Querying Heterogeneous Information Sources Using Source Descriptions

Alon Y. Levy
AT&T Research
levy@research.att.com

Anand Rajaraman*
Stanford University
anand@cs.stanford.edu

Joann J. Ordille
Bell Laboratories
joann@research.att.com

Abstract

We witness a rapid increase in the number of structured information sources that are available online, especially on the WWW. These sources store interrelated data on topics such as product information, stock market information, entertainment, etc. We would like to use the data stored in these databases to answer complex queries that go beyond keyword searches. We describe the Information Manifold, an implemented system that provides uniform access to a heterogeneous collection of more than 100 information sources, on the WWW. IM contains declarative descriptions of the contents and capabilities of the information sources. We describe algorithms that use the source descriptions to prune efficiently the set of information sources for a given query and practical algorithms to generate executable query plans. We also present experimental studies that indicate that the architecture and algorithms used in the Information Manifold scale up well to several hundred information sources.

1 Introduction

We witness a rapid increase in the number of structured information sources that are available online. The World-Wide Web (WWW), in particular, is a popular medium for interacting with such sources. The WWW is usually regarded as an interconnected collection of unstructured documents. However, a large number of structured information sources are now becoming available on the Web. These sources include both free and commercial databases on product information, stock market information, real estate, automobiles, and entertainment. The interface to such sources is typically a collection of *fill-out forms.* The query answer usually takes the form of an HTML document that is very structured, and can be parsed and converted into a set of tuples or more complex data types. There are other structured information sources that are available not on the WWW such as name servers, bibliographic sources, and university-wide and company-wide information systems, and they too provide query interfaces.

Most search tools available for the WWW today (e.g., AltaVista, Lycos, Inktomi, Yahoo) are based on keyword search. Keyword search is a useful way to search a collection of unstructured documents, but is not effective with structured sources. Currently, the interaction with such a large collection of structured sources is done manually. The user must consider the list of sources available, decide which ones to access, interact with each one individually, and manually combine answers from different sources. We would like to use the data stored in these databases to answer complex queries, and provide a *uniform* interface to the sources. In particular, the user should be able to express *what* he or she wants, and the system will find the relevant sources and obtain the answers, possibly by combining data from multiple sources.

Example 1.1 Suppose we are interested in purchasing a car. The parameters of interest to us are its category (sportscar or sedan), price, year of manufacture, model, and the car reviews. We ask query Q: *Get the price and reviews of sportscars for sale that were manufactured no earlier than 1992.* Suppose we have access to the online information sources shown in Figure 1, among many others. Some of the sources are obviously not useful to answer Q. We can straightaway determine that Source 4 is not useful to answer this query, because it has no information about cars. We can also conclude that Source 3 is not relevant. Here the reasoning is more subtle: we are interested only in cars manufactured after 1992, whereas Source

*Part of this work was done while this author was visiting AT&T Bell Laboratories.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 22nd VLDB Conference
Mumbai(Bombay), India, 1996

Source 1: Used cars for sale. Accepts as input a category or model of car, and optionally a price range and a year range. For each car that satisfies the conditions, gives model, year, price, and seller contact information.
Source 2: Luxury cars for sale. All cars in this database are priced above $20,000 Accepts as input a category of car and an optional price range. For each car that satisfies the conditions, gives model, year, price, and seller contact information.
Source 3: Vintage cars for sale (cars manufactured before 1950). Accepts as input a model and an optional year range. Gives model, year, price, and seller contact information for qualifying cars.
Source 4: Motorcycles for sale. Accepts as input a model and an optional price range. Gives model, year, price, and seller contact information.
Source 5: Car reviews database. Contains reviews for cars manufactured after 1990. Accepts as input a model and a year. Output is a car review for that model and year.

Figure 1: Example information sources.

3 has information only on cars manufactured before 1950. We are left with sources 1, 2, and 5 and two possible plans to answer Q:

1. Ask Source 1 for the models and prices of all sportscars manufactured after 1992. For each model, obtain a review from the Source 5.

2. Ask Source 2 for the models, years, and prices of sportscars. From the $\langle Model, Year, Price \rangle$ tuples that result, select only those where $Year \geq 1992$. For each model in the selected tuples, obtain a review from Source 5.

Notice that in plan 1 we took advantage of the capability of Source 1 to select a specified year range, whereas in plan 2 we had to do the selection ourselves because Source 2 cannot do it for us. Also note that the outputs of Sources 1 and 2 are enough to satisfy the inputs requirements of Source 5 (i.e., the year and model of the car). For example, if Source 5 would also require more specific information about the car (e.g., number of doors, engine type) in order to return a review, we would not be able to combine information from these three sources. It is possible to verify that these are the only two query plans to answer Q using these information sources. The answer to Q is the union of the sets of tuples produced by executing these two plans. $\square$

One of the key difficulties in providing access to a large collection of information sources is that several sources store interrelated data, and any query-answering system must understand and exploit the relationships between their contents. In particular, since the number of sources is very large, we must have enough information about the sources that enables us to prune the sources accessed in answering a specific query, and we must have effective techniques for pruning sources. Second, many sources are not full-featured database systems and can answer only a small set of queries over their data (for example, forms on the WWW restrict the set of queries one can ask). Moreover, most sources contain *incomplete information.* For example, there are several information sources advertising cars for sale. No single source contains information on all cars for sale.

We describe the Information Manifold (IM), a fully implemented system that provides uniform access to a heterogeneous collection of more than 100 information sources on the WWW. IM tackles the above problems by providing a mechanism to describe declaratively the contents and query capabilities of available information sources. There is a clean separation between the declarative source description and the actual details of interacting with an information source. The system uses the source descriptions to prune efficiently the set of information sources for a given query and to generate executable query plans. Specifically, we make the following contributions. First, we present a practical mechanism to describe declaratively the *contents* and *query capabilities* of information sources. In particular, the contents of the sources are described as *queries* over a set of relations and classes. Consequently, it is possible to model the fine-grained distinctions between the contents of different sources, and it is easy to add and delete sources. Modeling the query capabilities of information sources is crucial in order to interact with many existing sources. Second, we describe an efficient algorithm that uses the source descriptions to create *query plans* that can access several information sources to answer a query. The algorithm prunes the sources that are accessed to answer the query, and considers the capabilities of the different sources. Finally, we describe experiments that show that our query planning algorithm will scale up as the number of information sources increases. The experiments show the performance of our query planning algorithm using 100 information sources.

