset sizes and execution times. Given similar resultset sizes for two sources, we cannot deduce that both binding patterns are HTBP since the sources may have differing processing power and database size. Hence we store the execution time for queries on a source and use these to determine the average response time and resultset

Table II. HTBP for Real Sources Dp1 and Dp2

Dp1(A,Co,C,Y)	Dp2(A,T,C,Y)
F,B,B,B	F,F,F,B
	F,F,B,F
	F,B,F,F
	B,F,F,F

Table III. Results of accessing Dp1, DP2 using BE and HT

	BE	HT
Source calls	20.8/15.3	12.7/11.5
Join size	8.8/9.0	8.2/7.9
Time	30.5/21.8	18.8/16.4

size returned by the source. Any binding pattern that generates larger result sets than average resultset size or has considerably higher response time than the average case is considered HTBP. The binding pattern restriction for Dp1 is Dp1 ($A, %Co, C, Y) and that for Dp2 is Dp2 ($A, T, C, Y). The "%" annotation describes that the attribute has to be filtered locally. But both the sources must bind the attribute 'Author' to retrieve tuples. The experimental setup is thus:

$Query : Q(A, Co, T, C, Y)$
$Plan : Q(A, Co, T, C, Y) :-$ Dp2(A,T,C,Y),Dp1(A,Co,C,Y)

Table III shows the performance of various algorithms for queries over Dp1 and Dp2. Performance is measured as the number of source calls made, resultset size (Joinsize) and the query response time (Time). The time incurred in executing the source call ordering algorithms were negligible compared to the execution costs. The results show that HT performs better than BE for these sources. The example also shows that the HTBP statistics can be collected for real Internet sources and that our algorithm does perform better than other existing source call algorithms.

8. Related Work

As we mentioned, systems that consider integration in the context of information gathering use LAV approach to model sources. In the LAV approach, sources are seen as materialized views over the mediated schema. The user query, posed in terms of the mediator schema, has to be re-written solely in terms of source calls. Although this problem, on the surface, is similar to query rewriting in terms of materialized views [6], there are several important differences that complicate the query rewriting:

– We are only interested in rewritings that are entirely in terms of source relations.

– The sources may not contain all tuples satisfying their view definition. This leads to the so-called open-world assumption, and changes the objective of query planning from finding a "sound and complete" query plan to finding a "sound and maximally contained" [10] query plan (a query plan P for the query Q is maximally contained if there is no other query plan P' for Q that can produce more answers for Q using the same set of sources).

– The materialized views represented by the sources may have a variety of access restrictions. In such a case, the maximally contained query plan may be a "recursive" query plan (or equivalently, an infinite union of conjunctive plans).

IM [33] and Occam [27] are among the first systems to consider query planning in the LAV approach. Both these systems search through the space of conjunctive query plans, to find sound rewritings of the user query, and optimizing them for efficient execution.

There are two problems with the approaches used by IM and Occam, when some sources have access restrictions. To begin with, as shown by Duschka et. al. [10, 11], maximally contained plans will be recursive when we have sources with access restrictions. The approach of finding sound conjunctive query plans, used by IM and Occam essentially "unfolds" the recursion[14], forcing them to handle infinite unions of conjunctive plans. IM gets around this by sacrificing guarantees of maximal containment. Specifically, as mentioned in [11], while IM ensures that the query plans returned by the system are feasible (i.e., respect all access restrictions), it does not guarantee maximally contained plans. Occam [27] was the first to formally recognize that the maximally contained plan may correspond to an infinite union of conjunctive query plans. It searches in the space of conjunctive query plans of increasing lengths, pruning candidate plans when they are found to be redundant.

The unfolding of recursion inherent in IM and Occam also leads to inefficient query plan execution. Specifically, the unfolded conjunctive query plans found by these algorithms tend to have a significant amount of overlapping structure, and executing them separately leads to significant redundant computation.

Emerac uses Duschka's source inversion algorithm [12] to generate a datalog query plan that is maximally contained with respect to the given query. The query minimization and optimization are done directly on the datalog query plan, without converting it to (a potentially infinite union of) conjunctive query plans. *Emerac* thus avoids the redundant processing involved in executing unfolded conjunctive query plans.

Friedman and Weld [18] offer an efficient algorithm for minimizing a non-recursive query plan through the use of LCW statements. Their algorithm is based on pair-wise subsumption checks on conjunctive rules. Recursive rules correspond to infinite unions of conjunctive queries, and trying to prove subsumption through pair-wise conjunctive rule containment checks will not be decidable. The approach in Duschka [9] also suffers from similar problems as it is based on the idea of conjunctive (un)foldings of a query in terms of source relations [39]. In the case of recursive queries or sources with binding restrictions, the number of such foldings is infinite. In contrast, our minimization algorithm is based on the notion of uniform containment for recursive datalog programs [40]. This approach can check if sets of rules subsume a single rule. Thus it can minimize a much greater range of plans.

