

Planning for Information Gathering: A Tutorial Survey

Eric Lambrecht & Subbarao Kambhampati
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287

ASU CSE Technical Report 96-017; May, 1997

Abstract

We discuss the problem of information gathering, which is that of integrating multiple heterogeneous information sources so they appear as a single information source that a user can pose queries on. We characterize information gathering as being made up of two subproblems: query planning and plan execution. Query planning is the process of searching through the space of rewrites of the original query to find rewrites containing only queries on available information sources. Query execution is the process of executing those queries and dealing with execution failure. We relate this general view of information gathering with various information gathering systems, including the Information Manifold, SAGE, Occam/Razor, and TSIMMIS.

1 Overview

The information gathering problem is that of integrating a set of autonomous information sources together so they can be queried as if they were a single information source. The goal is to be able to present to the user a single, conceptual information source on which queries can be posed. An information gatherer solves this problem by converting a query on the conceptual information source into queries on individual information sources and then returning the answer back to the user in terms of the conceptual information source.

One obvious impetus for IG is the proliferation of information sources available on the world wide web. There are information sources for movies, people, automobiles, weather, news, etc. The interface to each source is different, and there are no standard annotations as to which information sources contain related information. A great deal of effort is required to manually combine information from multiple information sources. The creation of an information gatherer for this domain would make it easier to retrieve related information from the world wide web and networked information sources.

The aim of this paper is to define what an information gathering problem is, and provide a tutorial and survey of database and AI planning research on solving this problem. In section 2 we clearly define what an information gathering problem is and we describe the

simplifying assumptions made by current information gathering researchers. In section 3, we describe how the problem is formally modelled given our simplifying assumptions. In section 4 we define a general algorithm that generates a plan to solve an information gathering problem, and in section 4.5 we describe how that plan is executed. Section 5 discusses some implemented information gatherers, and section 6 summarizes the paper and points out areas of future research.

2 Problem Formulation

We formulate the general information gathering problem as this: given a world model, a set of information sources, a mapping of the contents of information sources to the world model, and a query on the world model, return information contained in the information sources that answer the query on the world model.

Consider that under this formulation Internet search engines and directories (such as Infoseek or Yahoo) are not full information gatherers (do not solve information gathering problems). These tools merely return *pointers* to *possible* information related to the user's query. An information gatherer must return the actual information that satisfies the user's query and it should not return information not currently contained within the set of information sources.

Current information gathering research makes some simplifying assumptions about the information gathering problem that are not expressed in the above formulation. The first is that information sources can all be modelled as relational databases. This leads to the formulation of the world model as a global relational schema and (as will be described later) the mappings of the contents of the information sources to the world model as *views* of the world model. Information sources are assumed to be able to answer simple relational queries that possibly require binding patterns to be satisfied.

We also make the assumption that an ideal information gatherer should generate all possible information contained in the information sources that answer the query using the fewest number of information sources. The reasoning behind this assumption is that users want all available answers to their query, and that the cost of information gathering is dominated by the number of information sources accessed to answer a query. An alternative to this formulation might be to assume a different cost for querying each information source, and the goal would be to return as much information as can be gathered within a certain cost, or return the most amount of information for the cheapest cost.

3 Formalization of the Information Gathering Problem

In this section we describe how we formally model information sources, the world model, queries on the world model, valid plans for answering a world query, and the mapping of information sources to the world model.

3.1 Modelling Queries and Information Sources

We model the information stored in information sources as simple unique relations. All tuples for some source are made up of N typed elements. The information stored in some source is then represented by a uniquely named relation with N arguments, each argument represented by a variable. If all variables are replaced with constant values, the ground relation is defined to be true if a tuple with exactly those constant values exists in the database. Consider a white pages information source that stores tuples with name, area code, phone number, and phone company attributes. We can describe it with the relation $\text{WHITEPAGES}(\text{Name}, \text{Area}, \text{Phone}, \text{Co})$ (all information source relations will be in SMALL CAPS type).

Information source queries are defined as *views*. The notion of a view comes from database literature, and looks like the following:

$$\text{query}(\text{Phone}, \text{Company}) \quad :- \quad \text{WHITEPAGES}(\text{Name}, \text{Area}, \text{Phone}, \text{Company}) \wedge \\ \text{Name} = \text{"Eric"}$$

The portion to the right of the “:-” is called the *body* of the view and is a logical conjunctive sentence consisting only of information source relations and constraints on their arguments. The portion to the left of the “:-” is called the *head* and defines the size of the tuples returned by the query. For every true instance of the conjunctive sentence in the body, the same variable bindings are applied to the head to define a tuple that satisfies the query. Our example query returns tuples with two attributes, the first of which is Eric’s phone number, and the second is the phone company that services that number. In general we only allow conjunctive sentences with positive terms in the body of the query.

In the same way that we describe the contents of our information sources using relations, we can build a *world model* by creating relations that specify all the information we think we would like to query. These are the relations that the user will pose queries on. The world model relations can be thought of as representing information contained in a virtual database of all possible information. For example, we might have the world model relations $\text{name}(\text{Name})$, $\text{phone-of}(\text{Name}, \text{Area}, \text{Phone})$, $\text{phone}(\text{Area}, \text{Phone})$, and $\text{phone-company}(\text{Name}, \text{Co})$ which represent all possible names, phone numbers, and phone companies.

We can also express world model queries as views, except that the body of the view may contain only world model relations. For example the world model query:

$$\text{query}(\text{Phone}, \text{Company}) \quad :- \quad \text{phone-of}(\text{Name}, \text{Area}, \text{Phone}) \wedge \\ \text{phone-company}(\text{Name}, \text{Company}) \wedge \\ \text{Name} = \text{"Eric"}$$

asks for Eric’s phone numbers, and the companies that service those numbers.

