Efficiently Executing Information Gathering Plans

Eric Lambrecht and **Subbarao Kambhampati**
{eml, rao}@asu.edu
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287
(602) 965-2735

Abstract

The most costly aspect of gathering information over the Internet is that of transferring data over the network to answer the user's query. We make two contributions in this paper that alleviate this problem. First, we present an algorithm for reducing the number of information sources in an information gathering (IG) plan by reasoning with localized closed world (LCW) statements. In contrast to previous work on this problem, our algorithm can handle recursive information gathering plans that arise commonly in practice.

Second, we present a method for reducing the amount of network traffic generated while executing an information gathering plan by reordering the sequence in which queries are sent to remote information sources. We will explain why a direct application of traditional distributed database methods to this problem does not work, and present a novel and cheap way of adorning source descriptions to assist in ordering the queries.

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* which take the user's query, and develop and execute an effective *information gathering plan* to answer the query. Several first steps have recently been taken towards the development of a theory of such gatherers in both databases and artificial intelligence communities. The information gathering problem is typically modeled by building a global schema for the information that the user is interested in, and describing the information sources as materialized views on the global schema. When a query is posed on the global schema, the information gathering plan for answering it will be a datalog program whose EDB predicates correspond to the materialized view predicates that represent information sources.

Recent research by Duschka and his co-workers (Duschka & Genesereth 97; Duschka & Levy 97) has resulted in a clean methodology for constructing information gathering plans for user queries posed in terms of a global schema. The plans produced by this methodology, while complete – in that they will retrieve all accessible answers to the query, tend to be highly redundant. Naive execution of such plans can be costly in that it can generate prohibitively large amounts of network traffic. In decentralized and geographically distributed scenarios like the World Wide Web, the cost to transfer information and access information sources is particularly high. Consequently, minimization of the amount of network traffic generated to answer the user's query has become a critical problem. In this paper we present two methods for optimizing information-gathering plans for reducing network traffic: one for removing redundant information sources from an information gathering plan, and one for informing the gatherer of more efficient orders to execute an information gathering plan.

A popular approach to minimizing information gathering plans involves the use of localized closed world (LCW) statements (Etzioni *et al.* 97), which state what portion of the data in each information source is complete with regard to the global schema. Intuitively, if the user's query is subsumed by this portion of the data, then no other relevant information source needs to be accessed. Our first contribution in this paper is a systematic methodology for minimizing recursive information gathering plans (such as those generated by (Duschka & Genesereth 97)). Our approach removes redundant materialized view references by using LCW statements, and an algorithm for minimizing datalog programs under uniform equivalence.

Although there exists some previous research on minimizing information gathering plans using LCW statements (Duschka 97; Friedman & Weld 97), 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 (Duschka

& Genesereth 97). 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 gatherer, *Emerac*, attest to the cost-effectiveness of our minimization approach.

Once an information gathering plan is generated, the order in which information sources in the plan are queried can affect the amount of data transferred over the network. By delaying a query to some source until others are completed, the gatherer may be able to generate more specific queries on the source using data collected from other sources. This problem has been tackled in the distributed databases domain, but the assumptions there do not hold in the IG domain. Our second contribution in this paper is an adornment for information source descriptions that provide the gatherer with information it can use to reason about an efficient order to query sources.

The rest of the paper is organized as follows. First we review previous work on building and minimizing information gathering plans. Then we discuss the algorithm for minimizing a datalog program under uniform equivalence that forms the core of our minimization algorithm. We also discuss a modification to the plan generation procedure described in (Duschka & Genesereth 97) to allow our adaptation of the minimization algorithm to work more effectively. Then we present the full algorithm for minimizing an information gathering plan. Next, we discuss the extra information we give to the gatherer that allows it to reason about efficient plan orderings, and we show how the gatherer can use that information. Finally, we relate our work with others, and present our conclusions and directions for future work.