There are several important issues in building a system that provides a uniform interface to multiple information sources, that are not discussed here. In particular, the issue of deciding that two constants in two different information sources refer to the *same* object in the world (e.g., the same person appearing in two different information sources). Briefly, our implementation tries first to find unique identifiers for each constant (e.g., social security number of a person). When it cannot find such identifiers it uses heuristic *correspondence functions* as in the Remote-Exchange system [FHM94].

2 Data Model

We use the relational model, augmented with certain object-oriented features that are useful for describing and reasoning about the contents of information sources. The data model includes (1) *relations* of any arity, (2) *classes* and a class hierarchy. There is a partial order $\prec$ such that $C \prec D$ whenever class C is a subclass of class D, and (3) a set of *attributes* associated with each class. A class also inherits attributes from its superclasses. Attributes may be *single-valued* or *multi-valued.*

Relations contain tuples while classes contain objects. Each object has a unique identifier. The attribute values of a relation or a class can be either atomic values (strings or integers) or object identifiers. An object may belong to more than one class (even if the classes are not related via $\prec$). It is possible to declare a pair of classes to be *disjoint*, meaning that no object can belong to both classes.

In order to be able to treat relations and classes uniformly, we associate a unary relation with each class and a binary relation with each attribute. The contents of these relations are as follows (we use the same name for the class and its associated relation):

- For class C, $\langle X \rangle \in C$ whenever x is the identifier of an object o and C is one of the classes of o.
- For attribute A on class C, $\langle X, Y \rangle \in A$ whenever $\langle X \rangle \in C$ and $x.A = y$ (y is called the *A-filler* of x).

For single-valued attributes we often use $A(x)$ to denote the only value for which $A(x, y)$ can hold. In order that these relations fully capture the semantics of the class hierarchy, we also enforce certain integrity constraints. These constraints take the form of *inclusion dependencies* and *functional dependencies.* In particular:

- Whenever $C \prec D$ when C and D are viewed as classes, the inclusion dependency $C \subseteq D$ holds when C and D are viewed as relations.
- For each auxiliary relation $A(X, Y)$ corresponding to a single-valued attribute A, we have the functional dependency $A : X \rightarrow Y$.
- For each pair of disjoint classes C and D, $C \cap D = \emptyset$ holds when C and D are viewed as relations.

Table 1 shows the classes and attributes we use throughout the paper.

The World View: In the Information Manifold, the user poses queries in terms of a *world view* which is a collection of virtual relations and classes. Thus, the world view is like a schema. We use the term world view instead of schema to emphasize the fact that no data is actually stored in the relations and classes of the world view.[1] It serves as the schema against which the user poses queries (thereby freeing the user from having to interact with each source schema individually), and it is used for describing the contents of the information sources.

Example 2.1 The world view we use throughout this paper consists of the classes in Table 1 (all the attributes of which are single-valued) and the relation *ProductReview*(*Model*, *Year*, *Review*). □

In this paper, a *query* is a conjunctive query over the world-view relations and the built-in predicates $<$, $\leq$. We require the queries to be range-restricted.

Example 2.2 The following query asks for models, prices, and reviews of sportscars for sale that were manufactured no earlier than 1992 (query Q of Example 1.1):

$$\begin{aligned} q(m,p,r) \quad \leftarrow \quad & \mathit{CarForSale}(c),\ \mathit{Category}(c, \mathsf{sportscar}), \\ & \mathit{Year}(c,y),\ y \geq \mathsf{1992},\ \mathit{Price}(c,p), \\ & \mathit{Model}(c,m),\ \mathit{ProductReview}(m,y,r) \end{aligned}$$

□

We use this font to denote constants, lowercase letters for variable names, and uppercase letters with bars to denote tuples of variables and constants. Formally, a query is of the form:

$$Q(\overline{X}) \leftarrow R_1(\overline{Z}_1),\ \ldots,\ R_n(\overline{Z}_n),\ C_Q$$

[1] However we do not mean to imply that the world view is a schema for *all* domains.

Class	Subclass of	Attributes	Disjoint from
Product		*Model*	*Person*
Automobile	*Product*	*Model*, *Year*, *Category*	*Stereo*
Motorcycle	*Automobile*	*Model*, *Year*	*Car*
Car	*Automobile*	*Model*, *Year*, *Category*	*Motorcycle*
NewCar	*Car*	*Model*, *Year*, *Category*	*UsedCar*
UsedCar	*Car*	*Model*, *Year*, *Category*	*NewCar*
CarForSale	*Car*	*Model*, *Year*, *Category*, *Price*, *SellerContact*	

Table 1: A class hierarchy. The classes *Person* and *Stereo* are not shown.

where: (1) $R_1, \ldots, R_n$ are relations in the world view, (2) C_Q is a conjunction of *order subgoals* of the form $u \theta v$, where $\theta \in \{<, >, \leq, \geq\}$ and $u, v \in \bigcup_{1 \leq i \leq n} \overline{Z}_i$, and (3) $\overline{X} \subseteq \bigcup_{1 \leq i \leq n} \overline{Z}_i$.

3 Describing Information Sources

Queries are posed to the system in terms of the world view. However, the data to answer these queries is actually stored in external information sources. Therefore, to answer a query, we need descriptions that relate the contents of each information source to the classes, attributes and relations in the world view. Furthermore, since sources may not be able to answer arbitrary queries about their contents, we need to describe the *capabilities* of the information sources in order to create plans that can actually be executed.

3.1 Contents of Information Sources

There are several desiderata for descriptions of the contents of information sources. First, since the number of information sources is large and frequently changing, we should be able to add new information sources without changing the world view, and without affecting the descriptions of other information sources. Second, since many sources contain closely related information, the descriptions should be able to model fine-grained differences between their contents, so that the set of sources relevant to a query can be determined as "tightly" as possible.

