Querying Heterogeneous Information Sources Using Source Descriptions

Alon Y. Levy — AT&T Laboratories — 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 include commercial databases on product information, stock market information, real estate, automobiles, and entertainment. We would like to use the data stored in these databases to answer complex queries that go beyond keyword searches. We face the following challenges: (1) Several information sources store interrelated data, and any query-answering system must understand the relationships between their contents. (2) 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). (3) Since the number of sources is very large, effective techniques are needed to prune the set of information sources accessed to answer a query. (4) The details of interacting with each source vary greatly.

We describe the Information Manifold, an implemented system that provides uniform access to a heterogeneous collection of more than 100 information sources, many of them 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. 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. The query plans we generate can involve querying several information sources and combining their answers. 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.[1] 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

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

[1] See `http://www.intbc.com/sleuth/` for an index that focuses mostly on such sources.

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: A set of related information sources. These information sources are typical of those found on the World-Wide Web.

is very structured, and can be parsed and converted into a set of tuples or more complex data types (e.g., using techniques such as [ACM93, RU96]). 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) are based on keyword search, and much research has been devoted to efficient techniques for indexing large collections of documents (e.g., [GGMT94, BDMS94]). 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 the category of the car (sportscar or sedan), the price, the year of manufacture, the 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 4 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. Produce a set of $\langle Model, Price, ProductReview\rangle$ tuples.

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. Produce a set of $\langle Model, Price, ProductReview\rangle$ tuples.

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. $\square$

Some of the challenges involved in providing uniform access to a large collection of information sources are:

1. Several information 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 prunning sources.

2. 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.

This paper describes the Information Manifold (IM), a fully implemented system that provides uniform access to a heterogeneous collection of more than 100 information sources, many of them 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. The query plans we generate can involve querying several information sources and combining their answers. The contributions of this paper are the following:

1. 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.

2. 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. The problem of creating query plans is closely related to the problem of answering queries using views [YL87, CKPS95, LMSS95, RSU95, Qia95] which was shown to be computationally expensive. One of our contributions is an algorithm that is designed to scale up well in practice to a large number of information sources.

3. 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, most of which are on the WWW.

There are several issues to be addressed in providing a uniform interface to multiple information sources, many of which are not addressed in this paper. In particular, the goal of Information Manifold is to provide only a *query* interface, and not update or transaction facilities. As a consequence, we do not address issues such as consistency and transaction processing which are addressed by research on multidatabase systems. Issues of security and payment for information are also beyond the scope of this paper.

An important issue that is addressed in our system but we do not discuss in this paper is how we decide 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].

1.1 Related Work

The fundamental difference between our work and other work that attempts to provide access to collections of information sources is our focus on *describing* declaratively the contents of an information source (e.g., "used cars for sale priced over $20,000") and its query capabilities. Given a query, our algorithm uses the descriptions to generate plans to answer the query. Thus our approach is source-centric rather than query-centric. Other projects (e.g., TSIMMIS [CGMH+94], HERMES [SAB+95]) are query-centric: they choose a set of queries, and for each such query they provide a procedure to answer the query using the available sources. Given a new query, their algorithms answer it by trying to relate it to existing queries. Our approach has two main advantages. First, we are not restricted by which queries can be answered by the system. Second, it is easier to add or delete sources because we do not have to modify the query-specific procedures to accomodate the changes. A more detailed discussion of related work appears in Section 6.

1.2 Paper Organization

This paper is organized as follows. Section 2 describes the data model underlying the Information Manifold, and Section 3 formally describes the source descriptions and query plans. Section 4 presents our algorithm for pruning sources and generating query plans. In Section 5 we describe the implemented Information Manifold system and present our experimental results. Section 6 discusses related work, while Section 7 contains concluding remarks.

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:

- *Relations* of any arity.
- *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.
- 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 of a class. The contents of these relations are as follows (we use the convention that the relation associated with a class has the same name as the class, and similarly for attributes):

- 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 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.

Example 2.1 Table 1 shows some classes and their attributes. In this example, we have $Car \prec Automobile$ and $Automobile \prec Product$, among other such relationships. Since $Automobile \prec Product$, *Automobile* inherits the attribute *Model* from *Product*. Classes *NewCar* and *UsedCar* are declared to be disjoint, reflecting the fact that a car cannot be both new and used. However, class *CarForSale* is disjoint with neither *NewCar* or *UsedCar*. Disjointness information can also be inferred from the class hierarchy: *UsedCar* and *Motorcycle* are disjoint because $UsedCar \prec Car$ and *Car* is disjoint from *Motorcycle*. $\Box$

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.

2.1 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.[2] 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 (as we explain in Section 3).