The task of an information gatherer, given our formalism, is to convert a world model query into information source queries that generate information that would satisfy the world model query and then execute those queries. We call a query with only information source relations in it a *plan* because it describes what information sources need to be accessed and [implicitly] in what order they need to be accessed.

3.2 Relating Information Sources to the World Model

We can relate our information sources to the world model by creating views that define the information source relations in terms of queries on the world model. We will put an information source's relation in the head, and in the body we will put the query over the world model that would produce equivalent information as contained in the source. For example:

$$\begin{aligned} \text{WHITEPAGES}(Name, Area, Phone, Co) \Leftrightarrow & \text{ name}(Name) \wedge \\ & \text{ phone-of}(Name, Area, Phone) \wedge \\ & \text{ phone}(Area, Phone) \wedge \\ & \text{ phone-company}(Name, Co) \end{aligned}$$

In doing this, we are defining all the tuples that satisfy the world model relations in the body as being exactly those that exist in the information source named in the head.

It may be the case that our information sources do not map up exactly to a query in the world model. For instance, we might have a phone book information source which does not contain all name and phone number pairs that would be available in the world model, and we cannot define a specific enough (and reasonably small enough) query over the world model that exactly defines all the tuples contained in the source. In this case we will define a view where the body is over-general and describes more information than is actually contained in the information source (just so we do not miss anything during information gathering). We will write the view definition with a “:-” rather than a “ \Leftrightarrow ” to denote that while all the information described in the head of the rule is contained in the information described in the body, it is not the case the all the information described in the body is contained in the head. Our new definition for our white pages database will be:

$$\begin{aligned} \text{WHITEPAGES}(Name, Area, Phone, Co) \text{ :- } & \text{ name}(Name) \wedge \\ & \text{ phone-of}(Name, Area, Phone) \wedge \\ & \text{ phone}(Area, Phone) \wedge \\ & \text{ phone-company}(Name, Co) \end{aligned}$$

We are also now going to refer to the white pages database as being *incomplete*. An information source is incomplete if the information contained in it is less than the information that would be generated by executing the world query in its view on the conceptual database of all information that the world model describes. Conversely, if execution of the world query in its view would generate exactly the same information that is in the information source, we say the information source is *complete*. Our notion of completeness, then, is defined as how accurately the information contained in some information source can be described in terms of the world model.

This notion of completeness is important, because the information gatherer should retrieve as much information described in the world query as possible, while at the same time minimizing the amount of redundant information gathered. If we do not state that some information sources are incomplete, the information gatherer might incorrectly reason that the information it has retrieved from some information sources completely answers a world query, when it actually does not. Also, the information gatherer might needlessly retrieve

information from two sources where both of them contain exactly the same information.

3.3 Modelling Constrained Sources

Often a real-world information source cannot answer all arbitrary queries over the information it contains. In this case, we say that the information source has *query constraints* and is not, in database terminology, *fully relational*. One important class of restrictions are *binding restrictions*, where an information source requires that some attributes be bound in any query sent to it. Our white pages information source, for example, might require that the *Name* attribute in any query posed to it be bound. So the following query is valid:

$$\text{WHITEPAGES}(\text{Name}, \text{Area}, \text{Phone}, \text{Co}) \wedge \text{Name} = \text{“Eric Lambrecht”}$$

but the next query is not, because the *Name* attribute is not bound to any value:

$$\text{WHITEPAGES}(\text{Name}, \text{Area}, \text{Phone}, \text{Co}) \wedge \text{Co} = \text{“USWest”}$$

By convention, we express these binding restrictions in the description of the information source in terms of the world view by putting a “\$” character in front of the attributes that must be bound for a query. For example, we can express that the whitepages information source has the binding restriction on its *Name* attribute by defining its world view as:

$$\begin{aligned} \text{WHITEPAGES}(\$ \text{Name}, \text{Area}, \text{Phone}, \text{Co}) \quad :- \quad & \text{name}(\text{Name}) \wedge \\ & \text{phone-of}(\text{Name}, \text{Area}, \text{Phone}) \wedge \\ & \text{phone}(\text{Area}, \text{Phone}) \wedge \\ & \text{phone-company}(\text{Co}) \end{aligned}$$

3.4 Desiderata for Information Gathering Plans

Having discussed the way information sources and queries are modeled, we are now ready to define information gathering plans, and their desirable characteristics. Formally, an information gathering plan for a given query is a conjunction of information source relations. For example, given the query for generating the list of telephone numbers of all ASU CS students:

$$\begin{aligned} \text{query}(\text{Phone}, \text{Company}) \quad :- \quad & \text{phone-of}(\text{Name}, \text{Area}, \text{Phone}) \wedge \\ & \text{phone-company}(\text{Name}, \text{Company}) \wedge \\ & \text{Student}(\text{Name}, \text{CSDEPT}) \end{aligned}$$

a plan for satisfying this query might be:

$$\text{plan1}(\text{Phone}, \text{Company}) \quad :- \text{CSSTUDENTS}(\text{Name}) \wedge \text{WHITEPAGES}(\text{Name}, \text{Company}) \wedge$$

(where CSSTUDENTS is presumably a database that lists all the students in the ASU CS department).