We model the contents of an information source as tuples in one or more relations. The key to the flexibility in our source descriptions is that we describe relations in the sources as *queries* over the world-view relations and the comparison predicates. Formally, each source is modeled as containing tuples of a relation (or several relations) which we call *source relations*. The names of the source relations are disjoint from the names of the world view relations. For each source relation, we specify a conjunctive query over the world view relations that describes the conditions the tuples in the relation must satisfy. Note that the source need not contain *all* the tuples that satisfy the query; for example, no database of cars for sale contains all cars for sale. We emphasize this incompleteness by using the connective $\subseteq$ to relate the head and body of the description instead of the conventional $\leftarrow$ used in queries. Figure 2 shows the content descriptions corresponding to the informal descriptions in Figure 1.

It should be emphasized that the features of our data model (the class hierarchy, disjointness of classes and built-in predicates) and the fact that we describe contents as queries enables us to describe very tightly the contents of the sources, and therefore to be able to prune the sources relevant to a given query. Furthermore, adding sources does not affect the descriptions of other information sources.

3.2 Capabilities of Information Sources

The content description tells us *what* is in an information source, but it does not tell us *which queries* the source can answer about its contents. Many information sources permit only a subset of the possible queries on their databases. For example, the used car database in Example 1.1 requires either the category or model of the car as an input. When generating query plans it is important to adhere to the capabilities of the information sources, and exploit them as much as possible. In Example 1.1, the query plan involving sources 2 and 5 was different from the plan involving sources 1 and 5 because source 1 was able to perform the selection on the year of the car.

We describe the capabilities of an information source using *capability records*. Capability records are meant to capture the two kinds of capabilities encountered most often in practice: the ability of sources to apply a (perhaps limited) number of selections, and the limited forms of variable bindings that an information source can accept. The capability records specify which inputs can be given to the source, the minimum and maximum number of inputs allowed, the possible outputs of the source and the selections the source can apply. Sources with capabilities to perform arbitrary relational operators (e.g., full fledged databases) are

Source 1: Used cars for sale. **Contents:** $V_1(c) \subseteq CarForSale(c),\ UsedCar(c)$ **Capabilities:** $(\{Model(c), Category(c)\}, \{Model(c), Category(c), Year(c), Price(c), SellerContact(c)\}, \{Year(c), Price(c)\}, 1, 4)$
Source 2: Luxury cars for sale. All cars in this database are priced above \$20,000 **Contents:** $V_2(c) \subseteq CarForSale(c),\ Price(c,p),\ p \geq 20000$ **Capabilities:** $(\{Category(c)\}, \{Model(c), Category(c), Year(c), Price(c), SellerContact(c)\}, \{Price(c)\}, 1, 3)$
Source 3: Vintage cars for sale (cars manufactured before 1950). **Contents:** $V_3(c) \subseteq CarForSale(c),\ Year(c,y),\ y \leq 1950$ **Capabilities:** $(\{Model(c)\}, \{Model(c), Category(c), Year(c), Price(c), SellerContact(c)\}, \{Year(c)\}, 1, 2)$
Source 4: Motorcycles for sale. **Contents:** $V_4(c) \subseteq Motorcycle(c)$ **Capabilities:** $(\{Model(c)\}, \{Model(c), Year(c), Price(c), SellerContact(c)\}, \{Price(c)\}, 1, 2)$
Source 5: Car reviews database. Contains reviews for cars manufactured after 1990. **Contents:** $V_5(m,y,r) \subseteq Car(c),\ Model(c,m),\ Year(c,y),\ ProductReview(m,y,r)$ **Capabilities:** $(\{m,y\}, \{m,y,r\}, \{\}, 2, 2)$

Figure 2: Source descriptions for the sources in Figure 1

considered in [LRU96].

Formally, a capability record specifies which *parameters* can be given to the source. A *parameter* of a source relation $R(\overline{X})$ is either a variable $x \in \overline{X}$ or $A(x)$ where A is an attribute name and $x \in \overline{X}$. With every source relation we associate exactly one capability record of the form $(S_{in}, S_{out}, S_{sel}, min, max)$, where S_{in}, S_{out} and S_{sel} are sets of parameters of R, and min and max are integers. Every variable in $\overline{X}$ must appear in a parameter either S_{in} or S_{out}.

The meaning of the capability description is the following. In order to obtain a tuple of R from the information source, the information source must be given bindings for at least min elements of S_{in}. The elements in S_{out} are the parameters that can be *returned* from the information source. The elements of S_{sel}, which must be a subset of $S_{in} \cup S_{out}$, are parameters on which the source can apply selections of the form $\alpha\, op\, c$, where c is a constant and $op \in \{\leq, <, \neq, =\}$. Given a source relation R, providing the information source with the values $a_1, \ldots, a_n$ for the elements $\alpha_1, \ldots, \alpha_n$ in S_{in}, asking for the values of $\beta_1, \ldots, \beta_l$ in S_{out}, and passing the selections $\gamma_1, \ldots, \gamma_k$ to the source will produce the tuples $(Y_1, \ldots, Y_l)$ that satisfy the following conjunction:

$$R'(Y_1, \ldots, Y_l) : -R(X_1, \ldots, X_m), \alpha_1 = a_1, \ \ldots, \\ \alpha_n = a_n, \beta_1 = Y_1, \ \ldots, \beta_l = Y_l, \gamma_1, \ \ldots, \gamma_k.$$

Given a content description of the form $R \subseteq Q_R$ and input/output specifications as described above, the following is called the *augmented description* of R w.r.t. the input/output specifications:

$$R'(Y_1, \ldots, Y_l) \quad \subseteq \quad Q_R, \alpha_1 = a_1, \ \ldots, \alpha_n = a_n, \\ \beta_1 = Y_1, \ \ldots, \beta_l = Y_l, \gamma_1, \ \ldots, \gamma_k$$

In our query-planning algorithm we use a specific *canonical* augmented description of R in which the inputs include all of S_{in}, the outputs include all of S_{out} and there are no selections (note that this does not mean that our query plans necessarily provide all the inputs and extract all the outputs from a source). Figure 2 lists the capability records describing the information sources in our example.