Execution optimization in *Emerac* involves ordering the calls to the sources so as to reduce the access and transfer costs. There are some similarities between this source call ordering and the join ordering in traditional databases. In contrast to traditional databases however, we cannot assume access to full statistics in the case of information gathering. For example, the execution ordering algorithm used in the Information Manifold (IM) system [33] generalizes the bound-is-easier style approach (first proposed as part of the Nail! system [35]) to work with Internet sources with multiple capability records (essentially, multiple feasible binding patterns per source), and to reduce tuple transfer costs by pushing selections to the sources. Our work can be seen as further extending the IM algorithm such that it uses coarse information about the result cardinality for each feasible binding pattern, as well as unselectable attribute limitations. Unlike the IM algorithm, we also explicitly consider optimizing both source access cost and tuple transfer cost. In contrast to IM which focuses on the tuple transfer costs, Yerneni and Li [46] focus exclusively on minimizing access cost. The algorithm described in this paper may be seen as striking a middle ground between minimizing source access costs alone or minimizing tuple transfer costs alone. The experiments in Section 7.3 establish the importance of considering both types of costs. Finally, while use of heuristic algorithms for query optimization is one approach for dealing with lack of source statistics, another approach would be to *learn* the statistics. In [25], Gruser *et. al.* describe an approach for online learning of response times for Web-accessible sources.

It should be noted that systems that address integration in the context of federated database systems do use more cost-based approaches. For example, the issue of join ordering in the context of heterogeneous distributed databases is considered in the DISCO [1] and Garlic [22] projects. In contrast to our approach, both these projects assume availability of full statistics for the sources being integrated, and thus concentrate on cost-based optimization methods. For example, the Garlic optimizer assumes full knowledge of the statistics about the databases being integrated as well as their access and query support capabilities. The query processing capabilities are represented as a set of rules which are used by a Starburst-style optimizer [34] to rewrite the mediator query. The statistics are used for cost-based optimization. In the DISCO [1] approach, the optimizer assumes that the wrapper for each source provides a complete cost model for the source. The main difference between our approach and the Garlic approach is in terms of the granularity of knowledge available about the information sources being integrated. While the Garlic and DISCO approaches are well suited for federated database systems, where there is some level of central authority, they are less well suited for integration in the context of information gathering, where the sources are autonomous and decentralized, and are under no obligation to export source statistics. Our approach relies on more coarse-grained statistics and thus can get by without insisting on full knowledge of the source capabilities and statistics. In our current *Havasu* data integration project, we are pursuing a complementary approach–that of *learning* the needed statistics, and then using them as the basis for cost-based query optimization [36, 37, 38].

9. Conclusion

In this paper, we considered the query optimization problem for information gathering plans, and presented two novel techniques. The first technique makes use of LCW statements about information sources to prune unnecessary information sources from a plan. For this purpose, we have modified an existing method for minimizing datalog programs under uniform containment, so that it can minimize *recursive* information gathering plans with the help of source subsumption information. The second technique is a greedy algorithm for ordering source calls that respects source limitations, and takes both access costs and traffic costs into account, without requiring full source statistics. We have then discussed the status of a prototype implementation system based on these ideas called *Emerac*, and presented an evaluation of the effectiveness of the optimization strategies in the context of *Emerac*. We have related our work to other research efforts and argued that our approach is the first to consider end-to-end the issues of redundancy elimination and optimization in recursive information gathering plans.

We are currently exploring the utility of learning rudimentary source models by keeping track of time and solution quality statistics, and the utility of probabilistic characterizations of coverage and overlaps between sources [37, 38]. We are also working towards extending our current greedy plan generation methods, so as to search a larger space of feasible plans and to make the query optimization sensitive to both coverage and cost [36].

References

1. L. Raschid A. Tomasic and P. Valduriez. Scaling access to heterogeneous data sources with disco. *IEEE TKDE*, 10(5), 1998.
2. Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '98*, 1998.
3. Sibel Adalı, K. S. Candan, Yannis Papakonstantinou, and V.S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 137–148, June 1996.
4. Sibel Adalı, Kasim S. Candan, Yannis Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proceedings of the ACM Sigmod International Conference on Management of Data*, pages 137–148, 1996.
5. Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 98*, pages 34–43, 1998.
6. Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseak Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering*, IEEE Comput. Soc. Press, pages 190–200, Los Alamitos, CA, 1995.
7. Surajit Chaudhuri and Kyuseok Shim. Query optimization in the presence of foreign functions. In *Proc. 19th VLDB Conference*, 1993.
8. Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of the 100th Anniversary Meeting*, pages 7–18, Tokyo, Japan, October 1994. Information Processing Society of Japan.
9. Oliver M. Duschka. Query optimization using local completeness. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97*, pages 249–255, Providence, RI, July 1997.
10. Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '97*, pages 109 – 116, Tucson, AZ, May 1997.
11. Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI*, Nagoya, Japan, August 1997.
12. O.M. Duschka. *Query Planning and Optimization in Information Integration*. PhD thesis, Stanford University, 1997.
13. R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Company, Inc., 2nd edition, 1994.
14. Oren Etzioni, Keith Golden, and Daniel Weld. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence*, 89(1–2):113–148, January 1997.