This plan can be given both declarative and operational semantics. Declaratively, it is

just a relation defined by the conjunction of two other information source relations. The operational semantics of the plan involve source calls and standard database operations on the returned tuples, such as joins, selections, unions etc. Our plan above involves first calling the CSSTUDENTS source, and then calling WHITEPAGES source for each of the names returned by the call to CSSTUDENTS.

One desirable characteristic of a plan is that of soundness. An information gathering plan is said to be *sound* if each of the tuples returned by the execution of this plan actually satisfy the query. Soundness of a plan is strongly related to the soundness of the information sources.

Another characteristic is completeness. It is useful to talk about two different notions of completeness. An information gathering plan is said to be *operationally complete* if it returns all the tuples that satisfy the user's query that can be extracted from the available information sources. We can also define operational completeness in terms of plan subsumption. An information gathering plan P_1 for a query Q is said to subsume another plan P_2 if P_1 returns all the information that is returned by P_2 . A plan P is said to be operationally complete for a query Q if it subsumes every other possible plan that generates information that satisfies the user's query.

The second notion of completeness is conceptual completeness. An information gathering plan is said to be *conceptually complete* if the information generated by it is exactly the information that would have been generated if the user's query were executed on the conceptual database.

Note that conceptual completeness implies operational completeness, but not vice versa. A planner should be judged in terms of its ability to produce operationally complete plans. Conceptual completeness may be impossible to achieve given the accessible sources and the restrictions on them. In our example, if some of the students in the CS department have phones that are "unlisted" in the WHITEPAGES source, then we cannot blame the planner for not returning telephone numbers of all the students. The situation would be different if the planner had access to the university records of the students, which presumably list everyone's phone number. Similarly, if the query was for everybody's telephone number, the binding restrictions on the WHITEPAGES source implies that there is no practical way to satisfy the query. This is because we cannot reason about when we have "the names of all people with phone numbers", we can only reason if we have "the names of all people".

Finally, an information gathering plan is said to be optimal if it has the lowest cost of execution. In general, execution cost may depend upon a variety of factors, including the cost of accessing different sources. A common cost model used in the literature assumes that all source calls are equally costly. In this model, the cost of a plan is proportional to the number of information sources accessed, and a necessary condition for optimality is *plan minimality*. An information gathering plan is said to be *minimal* if it is not subsumed by any subplan derived from it, by dropping some source calls.

3.5 Annotating source descriptions to support reasoning about plan completeness

In the previous section, we saw that the desiderata for information gathering plans are that they be sound, operationally complete, and optimal. Operational completeness alone is easy to ensure: just call *all* sources that are relevant to the query. However, this method often leads to very inoptimal plans, as different sources may contain the same or highly overlapping information. It is hard to directly reason about the operational completeness of plans since it requires reasoning about the information that might be generated by all possible plans. There are at least two ways of reasoning about it indirectly. First, we can reason about conceptual completeness of a plan (since it implies operational completeness), and stop as soon as we find a minimal plan that is conceptually complete. Second, we can reason about the plan subsumption, and discard (or avoid generating) longer plans that generate information that is subsumed by shorter ones.

To support the first type of reasoning, we need to characterize the completeness of the information sources with respect to the conceptual database. Supporting the second type of reasoning requires characterizing the relative information content of different information sources.

Although we touched upon completeness of a source with respect to conceptual database in Section 3.2, our modeling there was restricted to differentiating complete and incomplete sources. Using only what we've described so far, there is no way to differentiate two incomplete information sources that have the same view. Consider two white pages databases for Phoenix, where one has all USWest customers and some AT&T customers, while the other has all AT&T customers and some USWest customers. Both databases would have the same view:

$$\begin{aligned}
 \text{USWEST}(Name, Area, Phone, Co) & :- \text{ name}(Name) \wedge \\
 & \quad \text{ phone-of}(Name, Area, Phone) \wedge \\
 & \quad \text{ phone}(Area, Phone) \wedge Area = 602 \\
 & \quad \text{ phone-company}(Name, Co) \\
 \text{ATT}(Name, Area, Phone, Co) & :- \text{ name}(Name) \wedge \\
 & \quad \text{ phone-of}(Name, Area, Phone) \wedge \\
 & \quad \text{ phone}(Area, Phone) \wedge Area = 602 \\
 & \quad \text{ phone-company}(Name, Co)
 \end{aligned}$$

A necessary improvement to our use of views is to describe for each incomplete source the subset of its contents that can be completely described in terms of a world model query. These descriptions are called *localized closed world (LCW)* statements [?].

Recall that if we could not create a query to exactly describe the contents of some information source, we had to describe the information source using a world query that at least contained all the tuples that are in the information source. This ensured that we never ignore some information contained in the source. An LCW statement is a description of the information source in terms of a world query for which all the tuples in the world query are guaranteed to be contained in the information source. A sample LCW statement for the information source above that has all USWest customers might look like this:

$$\begin{aligned}
& name(Name) \wedge \\
& phone-of(Name, Area, Phone) \wedge \\
& phone(Area, Phone) \wedge Area = "602" \\
& phone-company(Name, Co) \wedge \\
& Co = "USWest" \quad :- \quad USWEST(Name, Area, Phone, Co)
\end{aligned}$$

This LCW statement says that this database contains the phone numbers of all people in Phoenix served by USWest. The semantics of the statement are the same as that of the view statement that we used to describe our information sources: all the tuples described by the query in the head of the rule are contained in all the tuples described by the query in the body of the rule.