3.3 Query Plans

A query plan is a sequence of accesses to information sources interspersed with local processing operations. A query plan must combine information from various sources in a way that guarantees semantically correct answers, and must adhere to the capabilities of the information sources. We explain these notions below. Given a query Q of the form

$$Q(\overline{X}) \leftarrow R_1(\overline{Z}_1), \ \ldots, R_n(\overline{Z}_n), C_Q$$

a plan to answer it consists of a set of *conjunctive plans.* Conjunctive plans are like conjunctive queries except that we also specify the inputs and outputs to every subgoal. An *executable* conjunctive plan is of the form:

$$P : Q(\overline{X}) \leftarrow V_1(\overline{U}_1)\ (in_1, out_1, sel_1)\ , \ \ldots, \\ V_m(\overline{U}_m)\ (in_m, out_m, sel_m)\ ,\ C_P.$$

Each of the V_i's is a source relation corresponding toan information source. The elements of the sets in_i are of the form $p_1 : p_2$, where p_1 is one of the parameters in the set S_{in} of the information source of V_i, and p_2 is either a value appearing in the query, or a parameter that appears in $out_1 \cup \ldots \cup out_{i-1}$. The set out_i is a subset of S_{out} in the capability record of the information source V_i, and sel_i is a set of selections to be passed to the information source. Finally, the cardinalities of in_i, out_i and sel_i must be consistent with the capability record of the information source of V_i. C_P is a set of selections that are applied locally by the query executor.

To define the semantic correctness of a conjunctive plan, we consider the canonical augmented content descriptions of the information sources. Recall that given the input and output specifications, each information source is modeled as containing a subset of the relation defined by a conjunctive query Q_i. Therefore, we can consider the *expansion* of the plan P as the query P' obtained by expanding the definitions of the subgoals V_i. Formally P' is obtained by replacing the subgoal $V_i(\overline{U}_i)$ by the body of the query Q_i after unifying the head variables of Q_i with $\overline{U}_i$. The conjunctive plan P is said to be *semantically correct* if P' is contained in Q, i.e., for any extension of the world view relations that satisfies the integrity constraints, the answer to P' would be a subset of Q.

Example 3.1 Consider our query asking for sports cars manufactured in 1992 or later:

$$\begin{aligned} q(m,p,r) \quad \leftarrow \quad & \mathit{CarForSale}(c),\ \mathit{Category}(c, \mathsf{sportscar}), \\ & \mathit{Year}(c,y),\ y \geq 1992,\ \mathit{Price}(c,p), \\ & \mathit{Model}(c,m),\ \mathit{ProductReview}(m,y,r) \end{aligned}$$

The following is a semantically correct plan for the query:

$$\begin{aligned} & P_1 :\ Q(m,p,r) \leftarrow \\ & V_1(c)\ (\{Category(c) : \mathsf{sportscar}\}, \{Price(c), Model(c)\}, \\ & \qquad \{Year(c) \geq 1992\}), \\ & V_5(m,y,r)\ (\{m : Model(c), y : Year(c)\}, \{r\}, \{\}). \end{aligned}$$

To see why, we can verify that the expansion query P_1' of P_1 obtained by unfolding the augmented descriptions of V_1 and V_5 is contained in the original query:

$$\begin{aligned} & P_1' :\ Q(m,p,r) \leftarrow \mathit{CarForSale}(c),\ \mathit{UsedCar}(c), \\ & \quad \mathit{Model}(c,m),\ \mathit{Category}(c,t),\ t = \mathsf{sportscar},\ \mathit{Year}(c,y), \\ & \quad \mathit{Price}(c,p),\ \mathit{ProductReview}(m,y,r),\ y \geq 1992. \end{aligned}$$

□

Semantically correct plans required only that P' be *contained* in Q and not *equivalent* to Q for the following reasons. First, even if P' were equivalent to Q, the answer obtained by executing P may not be complete because the sources may be incomplete. Second, conjunctive plans that produce only a subset of the answer are also useful. For example, if we are searching for sports cars manufactured after 1992, and we have an information source with cars manufactured after 1994, we would still want to query it. We define the set of answers to the query Q as all the tuples that can be obtained by some *executable* and *semantically correct* conjunctive plan for Q.

Interface Programs: Describing source query capabilities in terms of capability records provides a clean separation between query planning and the actual details of interacting with each information source. These details are encoded in *interface programs.* Logically, there is one interface program that accepts any query template available at the source and returns the appropriate answer. The interface program accepts the bound parameters to a query corresponding to the template, interacts with the information source, and produces a relation corresponding to the free parameters in the query template.

4 Algorithms for Answering Queries

Our algorithm for generating executable query plans has two steps. In the first we generate semantically correct conjunctive plans, and in the second we try to order the conjuncts of the plan to ensure that they are executable. The first step of the algorithm is described in detail in a companion paper [LRO96]. Here we only describe the aspects of this step that are needed to understand the second step and the experimental results.

A semantically correct plan guarantees that the answers produced will actually be answers to the query. Finding a semantically correct query plan amounts to finding a conjunctive query Q' that uses only the source relations and is *contained* in the given query Q. Therefore, our problem is closely related to the problem of answering queries using views [LMSS95, RSU95, YL87, CKPS95], where the source relations play the role of the views. However, the problem of answering queries using views is known to be NP-complete in [LMSS95], even for conjunctive queries without built-in atoms. The main source of complexity is the fact that there are an exponential number of candidate rewritings that need to be considered. This is especially significant in our context because that algorithm would be exponential in the number of information sources. Our algorithm drastically reduces the number of candidate rewritings considered by proceeding as follows. First, the algorithm computes a *bucket* for each subgoal in the query, each containing the information sources from which tuples of that subgoal can be obtained. In the second step, we consider all the possible combinations of information sources, one from each bucket, and check whether it is a semantically correct plan, or can be made semantically correct if additional built-in atoms are added to the plan. As we see in Section 5, the first step, considerably reduces the number of possibilities considered in the second step. The details of the first step are given in Figure 3.