Background

Building Information Gathering Plans

Given a query on some database schema and a set of materialized views defined over the same schema, it has been shown (Duschka & Genesereth 97) that a datalog program in which all EDB predicates are materialized views can be easily constructed to answer the query, if such a program exists. We call such a datalog program an *information gathering plan.* This method has been extended (Duschka & Levy 97) to deal with functional dependency information and information sources with query binding pattern requirements. The heart of the algorithm is a simple materialized view inversion algorithm.

We use the '$\rightarrow$' symbol to denote materialized view definitions. Consider that we have the information source SOURCEA represented as a materialized view as follows:

$$\text{SOURCEA}(X, Y) \rightarrow j(X, Y) \wedge k(Y, Z)$$

This definition states that the information source represented by the SOURCEA predicate contains tuples with two attributes, such that each tuple satisfies the logical sentence in the body of the view. Consider now that we have the query:

$$query(X, Y) \text{ :- } j(X, Y)$$

We can easily build an information gathering plan that will answer the query (at least partially) using only our materialized view above. This method works by *inverting* all materialized view definitions, then 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. All variables that appear in the head of the original rule are unchanged, but every variable in the body of the original view that does not appear in the head is replaced with a new function term $f_N(X_1, \ldots, X_m)$. When we invert our definition above, we achieve

$$\begin{aligned} j(X, Y) &\text{ :- } \text{SOURCEA}(X, Y) \\ k(Y, f_1(X, Y)) &\text{ :- } \text{SOURCEA}(X, Y) \end{aligned}$$

When these rules are added to the original query, they effectively create a logic program to answer the original query. Notice that the global schema predicates that were considered EDB predicates are now IDB, and all EDB predicates are now materialized views.

Function terms in the constructed logic program can easily be removed through a flattening procedure, to convert it into a true datalog program. Note that function terms only appear through inverted rules, which are never recursive, so there will never be an infinite number of function terms in the plan. We can remove function terms through a simple derivative of bottom up evaluation. If bottom up evaluation of the logic program can yield a rule with a function term in an IDB predicate, then a new rule is added with the corresponding function term. Then, function terms that appear in IDB predicates are replaced with the arguments to the function terms, and the name of the IDB predicate is annotated to indicate this replacement. Thus, the rule we created above:

$$k(Y, f_1(X, Y)) \text{ :- } \text{SOURCEA}(X, Y)$$

is replaced with

$$k^{<1, f(2,3)>}(Y, X, Y) \text{ :- } \text{SOURCEA}(X, Y)$$

In this fashion, all function terms in the logic program are removed, and it becomes a true datalog program. This completes our creation of an information gathering plan to answer the original query.

The materialized view inversion algorithm can be modified in order to model databases that cannot answer arbitrary queries, and have binding pattern requirements. Consider that we have a second informa-

to $r1[s \mapsto s \wedge v]$, which is

$$r5: j(X, Y) \;:-\; dom(X) \wedge \text{SOURCEC}(X, Y) \wedge j(X, Y) \wedge k(Y, \text{"Duschka"})$$

It is easy to see that there is indeed a containment mapping between *r4* and *r5*. This proves that *r2* subsumes *r1* and so *r1* can be removed from the rewritten query without affecting the answer.

Plan Minimization Preliminaries

We now consider the process of minimizing information gathering plans with recursion with the help of LCW statements. This process is a modification of a method for minimizing datalog programs under uniform equivalence, presented in (Sagiv 88). In this section, we first introduce the original method of reduction under uniform equivalence, and then introduce a modification to the view inversion algorithm that reduces the number of IDB predicates in the information gathering plan to enable the minimization algorithm to work more effectively on information gathering plans.

Minimizing Datalog Programs Under Uniform Equivalence

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 (Sagiv 88). Checking equivalence is known to be undecideable. Two datalog programs are uniformly equivalent if they produce the same result for all possible assignments of EDB and IDB predicates. Uniform equivalence is decideable, and implies equivalence. (Sagiv 88) offers a method for minimizing a datalog program under uniform equivalence that we present here, and later adapt for our information gathering plan minimization.

