Exploiting Run-Time Information for Efficient Processing of Queries *

Craig A. Knoblock
Information Sciences Institute and
Department of Computer Science
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
knoblock@isi.edu

Alon Y. Levy
AI Principles Research Dept.
AT&T Bell Laboratories
600 Mountain Ave., Room 2C-406
Murray Hill, NJ 07974
levy@research.att.com

Abstract

Information agents answer user queries using a large number of diverse information sources. The key issue in their perfomance is finding the set of information sources relevant to a query. Previous work has considered determining relevance soley based on compile-time analysis of the query. We argue that at compile-time, it is often not possible to significantly prune the set of sources relevant to a query, and that run-time information is needed. We make the following contributions. First, we identify the different types of information that can be obtained at run-time, and how they can be used to prune information sources. Second, we describe an algorithm which naturally extends query planning algorithms to exploit run-time information. Third, we describe the *discrimination matrix*, which is a data structure that identifies the information that can be used to help discriminate between different possible sources.

Introduction

Information gathering agents are programs that answer user queries using a large number of diverse information sources (e.g., sources on the Internet, company wide databases). These information sources do not belong to the agent, rather they are provided by autonomous sources, possibly for a fee. As such, an information agent does not maintain any real data, rather it only has *descriptions* of the contents of the available information sources. An agent has a domain model of its area of expertise (e.g., a class hierarchy describing properties of objects in its domain), and a description of how the contents of an information source relates to the classes in the domain model of the agent. User queries are posed using the domain model of the agent. Given a query, an agent serves as a mediator between the user and the information sources, by decomposing the query, sending requests to the appropriate information sources, and possibly processing the intermediate data. As such, the information agent frees the user of being aware of and sending queries directly to the information sources.

Several characteristics of this problem require us to develop solutions beyond those considered traditionally in knowledge and data-base systems. The main characteristic is that the number of information sources will be very large, and the agent has only the descriptions of the sources in determining which ones are relevant to a given query. Second, some information will reside redundantly in many sources, and access to sources will not always be possible (e.g., network failures) and may be expensive (either in time or in money). Finally, the information sources will be autonomous and, as a result, use different languages, ontologies and protocols.

To answer queries efficiently, an information agent must be able to determine precisely which information sources are relevant to a given query, and to retrieve as little as possible data from the relevant sources. Previous work on information agents (e.g.,SIMS (Knoblock, Arens, & Hsu 1994), the Information Manifold (Kirk, Levy, & Srivastava 1995)) have focussed on determining relevant sources by using information available at compile time (i.e.,the query and the descriptions of information sources). However, as the following example shows, it is not always possible to significantly prune the set of information sources based solely on compile-time information.

Example 1: Suppose we are given the following query asking for papers authored by AAAI fellows:

$$\mathsf{AAAI-Fellow(x)} \wedge \mathsf{Papers(x,y)}.$$

Furthermore, suppose we have one database giving us the listing of AAAI fellows, and numerous information sources providing titles of papers. By considering the query itself, the information agent is only able to prune information sources which provide *only* papers

*The first author is supported in part by Rome Laboratory of the Air Force Systems Command and the Advanced Research Projects Agency under contract no. F30602-91-C-0081, and in part by the National Science Foundation under grant number IRI-9313993. The second author is supported by AT&T Bell Laboratories. The views and conclusions contained in this paper are the author's and should not be interpreted as representing the official opinion or policy of DARPA, RL, NSF, Bell Labs, or any person or agency connected with them.

whose topic is not related to AI,[1] still leaving a large number of possibly relevant information sources. One method that may also be useful in our context is *sideways information passing* (Ullman 1989), i.e., using the values obtained by solving one subgoal to restrict the search for the a subsequent subgoal. However, this will not always suffice. For example, suppose the first subgoal yields the binding RonBrachman. The system does not know enough *about* Ron Brachman (e.g., affiliation, specific expertise) to prune the information sources considered for the second subgoal. One possibility is to obtain such information from the source providing the bindings for the first subgoal. For example, the AAAI database of fellows may also contain affiliations. Finally, it may even be beneficial for the query processor to *add* new subgoals to the query in order to obtain additional information about bindings in the query. For example, if our query were simply Papers(RonBrachman,y), the query processor may add subgoals (e.g., Affiliation(RonBrachman,y) that will provide additional information about Ron Brachman. □