Example 4.1 We illustrate the algorithm on our example. Consider our query asking for sports cars manufactured no sooner than 1992:

$$q(m_1, p_1, r_1) \leftarrow \mathit{CarForSale}(c_1),\ \mathit{Category}(c_1, \mathsf{sportscar}),\ \mathit{Year}(c_1, y_1),\ y \geq 1992,\ \mathit{Price}(c_1, p_1),\ \mathit{Model}(c_1, m_1),\ \mathit{ProductReview}(m_1, y_1, r_1).$$

and consider what happens when algorithm **Create-Buckets** looks at Source 1 and the first subgoal of our query $\mathit{CarForSale}(c_1)$. The canonical augmentation of the content description of Source 1 is:

$$V_1'(m, t, y, p, s) \subseteq \mathit{CarForSale}(c),\ \mathit{UsedCar}(c),\ \mathit{Price}(c, p),\ \mathit{Category}(c, t),\ \mathit{Year}(c, y),\ \mathit{Model}(c, m),\ \mathit{SellerContact}(c, s)$$

therefore, the algorithm will find the mapping $c \rightarrow c_1$ and check whether the following is satisfiable:[2]

$$\mathit{CarForSale}(c_1),\ \mathit{Category}(c_1, \mathsf{sportscar}),\ \mathit{Year}(c_1, y_1),\ y_1 \geq 1992,\ \mathit{Price}(c_1, p_1),\ \mathit{Model}(c_1, m_1),\ \mathit{ProductReview}(m_1, y_1, r_1),\ \mathit{UsedCar}(c_1),\ \mathit{SellerContact}(c_1, s)$$

Since the classes *CarForSale* and *UsedCar* are not disjoint, the conjunction is satisfiable and Source 1 is added to $bucket_1$. In a similar fashion, Source 2 is added to $bucket_2$. Source 3 does not get added because $(y \leq 1950, y \geq 1992)$ is not satisfiable, and Source 4 does not get added because *CarForSale* and *Motorcycle* are disjoint classes. Source 5 is the only source in the bucket of the subgoal $\mathit{ProductReview}(m_1, y_1, r_1)$.

The algorithm will now check the cartesian product of the buckets. For example, as shown in Example 3.1, the plan resulting from combining sources 1 and 5 is contained in the original query, and is therefore a semantically correct plan. □

4.1 Finding an Executable Ordering

In the second step of creating query plans we consider the semantically correct plans and try to order the subgoals in such a way that the plan will be executable, i.e., will adhere to the capability requirements of the information sources. Figure 4 describes an algorithm that given a semantically correct plan, finds an ordering on its subgoals that is executable, if such an ordering exists. The algorithm proceeds by maintaining a list of available parameters, and at every point adds to the ordering any subgoal whose input requirements are satisfied. Finally, the algorithm pushes as many selections as possible to the sources.

Example 4.2 Consider the semantically correct plan for answering our sportscar query:

$$P_1 :\ Q(m, p, r) \leftarrow V_1(c),\ V_5(m, y, r),\ Model(c, m),\ Year(c, y),\ Category(c, \mathsf{sportscar}),\ y \geq 1992.$$

The bindings available in the query are $\{Category(c)\}$. Therefore, the input requirements of $V_1(c)$ are satisfied and so it is put first. The outputs of $V_1(c)$ are $\{Model(c),\ Price(c),\ Year(c),\ SellerContact(c)\}$, therefore $BindAvail_1 = \{Category(c),\ Model(c),\ Price(c),\ Year(c),\ SellerContact(c)\}$, and so the input requirements of $V_5(m, y, r)$ are satisfied. Since the second information source provides the review, the ordering is executable. Finally, we add $y \geq 1992$ to the selections of the first source. We remove $SellerContact(c)$ from the outputs of the first subgoal because it is not needed anywhere in the query. □

The following theorem shows that our algorithm will find an ordering of a plan whenever an executable ordering exists, and will do so in polynomial time. The proof is omitted because of space limitations.

Theorem 4.1: *Let Q' be a semantically correct plan. If there is an ordering of the subgoals of Q' that results in an executable plan, then procedure* **create-executable-plan** *will find it. The running time of the procedure is polynomial in the size of Q'.* □

Algorithm **create-executable-plan** is a generalization of an algorithm by Morris [Mor88] for ordering subgoals in the presence of binding constraints. The key difference is that our capability records encode a *set* of possible binding patterns for each subgoal, and we find an ordering that chooses one pattern from every such set. Furthermore, our binding patterns involve not only variables occurring in the query, but also attributes on them, and also the possibility of pushing selections on parameters.

Our descriptions allow only one capability record for every source relation. This restriction essentially means that the parameters that can be obtained from the source do not depend on how we chose to satisfy the input requirements of the source. In practice, we have found this to be sufficient to describe the sources we encountered. Conceivably, there may be situations in which it will not suffice, and the output set depends

[2]Note that some variables (e.g., y_1 and y) get equated because of the single-valued attributes.

Algorithm CreateBuckets($\mathcal{V}$,Q)
Inputs: $\mathcal{V}$ is a set of content descriptions, and Q is a conjunctive query of the form $Q : Q(\bar{X}) \leftarrow R_1(\bar{X}_1),\ \ldots,\ R_m(\bar{X}_m),\ C_Q$.

Set $Bucket_i$ to $\emptyset$ for $1 \le i \le m$.
For $i = 1, \ldots, m$ do:
    For each $V \in \mathcal{V}$
        Let V be of the form: $V(\vec{Y}) \subseteq S_1(\bar{Y}_1),\ \ldots,\ S_n(\bar{Y}_n),\ C_V$
        For $j = 1, \ldots, n$ do
            If $R_i = S_j$ or R_i and S_j are nondisjoint classes
                Let ψ be the mapping defined on the variables of V as follows:
                If y is the k'th variable in $\bar{Y}_j$ and $y \in \bar{Y}$
                    then $\psi(y) = x_k$, where x_k is the k'th variable in $\bar{X}_i$.
                    else $\psi(y)$ is a new variable that does not appear in Q or V.
                Let Q' be the 0-ary query:
                $Q' \leftarrow R_1(\bar{X}_1),\ \ldots,\ R_m(\bar{X}_m),\ C_Q,\ S_1(\psi(\bar{Y}_1)),\ \ldots,\ S_n(\psi(\bar{Y}_n)),\ \psi(C_V)$
                If $Satisfiable(Q')$ then add $\psi(V)$ to $Bucket_i$.
End.