Notice that a complete information source view such as:

$$NAMELISTING(X) \Leftrightarrow name(X)$$

is equivalent to an incomplete view plus an LCW statement:

$$\begin{aligned}
NAMELISTING(X) & \quad :- \quad name(X) \\
name(X) & \quad :- \quad NAMELISTING(X)
\end{aligned}$$

Thus, defining a view with " \Leftrightarrow " is really only a shorthand for defining it with ":-" plus an LCW statement.

LCW statements are necessary in order to more accurately reason about how much of the user's query some set of information sources can completely answer. Consider that we formulate two plans that generate information that satisfies the user's query, and each plan contains incomplete information sources. The LCW statements of those sources allow the information gatherer to reason if one plan guarantees to generate more information than the other plan or if one plan can completely answer the user's query. Without LCW statements we would be forced to execute both plans, possibly at a large cost, since we lack greater knowledge of the contents of the incomplete sources.

It is also possible to accurately describe a portion of the content of some information source in terms of other sources. This knowledge, too, can be used by an information gatherer to reason about the operational and conceptual completeness of information gathering plans. For instance, we might have another white pages database, called `US-PAGES`(*Name, Area, Phone*), that we know contains all the information in our previous white pages database. We can express this with the following LCW statement:

$$WHITEPAGES(Name, Area, Phone, Co) \quad :- \quad US-PAGES(Name, Area, Phone, Co)$$

An LCW statement of this type is called a pairwise LCW statement - and can be generalized to note if any group of information sources subsume any other group.

4 Building and Executing Plans to Answer World Queries

Now that we have defined the world model and the information sources in terms of it, answering a world query is a two part problem: transforming a query on the world model into one or more plans for accessing the individual information sources, and executing those

plans to return the information to the user. The transformation of the query into one or more plans is called *query planning*, and the execution of the plans it produces is called *plan execution*.

As discussed earlier, a conjunctive query only with information source relations in it is equivalent to a plan. Thus searching for a valid query plan is equivalent to searching for a reformulation of the original query that contains only information source relations in it. Searching for valid query reformulations proceeds through the space of reformulations produced by reformulation rules, which can be built from information source views before any queries are answered by the information gatherer. The rewrite rules allow us to construct all possible rewrites of the original query that, when executed, will generate information contained in the query. As we progress through our search, we must make use of LCW statements to reason if rewrites we have found will generate data that subsumes what might be produced by partial rewrites still in the search space, or if some plan we have found is conceptually complete.

4.1 Building Reformulation Rules

The reformulation rules used to transform a world model query to information source queries are built from the *inverse* of information source views. The inverse, v^{-1} , of the view with head $v(X_1, \dots, X_m)$ and no binding restrictions is a set of 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 $f(X_1, \dots, X_m)$. For example, given the following movie information source

MOVIE-MANIA(*Movie*) :- *movie*(*Movie*) \wedge *reviewed-by*(*Critic*, *Movie*)

we can convert it into the following reformulation rules:

movie(*Movie*) :- MOVIE-MANIA(*Movie*)
reviewed-by(*f*(*Movie*), *Movie*) :- MOVIE-MANIA(*Movie*)

The semantics of these rules are that relations that match the head of some rewrite rule can be produced by the information source in the body of the rewrite rule. Thus, if we replace some relation in the original query with the body of a rule that has that relation as its head, we are replacing the specification of what we want (the world model relation) with what must be accessed to get it (the information source relation). Consider we have the query:

query(*X*) :- *movie*(*X*)

it can be transformed (by applying the rules above) into

query(*X*) :- MOVIE-MANIA(*X*)

which is a plan that the information gatherer can execute (by querying the MOVIE-MANIA

information source) to generate an answer to the original query.

Views with binding restrictions are inverted slightly differently. In this case, each relation that appears in the body of a view with binding constraints may generate multiple rules. Assume we are working on the relation, p , that appears in the body of some view with head v . For every combination of the relations in the body of the view such that all bound variables appear as arguments to the relations and each relation has at least one variable not appearing in any other relation in the combination, we will generate a new rule for p . Each rule will have p as the head, with any argument of p that does not appear in the combination replaced with a new function $f(X_1, \dots, X_m)$, and the body will be a conjunctive sentence with the relations in the combination and v .

For example, consider again our constraints white pages information source:

$$\begin{aligned} \text{WHITEPAGES}(\$Name, Area, Phone, Co) \quad &:- \quad name(Name) \wedge \\ & \quad phone\text{-of}(Name, Area, Phone) \wedge \\ & \quad phone(Area, Phone) \wedge \\ & \quad phone\text{-company}(Co) \end{aligned}$$

we can convert it into the following rules:

$$\begin{aligned} name(Name) \quad &:- \quad name(Name) \wedge \\ & \quad \text{WHITEPAGES}(Name, Area, Phone, Co) \\ name(Name) \quad &:- \quad phone\text{-of}(Name, Area, Phone) \wedge \\ & \quad \text{WHITEPAGES}(Name, Area, Phone, Co) \\ phone\text{-of}(Name, Area, Phone) \quad &:- \quad name(Name) \wedge \\ & \quad \text{WHITEPAGES}(Name, Area, Phone, Co) \\ phone\text{-of}(Name, Area, Phone) \quad &:- \quad phone\text{-of}(Name, Area, Phone) \wedge \\ & \quad \text{WHITEPAGES}(Name, Area, Phone, Co) \\ phone(Area, Phone) \quad &:- \quad name(Name) \wedge \\ & \quad \text{WHITEPAGES}(Name, Area, Phone, Co) \\ phone(Area, Phone) \quad &:- \quad phone\text{-of}(Name, Area, Phone) \wedge \\ & \quad \text{WHITEPAGES}(Name, Area, Phone, Co) \\ phone\text{-company}(Co) \quad &:- \quad name(Name) \wedge \\ & \quad \text{WHITEPAGES}(Name, Area, Phone, Co) \\ phone\text{-company}(Co) \quad &:- \quad phone\text{-of}(Name, Area, Phone) \wedge \\ & \quad \text{WHITEPAGES}(Name, Area, Phone, Co) \end{aligned}$$

Note that our inversion process is not different for complete and incomplete views. The reformulation rules are useful in that they point out which information sources might provide tuples that satisfy which world model relations, but they make no statement as to how many tuples each information source can generate. This means that any rewrite of the original query built from these reformulation rules may be incomplete, in that it does not generate all the information described by the original query.