This paper considers the problem of obtaining and utilizing additional information at run-time. Our first contribution is to identify the kinds of information which can be obtained at run-time, how they can be used to speed up query processing, and different ways in which such information can be obtained. For example, we can obtain additional information *about* bindings that appear in the query, additional information about the information sources, or information about *where* information has been found. Although some information can be gleaned from solving subgoals that appear in the query, we argue that it may be beneficial for the query processor to *add* subgoals to the query in order to obtain additional information. Our second contribution is a novel algorithm that extends traditional query planning algorithms to consider plans that may include additional information gathering actions (i.e., additional subgoals). A key component of this algorithm is a data structure, the *discrimination matrix*, that enables us to evaluate the utility of additional information gathering actions. Informally, the discrimination matrix tells us which of the information sources we may be able to prune if we perform a proposed information gathering action, thereby enabling us to estimate the cost of plans with additional subgoals.

Domain Model and Source Descriptions

An information agent serves as a mediator to multiple information sources. In order to integrate these sources, it has a representation of its domain of expertise, called the *domain model*, and a set of descriptions of information sources. We assume the domain model is represented in KL-ONE type language (Brachman & Schmolze 1985). Such a language contains unary relations (called *concepts*) which represent classes of objects in the domain and binary relations (*roles*) which describe relationships between objects. Concepts and roles can be either primitive or complex. Complex concepts and roles are defined by a *description* which is composed using a set of constructors (which vary from one language to another). Below we show a representative set of constructors that can be used in descriptions: (A denotes a primitive concept, C and D represent arbitrary descriptions and R denotes an arbitrary role):

$C, D \rightarrow A$ | (primitive concept)
$\top$ | $\bot$ | (set of all objects, the empty set)
$C \sqcap D$ | $C \sqcup D$ | (conjunction, disjunction)
$\neg C$ | (complement)
$\forall R.C$ | (universal quantification)
$\exists R.C$ | (existential quantification)
$(R < a)$ | $(R > a)$ (range constraints)
$(\geq n\ R)$ | $(\leq n\ R)$ (number restrictions)
(fills a R) (filler restriction)
(oneOf R $\{a_1, \ldots, a_n\}$) (restriction on fillers)

Because of space limitations we will not go into the details of the semantics of such a language. As two examples, the class $(\geq n\ R)$ represents the class of objects who have at least n fillers on the role R; the class (fills a R) represents the objects who have a as *one* of the fillers on role R. As an example of a domain model, we may have a primitive class Person, the class AI-Researcher, defined by (Person ⊓ (fills research occupation)⊓(fills field AI)), and the class AI-advisor, defined by (AI-Researcher⊓(≥ 1 student)).

In order to be able to use the data in external information sources, the agent must have descriptions of these sources. In our discussion, we assume that an information source provides data about some class. Formally, a description of an information source S is of the form $(D_S, r_1^s, \ldots, r_n^s)$. The first element, D_S is a description in our language, and $r_1^s, \ldots, r_n^s$ are role names. The description states that the source S contains individuals which belong to the class D_S, and it contains role fillers of the roles $r_1^s, \ldots, r_n^s$.

As examples of information sources, we may have ((AI-Researcher ⊓ (fills affiliation Bell Labs)),email-address, paper-titles) providing the email addresses and papers of Bell Labs researchers, and ((AI-Advisor ⊓ (fills affiliation CMU)),students, paper-titles) providing the students names and paper titles of advisors at CMU.

Given a query, the information agent needs to determine which information sources are relevant to it. In previous work (Arens *et al.* 1993; Levy, Sagiv, & Srivastava 1994) , we showed how to determine the relevant information sources based on the query and the descriptions of the information sources. As a simple example, suppose our query is AAAI-Fellow(x) ∧ Papers(x,y). The system can infer from looking at the query that since AAAI-Fellow is a subclass of AI-researcher, the only paper repositories that need to

[1] And not even those, under the likely assumption that our esteemed fellows make contributions in other fields as well.

be considered for the second subgoal are those that *may* provide papers by AI researchers. Hence, the two sources described above will be relevant, (because they both provide the role paper-titles for classes that are subsumed by AI-researcher), but a source that provides only papers by Historians at CMU would be deemed irrelevant.