The technique for minimizing a datalog program under uniform equivalence involves removing rules from the program, one at a time, and checking to see if the remaining rules can produce the same data as the removed rule, given an initial assignment of relations. If the remaining rules cannot do so, then the removed rule is reinserted into the datalog program. The initial assignment of relations is built by instantiating the variables in the body of the removed rule. To instantiate the variables of the relation $foo(X_1 \ldots X_n)$ means to create an instance of the relation *foo* with constants "X_1" ... "X_n".

Consider that we have the following datalog program:

$$r1: p(X) \;:-\; p(Y) \wedge j(X, Y)$$
$$r2: p(X) \;:-\; s(Y) \wedge j(X, Y)$$
$$r3: s(X) \;:-\; p(X)$$

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 simple bottom-up evaluation. Our initial assignment of relations is:

$$p(\text{"Y"})$$
$$j(\text{"X"}, \text{"Y"})$$

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 *j("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.

Modification of Inversion Algorithm

One artifact of the process of removing function terms from an information gathering plan is the large number of IDB predicates added to the plan. These extra predicates make it difficult to perform pairwise rule subsumption checks on rules in our information gathering plan. Recall that, in our example of function term removal above, we created a new predicate $edge^{<1,f(2,3)>}$ by flattening out an *edge* predicate with function term arguments. This new predicate is incomparable with the *edge* predicate, because they have different names and different arities, even though both refer to the same relation. Because of this mismatch, we try to eliminate the newly introduced predicates before attempting to minimize the information gathering plan. By reducing the number of IDB predicates in the plan to a minimum, pairwise subsumption checks work more effectively.

To further illustrate the problem, consider that we have the following information gathering program, where SOURCEA and SOURCEB are materialized view predicates:

$$query(X) \;:-\; j^{<1,f_1(2,3)>}(X, X, Y)$$
$$query(X) \;:-\; j^{<1,f_2(2,3)>}(X, X, Y)$$
$$j^{<1,f_1(2,3)>}(X, X, Y) \;:-\; \text{SOURCEA}(X, Y)$$
$$j^{<1,f_2(2,3)>}(X, X, Y) \;:-\; \text{SOURCEB}(X, Y)$$

There is no pair of rules for which we can build a containment mapping to prove subsumption, because each rule differs from all others by at least one predicate. However, if we remove the variants of the *j* predicate, we can obtain the equivalent program:

$$query(X) \;:-\; \text{SOURCEA}(X, Y)$$
$$query(X) \;:-\; \text{SOURCEB}(X, Y)$$

Since we know how to compare rules with materialized views using LCW and view statements, we can compare these rules to determine if one subsumes the other.

The IDB predicate removal algorithm works in two parts: first a search is performed to find predicates that can be safely removed without altering the meaning of the program, then the rules with those predicates in their head are removed and unified with the remaining rules. An IDB predicate can be safely removed if it does

tion source, SOURCEC that has a binding constraint on its first argument. We denote this in its view as follows:

$$\text{SOURCEC}(\$X,\ Y) \quad \rightarrow \quad j(X,\ Y) \wedge k(Y,\ Z)$$

The '$' notation denotes that X must be bound to a value for any query sent to SOURCEC. The inversion algorithm can be modified to handle this constraint as follows. When inverting a materialized view, for every argument X_n of the head that must be bound, the body of every rewrite rule produced for this materialized view must include the domain relation $dom(X_n)$. Also, for every argument X_i that is not required to be bound in the head, $\text{SOURCE}(X_1, \ldots, X_m)$, of some materialized view we must create a rule for producing *dom* relations. The head of each domain relation producing rule is $dom(X_i)$, and the body is the conjunction of the information source relation and a $dom(X_n)$ relation for every variable X_n that is required to be bound. Thus, after inverting the materialized view definitions for the SOURCEC view and the SOURCEA view with the modified algorithm, we obtain

$$\begin{array}{rcl}
j(X,\ Y) & :- & \text{SOURCEA}(X,\ Y) \\
k(Y,\ f_1(X,\ Y)) & :- & \text{SOURCEA}(X,\ Y) \\
dom(X) & :- & \text{SOURCEA}(X,\ Y) \\
dom(Y) & :- & \text{SOURCEA}(X,\ Y) \\
j(X,\ Y) & :- & dom(X) \wedge \text{SOURCEC}(X,\ Y) \\
k(Y,\ f_2(X,\ Y)) & :- & dom(X) \wedge \text{SOURCEC}(X,\ Y) \\
dom(Y) & :- & dom(X) \wedge \text{SOURCEC}(X,\ Y)
\end{array}$$