4.2 Performing Query Planning

Given our reformulation rules and a user query, we can perform a breadth first search over the space of all rewrites of the user's query to find the rewrites that contain only information source relations.

The search is initialized by placing the body of the user's query in a search queue. The search progresses by repeatedly pulling a query from the search queue, applying the search reformulation rules to all world model relations in the query to generate child queries, then inserting each child query into the queue to continue reformulation. If a query pulled out of the queue has no world relations, then a valid rewrite of the user's query has been found.

Depending on the constraints on available information sources, there may be zero to an infinite number of conjunctive plans extracted from the queue. First consider if all available information sources have no binding restrictions. In this case each reformulation rule we have generated will only have a single information source relation in its body. Therefore, every world model relation in the user's query will be replaced with a single information source relation. Given a query with N world model relations in the body, any reformulations will contain no more than N information source relations as well. The number of reformulations will be bounded by N^M where M is the number of available information sources. However, if the size of the original query is not large and the number of information sources relevant to this query is small, the search space is easily exhausted.

If information sources have binding restrictions, then the search space is much larger. Consider that when we invert information sources with binding restrictions, one or more world model relations can appear in the body of the rewrite rules. It's possible that a relation that appears as the head of a rule is also in its body, which allows for an infinite number of applications of that rewrite rule. Thus with binding restrictions, the search space is unbounded and there are an infinite number of plans we might find.

We can also illustrate the unboundedness of plans where binding restrictions are involved with an example. Consider the we have the following information sources:

$$\begin{aligned} \text{CS-STUDENTS}(X) & \text{ :- } cs\text{-student}(X) \wedge engineering\text{-student}(X) \\ \text{ENGINEERING-FRIENDS}(\$X, Y) & \text{ :- } engineering\text{-student}(X) \wedge engineering\text{-student}(Y) \\ \text{CSE471-STUDENTS}(\$X) & \text{ :- } engineering\text{-student}(X) \wedge cse471\text{-student}(X) \end{aligned}$$

The first source lists computer science students (who are also engineering students). The second source will give us a list of engineering friends for some engineering student. The final will tell us if some engineering student is in the CSE 471 class. Consider now that we want to answer the query:

$$student\text{-query}(X) \text{ :- } engineering\text{-student}(X) \wedge cse471\text{-student}(X)$$

This query requests all the engineering students who are in CSE 471. Here are three plans that can each return unique answers:

$$\begin{aligned}
student\text{-}query(X) & :- CS\text{-}STUDENTS(X) \wedge CSE471\text{-}STUDENTS(X) \\
student\text{-}query(X) & :- CS\text{-}STUDENTS(Y) \wedge ENGINEERING\text{-}FRIENDS(Y, X) \wedge \\
& CSE471\text{-}STUDENTS(X) \\
student\text{-}query(X) & :- CS\text{-}STUDENTS(Z) \wedge ENGINEERING\text{-}FRIENDS(Z, Y) \wedge \\
& ENGINEERING\text{-}FRIENDS(Y, X) \wedge CSE471\text{-}STUDENTS(X)
\end{aligned}$$

Clearly there can even be more plans if we continue to make use of the engineering-friends database to find more engineering students.

4.3 Pruning Search Using Local Completeness Statements

It is not the case that all rewrites of the original query generate all possible information that satisfies the user's query over the conceptual database of all information, so search cannot stop as soon as any rewrite is found. It is also not the case that every rewrite generates new information not generated by other rewrites. Therefore, in order to be efficient, the information gatherer must reason about the operational and conceptual completeness of plans. This information is helpful in that the information gatherer might reason if a valid rewrite subsumes another partial rewrite still in the search queue.

In order to reason about what can be generated by each plan, we need to build descriptions of the information generated by each plan. A world model query that describes all possible information generated by some plan can be created by simply replacing each information source relation in the plan with its view. This world model query is called the *expansion* of the plan. Likewise, we can replace every information source relation in the plan with the head of its LCW statement to generate an LCW statement for the plan. For example, consider we have the following information sources and LCW statements:

$$\begin{aligned}
SOURCEA(X, Y) & \Leftrightarrow a(X, Y) \\
SOURCEB(X, Y) & :- c(X, Y) \wedge d(Y) \\
c(X, Y) \wedge d(Y) \wedge e(Y) & :- SOURCEB(X, Y)
\end{aligned}$$

and we have generated the plan:

$$plan(X, Y, Z) :- SOURCEA(X, Y) \wedge SOURCEB(Y, Z)$$

Then the expansion and LCW statement for our plan will be, respectively:

$$\begin{aligned}
plan(X, Y, Z) & :- a(X, Y) \wedge c(Y, Z) \wedge d(Z) \\
a(X, Y) \wedge c(Y, Z) \wedge d(Z) \wedge e(Z) & :- plan(X, Y, Z)
\end{aligned}$$

Using this information, we can compare the amount of information generated by some plan with the user's query and other plans using a technique called *containment mapping*.