Run-time Information

There are several reasons why we cannot significantly prune information sources at compile-time:

1. The information agent does not know enough *about* the bindings in order to prune information sources. For example, the query processor does not know enough *about* Ron Brachman (e.g., affiliation) in order to prune paper repositories. This is because the domain model of the agent describes properties classes of objects in the domain, but not of the (probably huge number of) individuals in the domain.
2. According to the descriptions of the information source, there may be a large number of sources that contain a specific piece of information. It is hard at compile-time to know *which* of the information sources are likely to have the desired data, and therefore where to search first.
3. Descriptions of information sources may not be sufficiently detailed, entailing that they are relevant or when they are not.

Types and Uses of Run-Time Information

We identify three kinds of run-time information that is useful to speed up query answering. These can be viewed as directly addressing the problems outlined above:

Information about individuals: At run-time we can obtain information about individuals that enables us to determine that they belong to a more *specific* domain class than can be determined solely by the query. This information can be used to prune information sources considered for subsequent subgoals. Specific class information can be obtained by finding role fillers of the individual (e.g., finding the affiliation of Ron Brachman enables us to deduce that he is a member of class Bell-Labs-researcher, rather than the more general class Researcher). Information about an individual can also be obtained by finding out *how many* fillers it has on a specific role, or by range constraints on its fillers.

Information about location of individuals: Information sources may contain only a *subset* of a class of information specified in a query. Therefore, knowing that an individual was found in a specific information source can be used in subsequent subgoals that involve this individual. For example, if we found the individual a in source S, and a subsequent subgoal asks for the filler of a role R of a, we will first check whether S contains fillers for R (which will be known in the description). Some information sources may belong to the same provider (e.g., a collection of university databases). Therefore, if an individual was found in one of the sources belonging to a specific provider, it is likely that we will find other information concerning this individual in one of the sources of that provider.

Additional information about sources: A provider of an information source may not give the most detailed description possible of the source. At run-time we can obtain information that enables us to refine the existing description, such as the size of the information source (which can be used later in determining subgoal ordering, as done in traditional database systems), constraints on the individuals in an information source (e.g., the age of all individuals is between 30 and 50).[2]

Obtaining Run-Time Information

There are several ways to obtain run-time information that only require minor changes to the traditional query processing algorithms. First, run-time information can be found by simply solving subgoals that already exist in the query. Specifically, instead of passing only the *values* of the bindings that are found in solving a subgoal, we can also pass additional knowledge about their *type*. In particular, if, while solving a subgoal, we found an individual a in an information source whose description is $(D_S, r_1^s, \ldots, r_n^s)$, then we can infer that a is of type D_S, and use that to prune the information sources relevant to subsequent subgoals involving a. Such generalized sideways information passing is the basis of the query processor of the Information Manifold system (Kirk, Levy, & Srivastava 1995). Second, finding an individual in a specific information source can also be used to prune the sources considered for the *same* subgoal in the query. For example, lacking any other additional information about Ron Brachman, we may start searching for his papers in several repositories, until we find some of his papers in a repository of AI papers. At that point, we can infer that Ron Brachman is an AI researcher, and therefore focus our subsequent search to first consider repositories that may contain AI papers.

A third method, which is the focus of this paper, is based on *actively* seeking information about individuals in the query. Instead of relying on information obtained from solving subgoals that appear in the query, our approach involves adding *new* subgoals to the query, which may yield the desired information. For example, given the query Papers(RonBrachman,y), our approach will attempt to add subgoals that yield useful information about Ron Brachman, such as Affiliation(RonBrachman,x), which, if found, will enable

[2]Note that information obtained in this fashion is not guaranteed to persist over time.

us to prune the paper repositories considered for the second subgoal. The key question that needs to be addressed is how to find such new subgoals that will reduce the overall cost of solving the query (which now includes the cost of solving the new subgoal). We formalize and solve this problem in the following sections.