What is interesting to note here is that the plan to solve a non-recursive query with no recursion might contain recursion as a result of the method used to model query constraints on information sources. In fact, if any information sources with binding pattern requirements are available and relevent to the user's query, then the plan that answers the query will contain recursion through the *dom* predicates.

LCW statements & Rule Subsumption

Materialized view definitions alone do not provide enough information for information gathering plan generation to generate efficient plans. Consider an information source, SOURCEB, that subsumes SOURCEA (described above). Its materialized view definition could be

$$\text{SOURCEB}(X,\ Y) \quad \rightarrow \quad j(X,\ Y)$$

Note that even though this view definition appears to subsume the view for SOURCEA, it would be incorrect to conclude from the view definition alone that the information in SOURCEB subsumes that in SOURCEA. This is because the materialized view definition defines a query that all information in the view satisfies, but it doesn't necessarily mean that the view contains *all* the information that satisfies the query. Now consider the new information gathering plan for our original query that would be constructed if SOURCEA and SOURCEB were available. The plan contains redundancies because all rules that reference SOURCEA are unnecessary. That is, we can remove portions of the rewrite (specifically, the rules with SOURCEA in them) and still return the same answer. Without more knowledge of information sources, we cannot discover or eliminate this redundancy.

We can more accurately describe the information sources by using *localized closed world (LCW)* statements[1] (Etzioni *et al.* 97; Friedman & Weld 97). Where a materialized view definition describes all possible information an information source might contain in terms of a global schema, an LCW statement describes what information the source is guaranteed to contain in terms of the global schema. Defining an LCW statement for an information source is similar to defining a view. We use the '$\leftarrow$' symbol to denote an LCW definition. Consider the following LCW definitions:

$$\begin{array}{rcl}
\text{SOURCEA}(X,\ Y) & \leftarrow & j(X,\ Y) \wedge k(Y,\ \textit{"Duschka"}) \\
\text{SOURCEB}(X,\ Y) & \leftarrow & j(X,\ Y)
\end{array}$$

In this example, SOURCEB contains all possible instances of the j relation, and SOURCEA contains a subset of those instances.

Consider that 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. We cannot directly compare the two rules because materialized view predicates are unique and therefore incomparable. However, if we make use of the source's view and LCW statements, we can determine if one rule subsumes the other (Duschka 97; Friedman & Weld 97).

Given some rule, A, let $A[s \mapsto s \wedge v]$ be the result of replacing every information source predicate s_i that occurs in A with the conjunction of s_i and the body of its view. Also let $A[s \mapsto s \vee l]$ be the result of replacing every information source relation s_i that occurs in A with the disjunction of s_i and the body of its LCW statement. Given two rules, A and B, rule A subsumes B if there is a containment mapping from $A[s \mapsto s \vee l]$ to $B[s \mapsto s \wedge v]$ (Duschka 96).

Consider the following inverted rules from our example above:

$$\begin{array}{rcl}
r1:\ j(X,\ Y) & :- & dom(X) \wedge \text{SOURCEC}(X,\ Y) \\
r2:\ j(X,\ Y) & :- & \text{SOURCEB}(X,\ Y)
\end{array}$$

We can prove that rule *r2* subsumes rule *r1* by showing a containment mapping from one of the rules from $r2[s \mapsto s \vee l]$, which is:

$$\begin{array}{rcl}
r3:\ j(X,\ Y) & :- & \text{SOURCEB}(X,\ Y) \\
r4:\ j(X,\ Y) & :- & j(X,\ Y)
\end{array}$$