A plan is equivalent to the user's query if there is a containment mapping from the head of the plan's LCW statement to the body of the user's query and from the body of the user's query to the head of the plan's LCW statement. A containment mapping from query Q_1 to query Q_2 is a mapping of the symbols of Q_1 to the symbols in Q_2 , for each symbol in Q_1 . A containment mapping from Q_1 to Q_2 expresses the fact that all the information produced by Q_2 is a subset of the information produced by Q_1 (i.e. $Q_2 \subseteq Q_1$).

We can illustrate the notion of containment mapping with an example. Consider we have the following user query and the LCW statement for some plan:

$$\begin{aligned} \text{query}(X, Y) & \text{ :- } e(Y, X) \wedge e(X, Z) \\ e(Y, X) \wedge e(Z, X) \wedge e(X, Z_1) & \text{ :- } \text{plan}(X, Y) \end{aligned}$$

We can express the containment mapping from the body of the query to the head of the LCW statement as the function $h(x)$, where $h(X) = X$, $h(Y) = h(Z) = Y$, and $h(Z_1) = Z$. If we apply this mapping, we can transform the plan's LCW statement into:

$$e(Y, X) \wedge e(Y, X) \wedge e(X, Z) \text{ :- } \text{plan}(X, Y)$$

This makes it clear that the user's query \subseteq the information we know will be generated by the plan. We can also build a mapping function, $g(x)$, from the user's query to the LCW statement. Let $g(X) = X$, $g(Y) = Y$, and $g(Z) = Z_1$. This expresses the fact that the information we know will be generated by the plan \subseteq the user's query. Since we've done containment mappings in both directions, we are sure that the information we know will be generated by the plan is equivalent to the user's query.

4.4 Discovering Plan Subsumption

If the query planner finds a plan that can be proven (via containment mappings) to be conceptually complete, then the search for additional plans can halt since no other plan will generate more information. We expect, however, that in the real world this is typically not the case. A more useful real world procedure is to compare if the information produced by one plan will subsume the information produced by another plan. Even more useful, with respect to our search procedure for rewrites, is to determine if a plan we have extracted guarantees to subsume any partial rewrite still in our search queue.

Checking to see if the information produced by plan A subsumes the information produced by plan B is intuitively merely a process of finding a containment mapping from the LCW statement for A to the expansion of B. This exact procedure cannot be used, for reasons pointed out in [Duschka 97], but a slight variant of it can. Given some plan A, let $A[s \mapsto s \wedge v]$ be the result of replacing every information source relation s_i in the body of 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 in the body of A with the disjunction of s_i and the head of its LCW statement. Given two plans, A and B, then the information produced by A subsumes that of B if there is a containment mapping from $A[s \mapsto s \vee l]$ to $B[s \mapsto s \wedge v]$.

To illustrate, consider that we have the following information sources and their corresponding LCW statements:

$$\begin{aligned} \text{SOURCE1}(X) & :- w1(X) \\ w1(X) \wedge w2(X, Y) & :- \text{SOURCE1}(X) \end{aligned}$$

$$\begin{aligned} \text{SOURCE2}(X) & :- w1(X) \wedge w2(X, Y) \\ w1(X) \wedge w2(X, a) & :- \text{SOURCE2}(X) \end{aligned}$$

$$\begin{aligned} \text{SOURCE3}(X) & :- w3(X, Y) \\ w3(X, a) & :- \text{SOURCE3}(X) \end{aligned}$$

Given the user query

$$\text{query}(X) \quad :- \quad w1(X) \wedge w3(X, Y)$$

Assume the query planner found the following two reformulations

$$\begin{aligned} \text{planA}(X) & :- \text{SOURCE1}(X) \wedge \text{SOURCE3}(X) \\ \text{planB}(X) & :- \text{SOURCE2}(X) \wedge \text{SOURCE3}(X) \end{aligned}$$

The planner can check that planA subsumes planB by first computing the value of $\text{planA}[s \mapsto s \vee l]$, which is $\text{planA-1}(X) \cup \text{planA-2}(X) \cup \text{planA-3}(X) \cup \text{planA-4}(X)$ where

$$\begin{aligned} \text{planA-1}(X) & :- \text{SOURCE1}(X) \wedge \text{SOURCE3}(X) \\ \text{planA-2}(X) & :- w1(X) \wedge w2(X, Y) \wedge \text{SOURCE3}(X) \\ \text{planA-3}(X) & :- \text{SOURCE1}(X) \wedge w3(X, a) \\ \text{planA-4}(X) & :- w1(X) \wedge w2(X, Y) \wedge w3(X, a) \end{aligned}$$

and then computing the value of $\text{planB}[s \mapsto s \wedge v]$ to be

$$\begin{aligned} \text{planB}(X) & :- \text{SOURCE2}(X) \wedge w1(X) \wedge w2(X, Y) \wedge \\ & \quad \text{SOURCE3}(X) \wedge w3(X, Z) \end{aligned}$$

Now if a containment mapping can be found from $\text{planA}[s \mapsto s \vee l]$ to $\text{planB}[s \mapsto s \wedge v]$, then planA subsumes planB. There is indeed a mapping $h(x)$ from $\text{planA-2}(X)$ to $\text{planB}(X)$, where $h(X) = X$ and $h(Y) = Y$, so planB can be thrown away since it generates duplicate information.