A Cost Model

In order to address the question of the utility of obtaining additional information, we formally define the cost model we use to compare among various query plans. Numerous cost models have been proposed for evaluating database query plans (e.g., (Jarke & Koch 1984; Greiner 1991)). Since we do not want our analysis to be specific to any of these single models, we will consider a model that hides the model-specific details, and relies on features common to a wide array of models. We assume our query is specified as follows:

$$Q = Q_1(\bar{X}_1) \wedge \ldots \wedge Q_n(\bar{X}_n)$$

The Q_i's are predicates in the domain model, and the $\bar{X}_i$'s are tuples of variables or constants. Given a partial ordering $\mathcal{O}$ of the subgoals and a specific information source S that may be relevant to the subgoal $Q_i(\bar{X}_i)$, we assume that there is some function $c(Q_i(\bar{X}_i), S, \mathcal{O})$ that provides an estimate of the cost for accessing and retrieving the required data from S. In different cost models, this function may depend differently on factors such as the size of S, the expected number of relevant facts in S, considering the binding pattern of $\bar{X}_i$, or the initial cost of setting up a connection with S.

Given the query Q, we denote by $\mathcal{S}_i$ the set of sources that are deemed relevant to the subgoal Q_i, using the information present in the query (as outlined in the previous section). Since we may not know which information source contains the requested data, the overall cost of retrieving the data for Q_i will be the sum of the costs of retrieving the information from all of the relevant information sources, i.e.,

$$C(Q_i, \mathcal{S}_i, \mathcal{O}) = \sum_{S \in \mathcal{S}_i} c(Q_i, S, \mathcal{O}).$$

We assume there is a function $f(C(Q_1, \mathcal{S}_1, \mathcal{O}), \ldots, C(Q_n, \mathcal{S}_n, \mathcal{O}))$ that combines the costs of the single subgoals to determine the overall cost of a plan to answer this query. This function will determine heavily on the ordering $\mathcal{O}$.

Our approach to actively seeking information about objects involves considering query plans that contain one or more additional subgoals, called *discrimination queries*, of the form $p(a, X)$, where a is a constant, denoting an individual, that appears in the query.[3] Adding such subgoals changes the cost of the query plan in two ways. First, it adds the cost of solving the additional subgoal, which can be estimated as described above. Second, it enables us to prune the sets $\mathcal{S}_i, \ldots, \mathcal{S}_n$ of the information sources relevant to subsequent subgoals. The key to extending our cost model to consider the effect of additional subgoals is to estimate the value of $C(Q_i, \mathcal{S}_i, \mathcal{O})$, without knowing the values returned for $p(a, X)$, and therefore not knowing which subset of $\mathcal{S}_i$ will be relevant to the query.

The *discrimination matrix*, whose construction is described in detail in Section , is used to estimate these costs, by estimating the number of information sources that will be relevant, given the answer to the new subgoal. A discrimination matrix is built for a given role R in the domain, and it tells us how finding the value of the role R of an object will discriminate between the possible information sources. Clearly, the number of possible values for the role R may be infinite. The first piece of information given by the discrimination matrix is a partition of the set of values of R to *regions*, such that two values of R in the same region will deem the same set of information sources relevant. For example, if the R is a role whose range is the real numbers, and the descriptions of the information sources only mention constraints of the form $R < 100$ and $R > 200$, then the regions will be $\{(-\infty, 100), [100, 200], (200, \infty)\}$. Given these regions, the discrimination matrix tells us which information sources are relevant to the query, given that the role R falls in that region. Formally, given a set of information sources $\mathcal{S}$, we use the discrimination matrix to obtain a partition of $\mathcal{S}$ into the regions of values of R. If r is a region of R, its partition will contain the subset of $\mathcal{S}$ that are relevant if we are looking for individuals whose filler of R is in r. A role R provides good *discrimination* for a subgoal Q_i if it partitions the set $\mathcal{S}_i$ evenly over many regions.