[1] In (Duschka 96), the notion of a *conservative view* is equivalent to our LCW statements, and the notion of a *liberal view* is equivalent to our normal materialized view.

for each IDB predicate, p_i that occurs in P
    append '-*idb*' to p_i's name
repeat
        let r be a rule of P that has not yet been considered
        let $\hat{P}$ be the program obtained by deleting rule r from P
        let $\hat{P}'$ be $\hat{P}[s \mapsto s \vee l]$
        let r' be $r[s \mapsto s \wedge v]$
        if there is a rule, r_i in r', such that r_i is uniformly subsumed by $\hat{P}'$
            then replace P with $\hat{P}$
until each rule has been considered once

Figure 1: Information gathering plan reduction algorithm for some plan P

not appear as a subgoal of one of its own rules.

Once we have a list of all predicates that can be removed, we can replace references to those predicates in the information gathering program with the bodies of the rules that define the predicates. The algorithm in section 13.4 of (Ullman 89) performs such a task. After passing each predicate through this algorithm, we have successfully reduced the number of IDB predicates in our information gathering program to a minimum.

Minimizing Information Gathering Plans

The basic procedure for reducing an information gathering plan runs as in the datalog minimization under uniform equivalence algorithm (section). We 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 algorithm 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 1.

The process of transforming the candidate rule and the rest of the plan can best be described with an example. Consider the following problem. We have information sources described by the following materialized views and LCW statements:

$$\text{ADVISORDB}(S, A) \rightarrow \textit{advisor}(S, A)$$
$$\text{ADVISORDB}(S, A) \leftarrow \textit{advisor}(S, A)$$
$$\text{CONSTRAINEDDB}(\$S, A) \rightarrow \textit{advisor}(S, A)$$
$$\text{CONSTRAINEDDB}(\$S, A) \leftarrow \textit{advisor}(S, A)$$

and our query is

$$\textit{query}(X, Y) \text{ :- } \textit{advisor}(X, Y)$$

After rule inversion and addition of the query to the inverted rules, our information gathering plan is computed to be:

$$\begin{aligned}
\textit{r1: query}(X, Y) &\text{ :- } \textit{advisor}(X, Y)\\
\textit{r2: advisor}(S, A) &\text{ :- } \text{ADVISORDB}(S, A)\\
\textit{r3: advisor}(S, A) &\text{ :- } \textit{dom}(S)\\
&\quad \wedge \text{CONSTRAINEDDB}(S, A)\\
\textit{r4: dom}(S) &\text{ :- } \text{ADVISORDB}(S, A)\\
\textit{r5: dom}(A) &\text{ :- } \text{ADVISORDB}(S, A)\\
\textit{r6: dom}(A) &\text{ :- } \textit{dom}(S)\\
&\quad \wedge \text{CONSTRAINEDDB}(S, A)
\end{aligned}$$

Most of this plan is redundant. Only the rules *r1* (the query) and *r2* are needed to completely answer the query. The remaining rules are in the plan due to the constrained information source.

For our example, we'll try to prove that rule *r3* 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. If we call our rule r, then we want to transform it to $r[s \mapsto s \wedge v]$. This produces:

$$\begin{aligned}
\textit{advisor}(S, A) \text{ :- } &\textit{dom}(S) \wedge \text{CONSTRAINEDDB}(S, A)\\
&\wedge \textit{advisor}(S, A)
\end{aligned}$$

There is a problem here that must be dealt with before this transformation. The *advisor* relation in the head of the rule no longer represents the same thing as the *advisor* relation in the body of the rule. That is, the *advisor* predicate in the head represents an IDB relation in the information gathering plan, while the *advisor* predicate in the body represents an EDB predicate. Before we replace an information source in some rule with its view or LCW, we need to rename the global schema predicates in the rule so they don't match the predicates from the views. For every world predicate named *predicate* that appears in the rule, we rename it *predicate-idb*. The correct transformation of r, then, is

$$\begin{aligned}
\textit{advisor-idb}(S, A) \text{ :- } &\textit{dom}(S)\\
&\wedge \text{CONSTRAINEDDB}(S, A)\\
&\wedge \textit{advisor}(S, A)
\end{aligned}$$