Figure 3: Algorithm to create the relevant buckets for each query subgoal. The procedure $Satisfiable(Q')$ tests whether a query Q' is satisfiable. It tests that the conjunction of built-in atoms is satisfiable, and that there are no two subgoals $C(x)$ and $D(x)$ where C and D are disjoint classes. We assume that the source descriptions are given to the algorithm in their canonical augmented form.

on which set of input parameters we used. The following theorem shows that in such a case, the problem of determining whether there exists an executable ordering for a plan is intractable, and therefore the choice we made also has important computational advantages.

Theorem 4.2 : *If every source relation in the content descriptions of information sources could have more than one capability record of the form* $(S_{in}, S_{out}, S_{sel}, min, max)$, *then the problem of determining whether a semantically correct plan can have an executable ordering is NP-complete.* □

5 Implementation and Experiments

The Information Manifold system, whose architecture is shown in Figure 5 uses the techniques described in the previous sections to provide a uniform query interface to 100 structured information sources on the WWW. When a query is posed, the system uses the descriptions of information sources to compute executable query plans. An important aspect of the system is that it provides a *stream* of answers to the user, and therefore tries to minimize the time taken to begin and sustain the stream, as opposed to minimizing the time taken to provide *all* the answers to the query. Minimizing the time to the early tuples is important because the user is likely to find a satisfactory answer before all answers are exhausted.

In order to experimentally evaluate our algorithms, we selected a set of queries and studied how various parameters varied as we increased the number of information sources available to the system. Here we illustrate our results using three representative queries:

1. *Find titles and years of movies featuring Tom Hanks.*
2. *Find titles and reviews of movies featuring Tom Hanks.*
3. *Find telephone number(s) for Alaska Airlines.*

For each query, we varied the number of information sources available to the system from 20 to 100 and measured various parameters. The results are shown in Table 2. Our experiments were run on a SGI Challenge 150MHz computer. *Maximum bucket size* is the number of sources in the largest bucket created using Algorithm **CreateBuckets**. *Plans enumerated* is the number of candidate plans enumerated in the second stage of the query planning algorithm, while *plans generated* is the total number of semantically correct and executable query plans actually generated for a given query. Table 2 also gives the total time taken to generate all query plans and the time per plan.

We note that the number of information sources relevant to a query generally increases with the total number of sources available. However, Algorithm **CreateBuckets** is extremely effective in prun-

procedure create-executable-plan(Q')
/* **Input:** Q' is a semantically correct conjunctive plan whose non-interpreted subgoals are $U_1, \ldots, U_n$. */
The capability record of the information source of U_i is $(in_i, out_i, sel_i, min_i, max_i)$.
We assume all bindings in Q' are given explicitly using the $=$ relation as a conjunct.

Output: an executable query plan P', which is an ordering $V_1, \ldots, V_n$ of $U_1, \ldots, U_n$,
and triplets $(V^i_{in}, V^i_{out}, V^i_{sel})$ specifying the inputs and outputs of the conjuncts.
$C_{P'}$ is the set of selections that will be applied locally. */

$QueryBindings$ = The set of variables in Q' bound by values in the query.
Q_{out} = The head variables of Q'.
$QuerySelections$ = The set of variables in Q' for which the query contains a selection.
$BindAvail_0 = QueryBindings$.
for $i = 1, \ldots, n$
The i'th subgoal in the ordering, V_i, is any subgoal U_j of Q' that was not chosen earlier and
at least min_j of the parameters in in_j are in $BindAvail_{i-1}$.
if there is no such subgoal, **return** `plan not executable`, **else**
$BindAvail_i = BindAvail_{i-1} \cup out_j$.
V^i_{in} = A minimal set of parameters in $BindAvail_{i-1}$ that satisfied the input requirement of U_j.
V^i_{out} = All the parameters in out_j.
end for
if $Q_{out} \not\subseteq BindAvail_n$ **return** `plan not executable`.
for $i = 1, \ldots n$
Remove any element from V^i_{out} that is not needed as an input to a subsequent subgoal or for Q_{out}.
Add to as many parameters as possible from $QuerySelections \cup BindAvail_{i-1}$ to V^i_{in}
and selections using these parameters to V^i_{sel} such that the cardinality of $V^i_{in} \cup V^i_{sel}$ does
not exceed the input capacity of its source.
$C_{P'}$ includes all the built-in atoms in Q' that are not in one of the V^i_{sel}'s.
end create-executable-plan.

Figure 4: An algorithm for computing an executable ordering of a semantically correct plan. We assume that any pair of variables that are forced to be equal because of the functional dependencies have already been equated in the input.

ing away irrelevant sources. The effectiveness of the pruning is measured in terms of the reduction in the number of candidate plans that are enumerated when creating semantically correct plans. If there were no pruning (as suggested by the nondeterministic algorithm in [LMSS95]), we would have to enumerate $O(n^{|Q|})$ plans for query Q, where n is the total number of information sources and $|Q|$ is the number of subgoals in Q. For example, with 100 sources, we would have to enumerate more than 1 million plans for Query 1. However, the number of plans we actually enumerate is only 26 (a function of the product of the bucket sizes).

Observe also that although Query 1 and Query 2 both ask about movies, the number of sources relevant to Query 2 is more than the number of sources relevant to Query 1 (7 versus 2 with 100 sources, for example). This difference is due to our ability to model fine-grained distinctions among movie sources, which enables us to prune away certain sources for Query 1 that are relevant to Query 2.