It should be noted that the discrimination matrix can also be used to partition the sources given *multiple* discrimination queries about an object a. To determine the discrimination achieved by multiple subgoals, we simply take the cross products of the regions given by each subgoal. For example, suppose the analysis of the query shows that there are six potentially relevant information sources, $\mathcal{S} = S_1, \ldots, S_6$, for retrieving the information for a subgoal Q_i. We have two discrimination matrices, one for role R_1 and one for R_2, each having three regions. Suppose that for the role R_1 we obtain the partition $\{\{S_1, S_2\}, \{S_3, S_4, S_5\}, \{S_6\}\}$, and for R_2 the partition is $\{\{S_1\}, \{S_2, S_3\}, \{S_4, S_5, S_6\}\}$. Taking the cross products of these partitions to compute the discrimination that would be obtained by finding both R_1 and R_2, would yield 9 regions, the

[3] For simplicity of exposition we are assuming that the additional subgoals involve an object appearing in the original query. This may be generalized straightforwardly to the case where the discriminating query includes only variables from the Q. However, in that case the cost analysis will depend on whether we are solving the query tuple-at-a-time (as in Prolog), or using traditional database methods in which we solve the query by some sequence of join operations on the solutions of each subgoal.

non-empty ones being:[4]

$$\{S_1\}^{1,1}, \{S_2\}^{1,2}, \{S_3\}^{2,2}, \{S_4, S_5\}^{2,3}, \{S_6\}^{3,3}.$$

Finally, we use the partitions to estimate the cost of subgoals after a discriminating subgoal has been executed as follows. Suppose $\mathcal{I}_1^i, \ldots, \mathcal{I}_m^i$ is the partition by a role R of the sources $\mathcal{S}_i$ that are relevant to Q_i. Since we do not know in *which* of the regions the answer to the discrimination query will be, we assume the worst-case. The advantage of a worst-case estimate for the discrimination is that we can guarantee that we will never spend more time gathering additional information than the time to simply execute the query against all relevant sources. In the worst case the cost of the Q_i will be the maximum cost of all the partitions, i.e.,

$$C_R(Q_i, \mathcal{S}_i, \mathcal{O}) = max(C(Q_i, \mathcal{I}_1^i, \mathcal{O}), \ldots, C(Q_i, \mathcal{I}_m^i, \mathcal{O})).$$

Using this function, we can now compute the overall cost of answering the query Q using the additional discriminating query $R(a, X)$. If this cost is lower than the original cost of solving Q with the ordering $\mathcal{O}$, then we have saved some work. Note that we can execute several discriminating queries in Q, and the cost of each would simply be added to the total. Also, we can perform the discrimination recursively, i.e., try to reduce the number of sources needed in order to solve the discriminating queries.

Searching the Space of Query Plans

Given our cost model, we are now able to search the new space of plans. Originally, the space of plans included all the possible orderings of the subgoals of Q.[5] The new space includes plans that involve discriminating queries. Specifically, if $\alpha_1, \ldots, \alpha_m$ is a plan (where the α_i's are either subgoals of Q or new discriminating subgoals), then so is $\alpha_1, \ldots, \alpha_i, \beta, \alpha_{i+1}, \ldots, \alpha_m$, where β is a new discriminating query involving variables or constants appearing in $\alpha_{i+1}, \ldots, \alpha_m$. We can extend any algorithm for searching the original set of plans as follows. Given any plan P from the original space, we search through the space of plans in which one or more discriminating queries is added to P. That is, we select one of the subgoals, and a possible discriminating query, and evaluate the cost of the plan including the new subgoal. The search terminates where there are no additional discriminations that produce a cheaper plan than the best one found so far.

This approach for deciding on the best plan for evaluating a query has several useful features:

- If the cost of performing the discrimination is more expensive than just executing the query, then the algorithm would chose to just execute the query.
- If there is more than one possible discrimination that can be performed, the algorithm will consider the various combinations.
- If any of the discriminating queries can be retrieved from multiple sources, then discrimination will be considered recursively and the final choice will be made based on the overall cost.

The Discrimination Matrix

Recall that we view each information source S as providing fillers of roles of some instances of a domain class C_S. Hence, if we need the value of a certain role P of some object a, we will consider all sources who provide the role P, and such that C_S does not contradict the class information we already have about a. The discrimination matrix of a role R tells us which information sources would be relevant if we *also* knew the value of the filler of R of the object a. Given a set of information sources $\mathcal{S}$, which are deemed relevant from the analysis of the query, we can use the discrimination matrix of R to partition $\mathcal{S}$ depending on the value of R, which we may obtain at run-time.