We can use the same technique to compare a valid rewrite with rewrites that still contain world model relations in them (ie. rewrites that are still in our search queue). Consider that our planner has discovered planA from the previous example, but planB is still in the following form in the search queue:

$$\text{planB}(X) \quad :- \quad \text{SOURCE2}(X) \wedge w3(X, Y)$$

There is a containment mapping, $h(x)$, from $\text{planA}[s \mapsto s \vee l]$ to $\text{planB}[s \mapsto s \wedge v]$, where $h(X) = X$ and $h(Y) = Y$, so we can remove planB from the queue. Consider if we had left planB in the queue. The number of relations in the expression $\text{planB}[s \mapsto s \wedge v]$ would only increase. Since there are already enough symbols in the expression to build our containment map, it must be that case that there will always be enough symbols to build our containment map. Therefore, we can prune planB from the search space since we are assured any valid plan produced from it would be pruned anyway.

4.5 Query Plan Execution

Query execution is the process of executing plans generated by a query planner. The primary issues that must be dealt with in query execution are resource constraints and plan failure. Dealing with resource constraints involves ranking query plans in order to maximize information gain and minimize querying cost (through parallel queries and non-redundant queries), and plan failure involves dealing with a query plan that cannot be completed for some reason.

In the information gathering domain, the primary resource constraint is the cost of accessing a particular information source. The cost may involve the time needed to answer the query and return the results over a network, or it might be actual cost, where the owner of the information gatherer must pay money for an answer to the query. Several variants of this problem are explored in [Etzioni *et al.* 96].

In the case that a plan fails to execute because one or more information sources don't provide an answer, the executor might mark the information source as no longer functioning, and ask for a new plan from the query planner. This time, the query planner will not be able to insert the malfunctioning information source in any plans. When the executor asks for a new plan from the query planner, it might ask again for the original query, or it could ask for the portion of the original query that failed. Alternatively, the query planner might return plans with disjunction in them, that explicitly point out where two information sources might provide the same information in the case of a failure or timeout.

5 Implemented Systems

There are four systems that fit nicely into the framework of information gathering systems: Occam/Razor, SAGE, IM (Information Manifold) and Infomaster. All of these information gatherers make the same assumptions as in this paper about information sources. The first two are (at least originally) influenced by the work in AI planning, while the last two are influenced by the work on database query processing. Two other systems, XII and TSIMMIS, are listed because they share some of the goals of information gathering, although they are aimed at different problems.

5.1 Occam and Razor

The authors of the Occam planner ([Kwok & Weld 96]) pose query planning as a traditional AI planning problem, and the algorithms used in different versions of Occam are modelled after state and plan space planners. A closer look at these systems reveals that the use of classical planners are used basically for the purpose of “stringing together” information gathering plans. This exercises only the “backward chaining” (and in the case of original Occam based on forward state space planners, the “forward chaining”) aspects of these planning techniques. Action interactions, which in some sense is the critical problem that these classical planners were designed to solve, are not critical to the pure information gathering problem. The source inversion approach described here provides a much simpler. Weld *et al.* seem to have realized this too. Razor ([Friedman *et al.* 97]) was originally

written to execute and prune the plans generated by Occam. Pruning was done through the use of LCW statements, and multiple plans were merged together to more efficiently gather information. The current version of Razor generates all its own plans apparently using the source inversion techniques [Duschka & Genesereth 97] described in this paper. Razor continues to use LCW statements to prune plans, and [Friedman *et al.* 97] is the first paper to discuss making use of pairwise subsumption LCW statements.

5.2 SAGE

The Sage system is developed by Knoblock and his co-workers [Arens *et al.* 96], and was originally intended to be a query planner for the SIMS project, that deals with heterogeneous distributed databases. Sage assumes information source descriptions are complete, and that no source has query constraints. Sage casts the information gathering problem as a query reformulation problem. Sage uses a modified version of UCPOP, a classical AI planner, to search for the correct sequence of reformulation operations that will transform the user's query into an equivalent query only on information sources. The problem with this approach is that it contains all the baggage of a general AI planner to search for reformulation plans, while the view inversion scheme detailed here is much simpler and can solve the same problem.

5.3 Information Manifold

The Information Manifold is an evolving project by Levy and his co-workers. We base our discussion here on the latest published account of the system [Levy *et al.* 96]. IM generates plans under the assumption of incomplete sources without query constraints, then it checks to see if the generated plan can be executed on information sources with query constraints. If the plan cannot be executed, it is thrown away and no plan is found. The planner makes use of LCW constraints during planning to prune unnecessary source accesses.

Early work on IM concentrated on the problem of characterizing the information sources, with an emphasis on efficiently isolating sources relevant to a query. IM was one of the first systems to popularize the idea of modeling sources as views on a global conceptual database.

IM places a very strong emphasis on finding "conceptually complete" plans. Levy shows that in the presence of complete information and fully relational sources, a conceptually complete plan for a conjunctive query cannot have more source calls than the number of conjuncts in the original query. This is not surprising once we note that once a complete source relevant to a conjunct of the query is isolated, no other source can give further information on that conjunct. Rajaraman *et. al.* [1995] extend these length bounds to consider the case where query sources have restrictions. They show that if there is a conceptually complete plan, then it cannot be longer than the number of conjuncts in the query plus the number of restricted variables. This result too is easy to see once we realize that we need at most one source to supply the values of each restricted variable. These length bounds are used to prune plan generation.

IM's emphasis on conceptual completeness is problematic in general, since in many cases, "completeness" of plans in terms of equivalence with the query, is impossible to guarantee.

As we mentioned earlier, the main aim of planning for information gathering should be seen as gathering all available information, rather than to prove that the gathered information is the only possible information according to the conceptual databases.