For the remaining rules, P, we transform them into $P[s \mapsto s \vee l]$ (after renaming the IDB predicates), which represents the information guaranteed to be produced by the information sources in the rules. For our exam-

ple, we produce:

query(X, Y) :- *advisor-idb(X, Y)*
advisor-idb(S, A) :- ADVISORDB*(S, A)*
advisor-idb(S, A) :- *advisor(S, A)*
dom(S) :- ADVISORDB*(S, A)*
dom(S) :- *advisor(S, A)*
dom(A) :- ADVISORDB*(S, A)*
dom(A) :- *advisor(S, A)*
dom(A) :- *dom(S)* ∧ CONSTRAINEDDB*(S, A)*
dom(A) :- *dom(S)* ∧ *advisor(S, A)*

When we instantiate the body of the transformed removed rule, we get the following constants:

dom("S")
constrainedDB("S", "A")
advisor("S", "A")

After evaluating the remaining rules given with these constants, we find that we can derive *advisor-idb("S", "A")*, which means we can safely leave out the rule we've removed from our information gathering program.

If we continue with the algorithm on our example problem, we won't remove any more rules. The remaining *dom* rules can be removed if we do a simple reachability test from the user's query. Since the *dom* rules aren't referenced by any rules reachable from the query, they can be eliminated as well.

The final information gathering plan that we end up with after executing this algorithm will depend on the order in which we remove the rules from the original plan. Consider if we tried to remove *r2* from the original information gathering plan before *r3*. Since both rules will lead to the generation of the same information, the removal would succeed, yet the final information gathering plan would contain the *dom* recursion in it, which greatly increases the execution cost of the plan. A good heuristic for ordering the rules for testing for removal is by information source execution cost, from highest to lowest. That is, we first try to remove rules that involve calling one or more information sources that take a long time to return an answer. Rules which have a *dom* term should have a large extra cost added to them, since recursion often arises due to *dom* rules, and recursion implies high execution cost.

Informing the Gatherer of Query Order Optimizations

Given our final information gathering plan, an information gatherer might evaluate it using traditional database evaluation algorithms like bottom-up or top-down datalog evaluation. Because the gatherer most likely cannot retrieve the complete contents of the information sources due to binding restrictions or the cost of transferring so much data over the network, bottom-up evalution is impractical. Top-down datalog evaluation is much more applicable in the IG domain, as the data transferred between the gatherer and the remote information sources is much more "focused" than bottom-up, and doesn't require the transfer of as much data over the network.

A crucial practical choice we have to make during top-down evaluation is the order in which predicates are evaluated. If we correctly order the queries to multiple sources, we expect to be able to use the results of earlier queries to reduce the size of results in future queries and facilitate their computation. In database literature, this is referred to as the "bound-is-easier" assumption (Ullman 89).

Consider we have the following query, where each predicate represents a remote information source:

SOURCEA*("Lambrecht", X)* ∧ SOURCEB*(X, Y)*

Which source should we query first? In the absence of any additional information, distributed database literature assumes both SOURCEA and SOURCEB are fully relational databases of similar size (Özsu & Valduriez). In this case, we would then query SOURCEA first, because we would expect the answer to this query to be smaller than retrieving the complete contents of SOURCEB. The results of the query on SOURCEA can then be sent to SOURCEB to complete the evaluation. Consider that if we were to query SOURCEB first, we would have to transfer the entire contents of SOURCEB over the network, then the values bound to X would have to be sent to SOURCEA to finally retrieve the answer at what would likely be a higher cost

In the information gathering domain, however, the assumption that information sources are fully relational databases is no longer valid. An information source may now be a wrapped web page, form interfaced databases, 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, 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.

Consider, in our example above, if SOURCEB were a wrapped web page. Regardless of the selections we place on it, the same amount of data will always be transferred over the network. Thus it doesn't matter what selections we have on SOURCEB, because it will always be a constant cost query, and can be queried before all other information sources without increasing the cost of the plan.