Below we describe an algorithm for constructing and updating the discrimination matrix. The matrix is built by exploiting the descriptions of the contents of the information sources, and is a persistent data structure that needs to be modified only when an information source is added or deleted, and is therefore cheap to maintain. In particular, it can be built at compile-time before answering any queries. We distinguish roles whose range are numeric attributes from those whose range is non-numeric. In Section we describe an algorithm for creating a discrimination matrix for non-numeric roles, which uses constraints specified by the constructors fills and oneOf to partition the information sources. Section describes an algorithm that uses constraints specified by the constructors $>$ and $<$ to partition sources. It should be noted that we focus on the above constructors because these are the most likely to yield discrimination between sources. For example, it is likely that paper repositories will differ based on the affiliation of the authors, which is specified by the fills constructor. Information source providing the same data (e.g., stock market prices) may be distinguished based on the date of the data, which is specified using the $<$ and $>$ constructors. Finally, it should be noted that by treating the number of fillers as a numeric attribute, the algorithm presented in Section can be directly used to determine discrimination when the discrimination query returns the number of fillers of a role R.

Non-numeric Roles

In the algorithm below assume that the role R can have a single filler. If the role has multiple fillers, the algorithm is essentially the same, except that we ignore the fills constructor. Knowing the value b of a filler of

[4] The superscripts denote the original regions from R_1 and R_2 respectively.

[5] For some execution models, only partial orderings of the subgoals need to be considered.

the role R of an instance can affect the relevance of an information source S in two ways:

1. If the description C_S entails (fills R a), then S will be relevant if and only if a = b.
2. If the description C_S entails ((oneOf R $\{a_1, \ldots, a_m\}$), then S will be relevant if and only if b $\in$ $\{a_1, \ldots, a_m\}$.

Hence, if $\mathcal{A}$ is the set of constants that appear in fills and oneOf constraints in the descriptions of the information sources, then the possible values of R can be classified into $|\mathcal{A}|+1$ regions: one region for every value in $\mathcal{A}$ and one region for all values not mentioned in $\mathcal{A}$. The algorithm for building the discrimination matrix is shown in Figure 1. Informally, the algorithm creates a hierarchy of *labels*. Each label denotes a subset of the regions of R. With each label L, the algorithm associates a set of information sources $Sources(L)$. An information source S will be in $Sources(L)$ if the description of S entails that the value of R must be in one of the regions in L. Given a set of information sources $\mathcal{S}$, we use the discrimination matrix to partition them as follows. The set of information sources in the partition of a value r are those sources in $\mathcal{S}$ that appear in the set associated with *some* label that includes r.

It should be noted that although the number of possible labels that can be generated is exponential in the number of constants in $\mathcal{A}$, the algorithm will only generate at most a number of labels as the number of information sources.[6] Therefore, since adding a new information source to the matrix can be done in time $O(log(n))$, where n is the number of information sources, the overall complexity of the algorithm is $O(nlog(n))$,

Initialize matrix with the label L^1 =top and $Sources(L^1) = \emptyset$.
for every source S whose associated class is given by the description C_S
do: if $C_S \models$ (oneOf R $\{a_1, \ldots, a_n\}$) **then**
if there exists a label $L = \{a_1, \ldots, a_n\}$ **then** add I to $Sources(L)$.
else create a new label $L = \{a_1, \ldots, a_n\}$ with $Sources(L) = \{I\}$ and place L in the hierarchy.
if $C_S \models$ (fills R b) **then** do the same for the set $\{b\}$.
if C_S does not entail any such constraints on the fillers of R
then add I to $Sources(L^1)$.

Figure 1: Creating a discrimination matrix for non-numeric roles. TOP denotes the set of all regions of R.