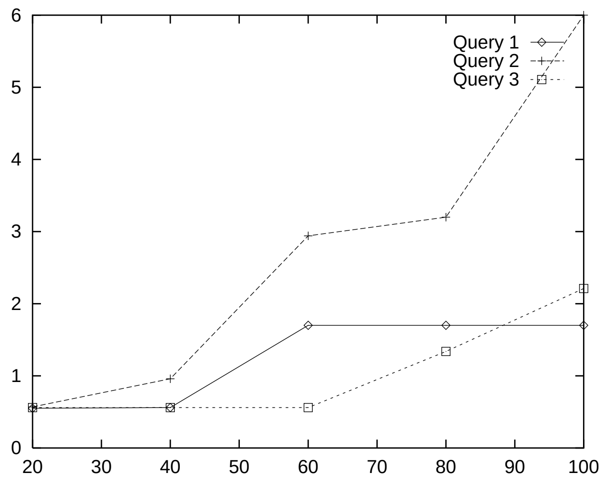

Figure 6: Total query planning time in seconds versus number of information sources.

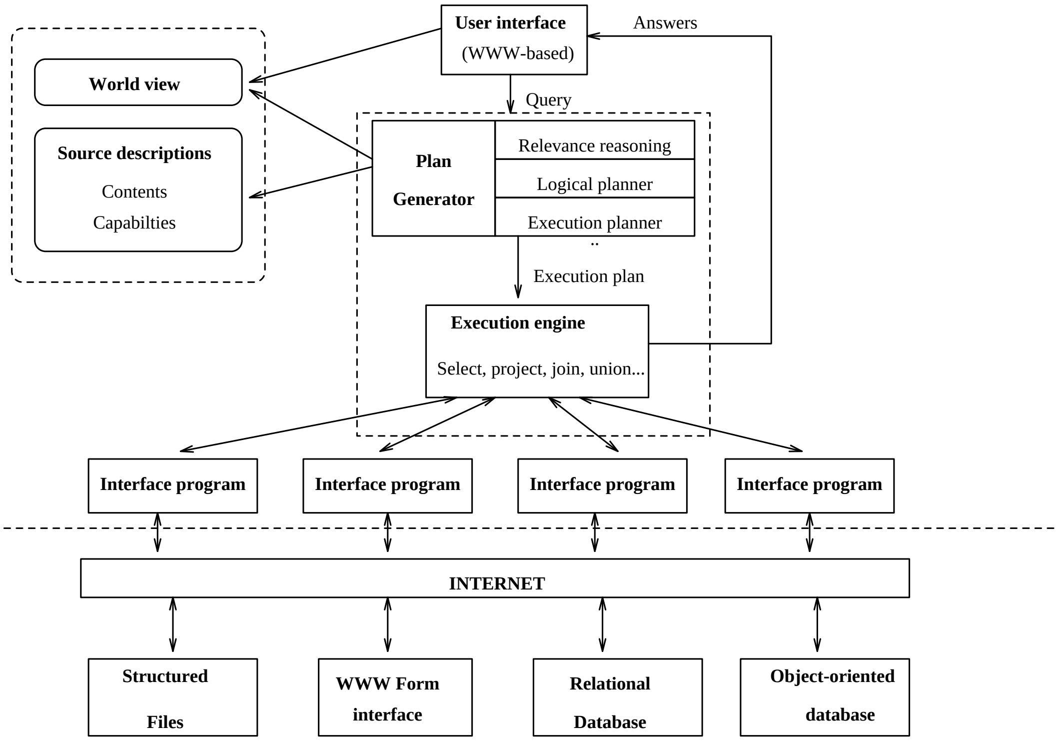

Figure 5: Architecture of the Information Manifold

Figure 6 plots the total time to generate all query plans for each query against the number of information sources available to the system. It is seen that the overall time generally increases with the number of information sources, but not exponentially. Due to the effective pruning, the time for plan generation is more a function of the number of *relevant* information sources than of *total* number of information sources.

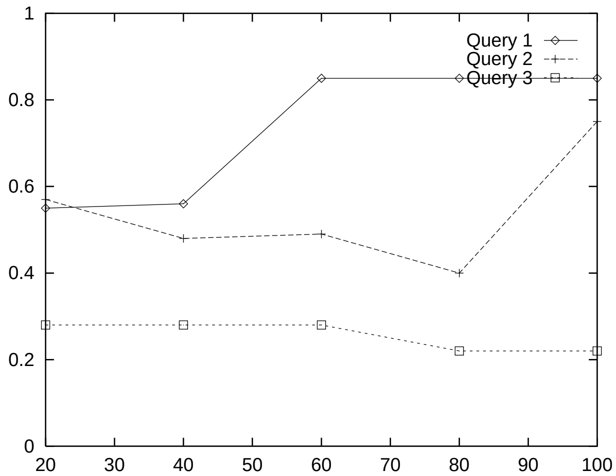

Figure 7: Average time per plan in seconds versus number of information sources.

The total time for query planning is not a very good indicator of system response time. In the Information Manifold, each query plan is executed a soon as it is generated, in parallel with further planning and executing other plans. Thus, a better measure of response time is the average time to generate one query plan. We plot the average time per plan against the number of information sources in Figure 7. In contrast to the total query planning time, we observe that the average plan time does not always increase with the number of information sources, nor does it increase as rapidly. This effect is due to the fact that increasing the number of sources available generally also increases the number of possible query plans. Finally, we observe that the average time per plan is within a tight range of less than 1 second for the queries we study, even when the number of information sources is large. This time is to be contrasted with the time taken to execute a query plan, which typically involves going over a network.