Alternatively, assume that SOURCEA*(W, X)* represents a form interfaced student directory database that accepts a student's name, *W*, and returns a series of *X* values that represent email addresses for this student. Since the directory doesn't accept email addresses

(bindings for X) as part of the query, it isn't worthwhile to query SOURCEB first to obtain those bindings.

Given the types of information sources we expect to find in the information gathering domain: wrapped web pages, form interfaced databases, and fully relational databases, there is a simple way to inform the gatherer as to what types of queries on an information source might reduce the data it transfers over the network. When defining for the gatherer a predicate to represent an information source, for every argument of the predicate that, if bound, might reduce the amount of data transferred over the network in response to a query on the information source, we adorn with a '%'.

Assume we have some fully relational database represented by the predicate RELATIONAL-SOURCE. We can adorn it as in the following:

RELATIONAL-SOURCE*(%X, %Y)*

The '%' on the X denotes that if the gatherer can bind X to some values before sending the query to RELATIONAL-SOURCE, we expect the amount of data to be transferred over the network in response to a query to be smaller than if we were to query RELATIONAL-SOURCE without binding X to any values. This also applies to the Y argument. Thus, we would expect the amount of data transferred over the network as a result of the query

RELATIONAL-SOURCE*(X, Y)*

to be larger than the amount of data due to the query

RELATIONAL-SOURCE*("Eric", Y)*

which in turn is expected to be larger than the amount of data due to query

RELATIONAL-SOURCE*("Eric", "Lambrecht")*

We do not adorn our SOURCEB wrapped web page, from the example above, with any '%' signs. This is because binding values to the arguments of SOURCEB before querying it have no effect on the amount of data transferred due to the query. That is, we don't expect the amount of data transferred over the network from the query

sourceB("Amol", Y)

to be smaller than that transferred due to the query

sourceB(X, Y)

Finally, we annote SOURCEA above as

sourceA(%W, X)

because we only expect that binding values to W will reduce the amount of data transferred over the network. Since SOURCEA doesn't accept bindings for X as part of the queries it will answer, any selections on X must be computed locally by the gatherer after querying SOURCEA for tuples with the given W bindings.

```
V := all variables bound by the head;
mark all subgoals "unchosen";
mark all subgoals with no %-adornments as "chosen";
for i := 1 to m do begin
    b := -1;
    for each unchosen subgoal G do begin
        find the bound and %-adorned arguments
        of G, given that V is the set of bound variables;
        if there are c > b bound %-adorned
        arguments of G and with this binding
        pattern G is permissible
        then begin
          b := c;
          H := G;
        end;
        if b ≠ -1 then begin
          mark H "chosen";
          add to V all variables appearing in H;
        end
        else fail
    end
end
```

Figure 2: Modified version of heuristic ordering algorithm in Figure 12.23 of [Ullman 89]

Given our adorned information source descriptions, we can order our access to information sources in a rule according to the number of bound adorned arguments. Predicates should be ordered within a rule so that all predicates with no adorned arguments appear first in any order, followed by the remaining predicates ordered such that the number of bound and adorned arguments in subsequent predicates never decreases. This algorithm is the same as one presented in (Ullman 89), except that we compare the number of bound %-adorned arguments, rather than just the number of bound arguments in each predicate. The algorithm appears in Figure 2 If the algorithm fails, then subgoals are not reordered. If the algorithm succeeds, then the order in which to access subgoals is specified by H.

Related Work

Friedman and Weld (Friedman & Weld 97) offer an efficient algorithm for reducing the size of a non-recursive rewritten query through the use of LCW statements. Their algorithm converts a non-recursive information gathering plan into a directed acyclic graph whose nodes represent relational operators and information sources. It then searches the graph for nodes that represent the relational operator ***union***, and it attempts to minimize the union by performing pairwise subsumption checks (making use of LCW statements) on each subgraph leading from the union (which represent portions of the plan that are unioned together). It is be-

cause of the fact that this algorithm only performs pairwise subsumption checks that it cannot handle recursion, since it does not make sense to compare a rule defined in terms of itself with another rule. The minimization algorithm in this paper, on the other hand, can check if sets of rules subsume a single rule. Thus while the algorithm presented in this paper is more costly to execute, it can minimize a much greater range of plans.