Numeric Roles

When the values of a role are numeric, they can affect the relevance of an information source mainly through constraints of the form $(< R\ a)$ or $(> R\ a)$.[7] We can assume that the description of an information source entails two constraints, one of the form $(> R\ a_{low})$ and the other of the form $(< R\ a_{high})$ (note that a_{low} may be $-\infty$ and a_{high} may be ∞). As in the non-numeric case, if $\mathcal{A} = \{a_1, \ldots, a_m\}$ is the sorted set of constants that appear in the descriptions of information sources, then the possible regions of R are: $\{(-\infty, a_1)$, $(a_1, a_2), \ldots, (a_{m-1}, a_m)$, $(a_m, \infty)\}$.

The algorithm, shown in Figure 2, builds a vector of triplets (a_i, S_i^+, S_i^-). The number a_i is in the set $\mathcal{A} \cup \{-\infty, \infty\}$, and S_i^+, S_i^- are sets of information sources. The set S_i^+ are the information sources who *become* relevant when the value of R is in the range (a_i, a_{i+1}), and are not relevant when the value of R is in the range (a_{i-1}, a_i). Similarly, the set S_i^- are the information sources that are irrelevant when $R \in (a_i, a_{i+1})$ and relevant when $R \in (a_{i-1}, a_i)$. The discrimination matrix can be used to partition a set of sources $\mathcal{S}$ as follows. The sources in the partition of a region $r = (a_i, a_{i+1})$, are the sources in $\mathcal{S}$ that appear in S_j^+ for some $j \leq i$, and do not appear in the S_j^- for any $j \leq i$.

Inserting an information source into the matrix can be done in time $O(log(n))$ in the number of information sources, n. Therefore, the overall running time of the algorithm is $O(nlog(n))$. Since the determination of the regions of R requires that we sort the values in $\mathcal{A}$, it is clear that $O(nlog(n))$ is also a lower bound on the time to build the discrimination matrix.

Begin with the vector $\mathcal{V} = \{(-\infty, \emptyset, \emptyset), (\infty, \emptyset, \emptyset)\}$.
for every source S whose associated class is given by the description C_S and $C_S \models (R > a_{low}) \wedge (R < a_{high})$, **do:**
if a_{low} is in position i of $\mathcal{V}$
then add I to S_i^+,
else add $(a_{low}, I, \emptyset)$ to $\mathcal{V}$.
if a_{high} is in position i of $\mathcal{V}$
then add I to S_i^-,
else add $(a_{low}, \emptyset, I)$ to $\mathcal{V}$.

Figure 2: Creating a discrimination matrix for numeric roles.

Example 2: Suppose we are given 3 sources with the following constraints on R, $S1 : (1 < R < 3)$, $S2 : (2 < R < \infty)$ and $S3 : (1 < R < 2)$. The algorithm would begin with the vector: $\{(-\infty, \emptyset, \emptyset), (\infty, \emptyset, \emptyset)\}$. Considering $S1$ would result in the vector $\{(-\infty, \emptyset, \emptyset), (1, \{S1\}, \emptyset), (3, \emptyset, \{S1\}), (\infty, \emptyset, \emptyset)\}$. After processing $S2$ we would get $\{(-\infty, \emptyset, \emptyset), (1, \{S1\}, \emptyset)$, $(2, \{S2\}, \emptyset)$, $(3, \emptyset, \{S1\})$, $(\infty, \emptyset, \{S2\})\}$. Finally, after considering $S3$ the resulting vector will be $\{(-\infty, \emptyset, \emptyset)$, $(1, \{S1, S3\}, \emptyset)$, $(2, \{S2\}, \{S3\})$, $(3, \emptyset, \{S1\})$, $(\infty, \emptyset, \{S2\})\}$.

[6] Furthermore, note that if the number of labels created is close to the number of sources, that would actually mean that R is providing good discrimination.

[7] In this section we only discuss < and > constraints, but the algorithm can be easily extended to consider $\leq$ and $\geq$. For simplicity of exposition we also ignore the distinction between open and closed regions.