5.4 Infomaster

Infomaster, a Stanford Logic group project, apparently makes use of abductive logic to reason about a user's query. Not much has been published about the inner workings of the system other than [Duschka, Genesereth 97].

The source inversion method discussed in this paper was first published by Duschka (a member of the Infomaster project) in [Duschka 96] and later in [Duschka & Genesereth 97], but they apparently are not used by the Infomaster system.

5.5 XII and Planning with Incomplete Information

The XII [Golden *et al.* 96] is a general-purpose planner which was originally designed to help an autonomous agent plan in the presence of incomplete information. Other planners of this genre include Cassandra [?] and IPEM [?]. XII can handle both "causative" goals and "knowledge/information" goals. As an example, one could use XII to first compress all the ".ps" files in a directory, and then list all files which are below a certain size. The first is a causative goal, while the second is an information-gathering goal, whose outcome might change based on the causative actions that the agent might take before considering this goal (in this case, some of the postscript files which were above the size threshold before the compression was done, my get below the threshold after the compression, and thus become eligible tuples for the information gathering goal).

XII can in principle be used to solve the pure information gathering problems, with source calls modeled as information gathering actions with "knowledge" effects. In fact, the original work on LCW statements was done in the context of XII [?]. However, use of XII for pure information gathering turns out to be an over-kill. This is because the absence of "causative" changes to the environment around the information gathering agent (the contents of the information sources are not modified by the queries sent to them, for example) vastly simplifies the planning problem, facilitating specialized methods such as the ones we described in this paper. The XII methodology may be useful however once we consider variants of the information gathering problem that model updates to sources (either made by the information gatherer, or more likely, by the source providers). The work on maintaining LCW characterizations in the presence of updates, described in [?] may be particularly relevant here.

5.6 TSIMMIS

The TSIMMIS project doesn't make use of a world model to integrate information sources, so it doesn't fit directly into this framework. Rather, the project is concerned with joining a static set of two or more information sources together and precomputing all the possible

queries that can be answered by this group of information sources. In [Ullman 97], Jeffrey Ullman compares TSIMMIS with the Information Manifold.

6 Summary

This paper described the information gathering problem, which is that of integrating a set of autonomous information sources together so they can be queried as if they were a single information source. We showed how this problem can be solved under the assumption that information sources are relational databases by using views to describe the contents of the information sources. We also showed how local closed world statements can be used to make our algorithm more efficient by reducing the number of redundant information gathering plans. We discussed how the information gatherer can deal with failure when some information source no longer works. Finally, we discussed some implemented information gatherers and how they relate to the information presented in this paper.

7 Bibliography

References

- [Ambros-Ingerson & Steel, 88] J. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In Proc. 7th National Conf. on AI. 1988.
- [Arens *et al.* 96] Y. Arens, C. A. Knoblock, W. Shen. Query Reformulation for Dynamic Information Integration. To appear in *Journal of Intelligent Information Systems*, Boston, MA, 1996.
- [Duschka 96] Oliver M. Duschka. Generating Complete Query Plans Given Approximate Descriptions of Content Providers. *Stanford technical report*, 1996.
- [Duschka 97] Oliver M. Duschka. Query Optimization Using Local Completeness. *Proceedings of AAAI*, 1997.
- [Duschka, Genesereth 97] Oliver M. Duschka and Michael R. Genesereth. Query Planning In Infomaster. *Proceedings of the Twelfth Annual ACM Symposium on Applied Computing*, February 1997.
- [Duschka & Genesereth 97] Oliver M. Duschka and Michael R. Genesereth. Answering Recursive Queries Using Views. *Proceedings of AAAI*, 1997.
- [Duschka & Levy 97] Oliver M. Duschka and Alon Levy. Recursive plans for information gathering. *Proceedings of IJCAI*, 1997.
- [Etzioni *et al.* 97] O. Etzioni, K. Golden and D. Weld. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence*, 89(1-2), 113–148.

- [Etzioni *et al.* 96] 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.
- [Friedman *et al.* 97] Marc Friedman, Daniel S. Weld. Efficiently Executing Information-Gathering Plans. Submitted to *International Joint Conference on AI (IJ-CAI)*, 1997.
- [Golden *et al.* 96] Keith Golden, Daniel S. Weld and Oren Etzioni. Planning with Execution and Incomplete Information. UW CSE TR 96-01-09. <ftp.cs.washington.edu/pub/ai/tr-96-01-09.ps.gz>
- [Kwok & Weld 96] Chung T. Kwok and Daniel S. Weld. Planning to Gather Information. *University of Washington technical report UW-CSE-96-01-04*, 1996.
- [Levy 96] Alon Y. Levy. Obtaining Complete Answers from Incomplete Databases. *Proceedings of the 22nd VLDB Conference*, Mumbai (Bombay), India, 1996.
- [Levy *et al.* 95] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. *Proceedings of the 22nd VLDB Conference*, Mumbai (Bombay), India, 1996.
- [Levy *et al.* 96] 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.
- [Pryor & Collins, 96] Pryor, L. and Collins, G. Planning for Contingencies: A Decision-based Approach, Volume 4, pages 287-339.
- [Rajaraman *et al.* 95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering Queries Using Templates With Binding Patterns (Extended Abstract) *Proceedings of Principles of Database Systems (PODS)*, 1995.
- [Ullman 89] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volumes I & II*. Computer Science Press, Rockville MD, 1989.
- [Ullman 97] Jeffrey D. Ullman. Information Integration Using Logical Views. Invited talk at the *International Conference on Database Theory*, Delphi, Greece, 1997.