15. Daniela Florescu, Daphne Koller, Alon Y. Levy, and Avi Pfeffer. Using probabilistic information in data integration. In *Proceedings of VLDB-97*, 1997.
16. Daniela Florescu, Alon Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *Proc. SIGMOD Conference*, 1999.
17. Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database techniques for world-wide web: A survey. *SIGMOD Record*, September 1998.
18. Marc Friedman and Daniel S. Weld. Efficiently executing information-gathering plans. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI*, Nagoya, Japan, August 1997.
19. Hector Garcia-Molina, Wilburt Labio, and Ramana Yerneni. Capability sensitive query processing on internet sources. In *Proc. ICDE*, 1999.
20. Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos, and Jennifer Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.
21. Donald F. Geddis, Michael R. Genesereth, Arthur M. Keller, and Narinder P. Singh. Infomaster: A virtual information system. In *Intelligent Information Agents Workshop at CIKM '95*, Baltimore, MD, December 1995.
22. Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *Proc. VLDB*, 1997.
23. Alon Halevy. Answering queries using views: A survey. *VLDB Journal*, 2001.
24. Chun-Nan Hsu. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In *Proceedings of the AAAI Workshop on AI and Information Integration*, pages 66–73, 1998.
25. L. Raschid J. Gruser and V. Zadorozhny. Learning response time for websources using query feedback and application in query optimization. *The VLDB Journal*, 9, 2000.
26. Subbarao Kambhampati and Senthil Gnanaprakasam. Optimizing source-call ordering in information gathering plans. In *Proc. IJCAI-99 Workshop on Intelligent Information Integration*, 1999.
27. Chung T. Kwok and Daniel S. Weld. Planning to gather information. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, 1996.
28. Eric Lambrecht. Optimizing recursive information gathering plans. Master's thesis, Arizona State University, August 1998.
29. Eric Lambrecht and Subbarao Kambhampati. Planning for information gathering: A tutorial survey. Technical Report ASU CSE TR 97-017, Arizona State University, 1997. rakaposhi.eas.asu.edu/ig-tr.ps.
30. Eric Lambrecht and Subbarao Kambhampati. Optimizing information gathering plans. In *Proc. AAAI-98 Workshop on Intelligent Information Integration*, 1998.
31. Eric Lambrecht, Subbarao Kambhampati, and Senthil Gnanaprakasam. Optimizing recursive information gathering plans. In *Proc. IJCAI*, 1999.
32. Alon Y. Levy. Obtaining complete answers from incomplete databases. In *Proceedings of the 22nd International Conference on Very Large Databases*, pages 402–412, Bombay, India, 1996.
33. Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases*, pages 251–262, Bombay, India, 1996.
34. G. Lohman L.M. Haas, J. Freytag and H. Pirahesh. Extensible query processing in starburst. *In Proceedings of SIGMOD*, 1989.
35. K.A. Morris. An algorithm for ordering subgoals in nail! *In Proceedings of PODS*, 1988.

36. Z. Nie and S. Kambhampati. Joint optimization of cost and coverage of query plans in data integration. In *Proc. CIKM*, 2001.
37. Z. Nie, U. Nambiar, S. Vaddi and S. Kambhampati. Mining coverage statistics for websource selection in a mediator. In *Proc. CIKM*, 2002.
38. Z. Nie, S. Kambhampati, U. Nambiar, and S. Vaddi. Mining source coverage statistics for data integration. In *Proc. Web Information and Data Management (WIDM) workshop*, 2001.
39. Xiaolei Qian. Query folding. In *Proceedings of the 12th International Conference on Data Engineering*, pages 48–55, New Orleans, LA, February 1996.
40. Yehoshua Sagiv. *Optimizing Datalog Programs*, chapter 17. M. Kaufmann Publishers, 1988.
41. Anthony Tomasic, Louiqua Raschid, and Patrick Valduriez. A data model and query processing techniques for scaling access to distributed heterogeneous databases in Disco. *IEEE Transactions on Computers, special issue on Distributed Computing Systems*, 1997.
42. Jeffrey D. Ullman. *Principles of Database and Knowledgebase Systems*, volume 2. Computer Science Press, 1989.
43. T. Urhan and M. Franklin. Cost-based query scrambling for initial delays. *In Proceedings of SIGMOD*, 1998.
44. Vasiis Vassalos and Yannis Papakonstantinou. Using knowledge of redundancy for query optimization in mediators. In *Proceedings of the AAAI Workshop on AI and Information Integration*, pages 29–35, 1998.
45. Vasilis Vassalos and Yannis Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In *Proc. VLDB*, 1997.
46. Ramana Yerneni and Chen Li. Optimizing large join queries in mediation systems. In *Proc. International Conference on Database Theory*, 1999.
47. Qiang Zhu and Per-Ake Larson. Developing regression cost models for multidatabase systems. In *In Proceedings of PDIS*, 1996.