From this matrix, we can deduce for example that in the region $(2,3)$ the relevant information sources will be $S1$ and $S2$. □

Related Work

Our work can be viewed as a form of semantic query optimization (SQO) (King 1981; Chakravarthy, Grant, & Minker 1990; Hsu & Knoblock 1994; Levy & Sagiv 1995), where new subgoals are added to a query by analyzing the integrity constraints known about the data. In our context, the analogue of integrity constraints are the descriptions of the information sources. The key issue in SQO, that has prevented it from wide usage in database systems, is that often the analysis of the integrity constraints introduces additional joins to the query, or even disjunctive subgoals, which are usually considered very expensive operations. In contrast, our work provides a cost model that better estimates *when* such joins will be beneficial. Furthermore, a key aspect in our cost model, which has no parallel in standard SQO, is the ability to determine that the *number* of information sources accessed is reduced.

Another related idea appears in the work by Finger (Finger 1987) on the use of *supersumption* to further constrain a design to find solutions faster. The idea is to add constraints to the design by inferring a set of ramifications from the partial design. These constraints can reduce the design space by constraining the values of certain aspects of the design. There are also interesting examples of supersumption that cannot be deduced from the ramifications, but Finger does not suggest an approach to adding these constraints. His work assumes that a set of axioms is given that can be used to augment a design. In our work, we do not assume that the constraints can be inferred directly, rather we developed an approach to gathering additional information at run time that is likely to further constrain a query.

The recent work on Softbots for Unix by Etzioni et al. (Etzioni & Weld 1994) exploits the use of additional information gathering actions to determine where to locate information. The information gathering actions are in the context of a software agent that can manipulate the environment as well as gather information. However, the specific information gathering goals are encoded as explicit preconditions of the planning operators. In this paper we present a more general information gathering framework that identifies a variety of different types of information about bindings that can be exploited and presents an approach to automatically selecting useful information gathering goals.

Conclusion

This paper provides an important capability for efficiently locating information, in a setting that involves a large number of sources which are costly to access. We have argued for the need for exploiting information obtained at run-time, and described the kinds of run-time information that would be useful for a query processor. We presented an algorithm that extends classical query planning algorithms to exploit run-time information. The algorithm is based on a cost model that is able to estimate the utility of additional discriminating queries at run-time. An interesting question raised by our work is to find properties of the extended space of plans that enable us to search it more efficiently.

References

Arens, Y.; Chee, C. Y.; Hsu, C.-N.; and Knoblock, C. A. 1993. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems* 2(2):127–158.

Brachman, R., and Schmolze, J. 1985. An overview of the KL-ONE knowledge representation system. *Cognitive Science* 9(2):171–216.

Chakravarthy, U. S.; Grant, J.; and Minker, J. 1990. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems* 15(2):162–207.

Etzioni, O., and Weld, D. S. 1994. A softbot-based interface to the internet. *Communications of the ACM* 37(7).

Finger, J. J. 1987. *Exploiting Constraints in Design Synthesis.* Ph.D. Dissertation, Department of Computer Science, Stanford University.

Greiner, R. 1991. Finding optimal derivation strategies in a redundant knowledge base. *Artificial Intelligence* 50(1):95–116.

Hsu, C.-N., and Knoblock, C. A. 1994. Rule induction for semantic query optimization. In *Proceedings of the Eleventh International Conference on Machine Learning.*

Jarke, M., and Koch, J. 1984. Query optimization in database systems. *ACM Computing Surveys* 16(2):111–152.

King, J. J. 1981. *Query Optimization by Semantic Reasoning.* Ph.D. Dissertation, Stanford University, Department of Computer Science.

Kirk, T.; Levy, A. Y.; Sagiv Y.; and Srivastava, D. 1995. The information manifold. In *In Working Notes of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments.*

Knoblock, C.; Arens, Y.; and Hsu, C.-N. 1994. Cooperating agents for information retrieval. In *Proceedings of the Second International Conference on Cooperative Information Systems.*

Levy, A. Y., and Sagiv, Y. 1995. Semantic query optimization in datalog programs. In *Proceedings of the ACM Symposium on Principles of Database Systems*, San Jose, CA.

Levy, A. Y.; Sagiv, Y.; and Srivastava, D. 1994. Towards efficient information gathering agents. In *Working Notes of the AAAI Spring Symposium on Software Agents, Stanford, California*, 64–70.

Ullman, J. D. 1989. *Principles of Database and Knowledge-Base Systems*, volume 2. Rockville, Maryland: Computer Science Press.