A complementary approach to optimizing information gathering plans is presented in (Ashish *et al.* 97). The authors present a method for executing additional queries on information sources to determine which ones are not necessary for answering the original query. We believe their technique could be integrated well with those presented in this paper.

Conclusions & Future Work

In this paper, we have presented two ways to reduce the amount of network traffic generated in the process of answering a user's query. The first is an algorithm that makes use of LCW statements about information sources to prune unnecessary information sources (and hence excess network traffic) from a plan. The second is a way of informing the information gatherer of the querying capabilities of information sources, so that it can reason about efficient orderings in which to query remote information sources.

We have implemented the reduction algorithm in our prototype information gatherer *Emerac*, written in Java. The most visible improvement after implementing the algorithm has been *Emerac*'s ability to remove unnecessary recursion. Given an information gathering plan that contains a set of information sources with no binding constraints that can completely answer the query, and a set of information sources with binding constraints that can completely answer the query, *Emerac* is able to remove the references to the information sources with binding constraints. This effectively removes all recursion from the information gathering program, and speeds up answering of the plan tremendously. We are currently improving the gatherer to more efficiently execute plans it has generated.

Our continuing research in IG involves further exploration into minimizing the costs involved in executing an information gathering plan. Given that the minimization algorithm can lead to different plans depending on the order in which rules are tested for removal, we would like to discover the best order for choosing rules for removal based upon our knowledge of execution costs. We are also looking into other cheap annotations on the source descriptions that can be exploited by the information gatherer to improve its efficiency and performance.

References

N. Ashish, C. A. Knoblock, A. Levy. Information Gathering Plans With Sensing Actions. *Proceedins of the Fourth European Conference on Planning*, Toulouse, France, 1997.

Y. Arens, C. A. Knoblock, W. Shen. Query Reformulation for Dynamic Information Integration. *Journal of Intelligent Information Systems*, Boston, MA, 1996.

Oliver M. Duschka. Generating Complete Query Plans Given Approximate Descriptions of Content Providers. *Stanford technical report*, 1996.

Oliver M. Duschka. Query Optimization Using Local Completeness. *Proceedings of AAAI*, 1997.

Oliver M. Duschka and Michael R. Genesereth. Answering Recursive Queries Using Views. *Principles of Database Systems*, 1997.

Oliver M. Duschka and Alon Levy. Recursive plans for information gathering. *Proceedings of IJCAI*, 1997.

O. Etzioni, K. Golden and D. Weld. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence*, 89(1-2), 113–148.

O. Etzioni, S. Hanks, T. Jiang, R. M. Karp, O. Madani, and O. Waarts. Efficient Information Gathering on the Internet. *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1996.

Marc Friedman, Daniel S. Weld. Efficiently Executing Information-Gathering Plans. *International Joint Conference on AI (IJCAI)*, 1997.

Chung T. Kwok and Daniel S. Weld. Planning to Gather Information. *University of Washington technical report UW-CSE-96-01-04*, 1996.

Alon Y. Levy. Obtaining Complete Answers from Incomplete Databases. *Proceedings of the 22nd VLDB Conference*, Mumbai (Bombay), India, 1996.

Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. *Proceedings of the 22nd VLDB Conference*, Mumbai (Bombay), India, 1996.

M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

Yehoshua Sagiv. "Optimizing Datalog Programs." *Foundations of Deductive Databases and Logic Programming.* M. Kaufmann Publishers, Los Altos, CA, 1988.

Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volume II.* Computer Science Press, Rockville, MD, 1989.

Jeffrey D. Ullman. Information Integration Using Logical Views. Invited talk at the *International Conference on Database Theory*, Delphi, Greece, 1997.