6 Related Work

Several systems (e.g., TSIMMIS [PGGMU95], HERMES [ACPS96], CARNOT [CHS91], DISCO [FRV95], Nomenclator [OM93]) for integrating multiple information sources are being built on the notion of a *mediator*. The key aspect distinguishing Information Manifold from the other systems is its generality, i.e., that it provides a source independent, query independent mediator. Instead of being tailored to specific infor-

Query	Number of sources	Max. bucket size	Plans enumerated	Plans generated	Time per plan (sec.)	Total time (sec.)
1	20	1	7	1	0.55	0.55
	40	1	7	1	0.56	0.56
	60	2	26	2	0.85	1.70
	80	2	26	2	0.85	1.70
	100	2	26	2	0.85	1.70
2	20	2	7	1	0.57	0.57
	40	3	11	2	0.48	0.96
	60	5	35	6	0.49	2.95
	80	6	44	8	0.40	3.20
	100	7	72	8	0.75	6.00
3	20	2	8	2	0.28	0.56
	40	2	8	2	0.28	0.56
	60	2	8	2	0.28	0.56
	80	6	49	6	0.22	1.32
	100	10	120	10	0.22	2.20

Table 2: Query planning statistics for queries 1, 2, and 3 as the number of available information sources is varied between 20 and 100.

mation sources and/or specific queries on these information sources, the input to Information Manifold is a set of descriptions of the contents and capabilities of the sources. Given a query, the Information Manifold will consider the descriptions and the query, and will create a plan for answering the query using the sources. Consequently, we do not have to build a new mediator for different queries or information sources. For example, the Nomenclator system incorporates multiple CCSO, X.500 and relational name servers. Source descriptions are given as equality selections on a *single* relation, and queries can only reference one relation.

The SIMS system [ACHK94] also describes information sources independently of the queries that are subsequently asked on them. The descriptions in the Information Manifold are richer than those in SIMS because they allow relations of arbitrary arity, and in particular allow us to express the fact that an information source contains a conjunctive view over world-view relations (either classes, roles or relations of higher arity). SIMS does not consider capability descriptions of the sources. SIMS, as well as the Internet Softbot [EW94] use Artificial Intelligence planning techniques for determining the relevant information sources and creating a query plan. These approaches do not provide the guarantees of ours, that is that we find *all* and *only* the relevant sources.

In [LSK95] a language for describing information sources that was less expressive than the one we describe here was proposed. The language did not consider the capability descriptions, and the algorithms described for finding relevant information sources did not deal with the case where source descriptions are given as queries on the world-view relations. Consequently, only a limited range of information sources could be incorporated. Practical algorithms and evaluation were also not discussed there.

A related line of work is on Web query languages. For example, W3QS [KS95] is a system for specifying high-level queries over unstructured information sources. This system enables the user to specify in the query patterns of nodes on the web and properties of such nodes. W3QS is a useful tool that enables a lot of otherwise manually done search to be done by a search engine, but it does not make use of contents of structured sources, and combine information from multiple sources.

7 Conclusions and Future Work

We described the query planning algorithms used in Information Manifold, a novel system that provides a uniform query interface to distributed structured information sources. The Information Manifold frees the user from having to interact with each information source separately, and to combine information from multiple sources. The techniques underlying the Information Manifold are applicable to sources on the WWW as well as other collections of information sources such as company-wide databases. The key aspect of our system is a mechanism for describing the contents and capabilities of the available information sources. This enables expressing fine-grained distinctions between the contents of different information sources, thereby enabling us to prune the sources that are irrelevant to a given query. A novel aspect of our system is that it describes the capabilities of information sources in addition to their contents, which is crucial in order to interact with remote sources. Our architecture and algorithms have been useful in practice, allowing us to describe many existing information encountered. The end result is the first system that provides a database-like interface to over 100 structured information sources on the WWW.

There are several important areas of research we are currently pursuing. One issue is how to obtain source descriptions. Currently, we are providing the source descriptions, but an active area of research is the design of tools for automating this process. We are also considering how to extend our source descriptions so that we will be able to infer that a source is relevant to a query with some degree of likelihood. For example, if we are searching for papers on database systems, and have access to a repository of papers on operating systems, we cannot completely ignore the repository, because we cannot state that these two fields are disjoint. However, we would like to access this repository only after we have accessed all other repositories that are closer to database systems.

Acknowledgments

We thank Marie-Christine Rousset, Arnie Rosenthal, Avi Silberschatz, Anthony Tomasic and Jeff Ullman for comments on earlier versions of this paper.

References

[ACHK94] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 1994.

[ACPS96] S. Adali, K. Candan, Y. Papakonstantinou, and V.S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proceedings of SIGMOD-96*, 1996.

[CHS91] C. Collet, M. N. Huhns, and W. Shen. Resource integration using a large knowledge base in carnot. *IEEE Computer*, 1991.

[CKPS95] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of ICDE-95*, 1995.

[EW94] Oren Etzioni and Dan Weld. A softbot-based interface to the internet. *CACM*, 37(7), 1994.

[FHM94] D. Fang, J. Hammer, and D. McLeod. The identification and resolution of semantic heterogeneity in multidatabase systems. In *Multidatabase Systems: An Advanced Solution for Global Information Sharing*. 1994.

[FRV95] Daniela Florescu, Louiqa Rashid, and Patrick Valduriez. Using heterogeneous equivalences for query rewriting in multidatabase systems. In *COOPIS '95*, 1995.

[KS95] David Konopnicki and Oded Shmueli. W3QS: A query system for the WWW. In *Proceedings VLDB*, 1995.

[LMSS95] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of ACM PODS*, 1995.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query answering algorithms for information agents. In *Proceedings of AAAI-96*, 1996.

[LRU96] A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external processors. In *Proceedings of ACM PODS*, 1996.

[LSK95] A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 5 (2), September 1995.

[Mor88] K. A. Morris. An algorithm for ordering subgoals in NAIL! In *Proceedings ACM PODS*, 1988.

[OM93] J. J. Ordille and B. P. Miller. Distributed active catalogs and meta-data caching in descriptive name services. In *Proceedings of the 13th International IEEE Conference on Distributed Computing Systems*, 1993.

[PGGMU95] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. Ullman. A query translation scheme for rapid implementation of wrappers. In *In proceedings of DOOD-95*, 1995.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proceedings of ACM PODS 1995*, 1995.

[YL87] H. Z. Yang and P. A. Larson. Query transformation for PSJ-queries. In *Proceedings of the 13th International VLDB Conference*, pages 245–254, 1987.