Example 2.2 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*). This relation contains triples (M, Y, R) such that R is a review of a product of model M manufactured in year Y (for example, a product review in the Consumer Reports). □

2.2 Queries

In this paper, a *query* is a conjunctive query over the set of relations in the world view (i.e., select-project-join queries) We also allow the order relations $<, >, \leq, \geq$ to appear in queries, and we require the queries to be range-restricted.

Example 2.3 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):

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

We use this font to denote constants and lowercase letters for variable names. □

Formally, a query is of the form:

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

where:

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

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 $u, v \in \bigcup_{1 \leq i \leq n} \overline{Z}_i$.

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:

- 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 each time we add an information source, and without affecting the descriptions of other information sources.

- 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 should be able to develop efficient algorithms to determine the set of sources relevant to a query and to generate query plans that access these sources.

We model the contents of an information source as tuples in one or more relations, or objects in one or more classes. Two challenges arise in precisely describing contents of sources in terms of the world view:

1. When adding a new information source, it is often the case that the tuples in the source do not correspond directly to tuples in any one relation of the world view. For example, suppose our world view includes the relation $Teaches(Course, Teacher, Hour, Room)$, but the online course listing makes available only $(Course, Teacher)$ pairs. We could introduce a new relation corresponding to these pairs in the world view, but doing so means modifying the world view. Furthermore, that would require having many relations in the world view and expressing complex dependencies between them in order to capture the relationship between contents of different sources. Our solution is to describe the online course listing as containing tuples in the relation $\pi_{Course, Teacher}(Teaches)$.

2. Even when the objects or tuples in an information source may be thought of as belonging directly to a relation or class in the world view, we may wish to specify certain additional

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:** $(\{Model(c), Category(c)\}, \{Model(c), Category(c), Year(c), Price(c), SellerContact(c)\}, \{Price(c)\}, 1, 3)$
Source 3: Vintage cars for sale (cars manufactured before 1950). **Contents:** $V_3(c) \subseteq CarForSale(c),\ Year(c,y), y \leq 1950$ **Capabilities:** $(\{Model(c)\}, \{Model(c), Category(c), Year(c), Price(c), SellerContact(c)\}, \{Price(c)\}, 1, 2)$
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

constraints that these objects or tuples satisfy. For example, consider the vintage car information source from Example 1.1. Even though each object in the source belongs to class *CarForSale*, we would like to specify that all the cars in this source were manufactured before 1950; we saw how we could use this additional information to prune this source as irrelevant to the query in Example 1.1.

The solution to both these problems is to specify the tuples (or objects) in an information source in terms of a *query* over the relations in the world view. For example, we say that the online course listing source discussed above contains tuples in the relation *CourseList*(*Course*, *Teacher*), such that:

$$CourseList(course, teacher) \subseteq Teaches(course, teacher, hour, room)$$

We describe the vintage car source as containing tuples of relation $CarForSale(c)$ such that:

$$VintageCar(c) \subseteq CarForSale(c),\ Year(c,y), y \leq 1950$$

More 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 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. In Section 4 we show that we can effectively use the descriptions to create query plans. Finally, it should be noted that we do not claim that our data model *integrates* the relational and object oriented data model. It simply provides a mechanism necessary to *describe* sources using those data models, so that we can query them.

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. A conventional relational database can answer any relational query over its relations. However, information sources in general may permit only a subset of all relational queries over their relations. For example, we saw that the cars for sale database in Example 1.1 answers the query: Given a price range and a category of car, what cars of this category are available for sale within this price range? However, the source will not answer the query: List all cars in the database. Furthermore, when a source contains instances of a class, it may be able to answer queries only about a subset of the attributes of the class.

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, which are (1) 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 (also called *query templates* in [RSU95]). 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 are 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 either S_{in} or S_{out} (either the variable itself or an attribute on it). 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 a binding for at least min elements of S_{in}. If we provide the values $a_1, \ldots, a_n$ for the elements $\alpha_1, \ldots, \alpha_n$ in S_{in}, we will obtain all the tuples in the information source that satisfy $\alpha_1 = a_1, \ \ldots, \alpha_n = a_n$.

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:

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

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.

Example 3.1 The vintage-car information source can handle any query on relation $VintageCar(c)$ that bind at least one of *model* and *category* and can also handle range selections on *year* and *price*. It can return the model, year, price, category and seller contact information of the car. We can describe it using the capability record:

$\langle\{Model(c), Category(c)\}, \{Model(c), Category(c), Year(c), Price(c), SellerContact(c)\},$
$\{Year(c), Price(c)\}, 1, 4\rangle$

Figure 2 lists the capability records describing the information sources in our example. $\square$

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. Formally, a conjunctive plan is of the form:

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

Each of the V_i's is a source relation. The sets in_i and out_i are parameters on $\overline{U}_i$, and sel_i is set of selections applied to the parameters of $\overline{U}_i$. An element of in_i is of the form $p_1 : p_2$, where p_1 is one of the parameters that can be passed to the information source, and p_2 is is a parameter whose value either appears explicitly in the query or is in $out_1 \cup \ldots \cup out_{i-1}$. C_P is a set of selections that are applied locally by the query executor. A plan P is *executable* if the capabilities of the sources are satisfied, i.e., for every i, $1 \leq i \leq m$, (in_i, out_i, sel_i) is consistent with the capability record of the source V_i, and

- $in_i \subseteq Q_{in} \cup out_1 \cup \ldots \cup out_{i-1}$, where Q_{in} is the set of parameters available explicitly in the query.

To define the semantic correctness of a conjunctive plan, we consider the 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 tuples of a relation V_i defined by a conjunctive query S_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 S_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 the answer to Q.

Example 3.2 Consider our query asking for sports cars manufactured in 1992 or later:

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

The following is a semantically correct plan for the query:

$$\begin{aligned} P_1 : Q(m,p,r) \leftarrow V_1(c)\ & (\{Category(c) : \textsf{sportscar}\}, \\ & \{Price(c), Model(c)\}, \{Year(c) \geq 1992\ , Category(c) = \textsf{sportscar}\}), \\ V_2(m,y,r)\ & (\{m : Model(c), y : Year(c)\}, \{r\}, \{\}). \end{aligned}$$

To see why, consider the expansion query P_1' of P_1 obtained by unfolding the augmented descriptions of V_1 and V_2:

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

The query P_1' is contained in Q. To see why, suppose t is a tuple generated by P_1'; then t satisfies all the conjuncts in the body of P_1'. The body of P_1' contains all the conjuncts in the body of the query Q (it also contains additional conjuncts), and so t must also satisfy the query Q. □

3.4 Answers to a query

In the definition of a semantically correct plan we required only that P' be *contained* in Q and not *equivalent* to Q. There are two reasons for this. First, even if P' were equivalent to Q, the answer obtained by executing the conjunctive plan 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.

To conclude this section, 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. In the next section we describe our algorithm for computing query plans.

3.5 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 (which typically involves going over the network), and produces a relation corresponding to the free parameters in the query template. Interface programs also handle details such as contacting replicas if an information source is unavailable. Some of the details are given in Section 5.

4 Algorithms for Answering Queries

In this section, we present the algorithm used in the Information Manifold to generate semantically correct and executable query plans for a given query Q. Our algorithm proceeds in two stages. In the first stage, we generate conjunctive plans that are semantically correct. In the second stage we try to order the conjuncts of the plan to ensure that they are executable, i.e., that they satisfy the capability requirements of the query.

4.1 Generating Semantically Correct Query Plans

As explained in the previous section, a semantically correct plan guarantees that the answers produced will actually be answers to the query. In our discussion about semantically correct plans we ignore the input/output specifications of each subgoal in the plan (which will be computed in Section 4.2). Thus, in our discussion plans can be viewed as conjunctive queries.[3] As implied by the previous section, 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, Qia95], where the source relations play the role of the views.

The problem of answering queries using views is the following. Given a query Q using the relations $E_1, \ldots, E_m$, and a set of view definitions $V_1, \ldots, V_n$, over the same relations, find a query Q' that uses $V_1, \ldots, V_n$, such that Q is equivalent to Q'. There are two differences between our problem and previous treatments of the view rewriting problem. First, we require the rewriting only to contained in the query (but be a satisfiable query!), and not necessarily equivalent to the query. Second, we want to find *all* the rewritings of the query using the source relations, two equivalent plans (that use different information sources) will not necessarily produce the same set of answers.

The problem of answering queries using views is known to be NP-complete in [LMSS95], even for conjunctive queries without constraints. The algorithm suggested there is not very practical since it requires guessing a rewriting Q', and then checking whether it is a semantically correct solution. The main source of complexity is the fact that there are an exponential number of candidate rewritings. This is especially significant in our context because that algorithm would be exponential in the number of information sources. We now describe an algorithm that exploits the characteristics of the domain to drastically reduce the number of candidate rewritings considered.

Our algorithm has two steps. In the first, we compute 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

[3] We use the canonical augmented description of each source for testing correctness.

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 $1 \leq i \leq m$.
For $i = 1, \ldots, n$ 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 j'th variable in $\bar{Y}_j$ and $y \in \bar{Y}$
                    then $\psi(y) = x_j$, where x_j is the j'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.

bucket, and check whether it's a semantically correct plan. As we see in Section 5.2, the first step, whose running-time is polynomial in the number of sources, 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 Consider our query asking for sports cars manufactured no sooner than 1992:

$$\begin{aligned} q(m_1, p_1, r_1) \quad \leftarrow \quad & CarForSale(c_1), Category(c_1, \textsf{sportscar}), Year(c_1, y_1), y \geq 1992, \\ & Price(c_1, p_1), Model(c_1, m_1), ProductReview(m_1, y_1, r_1) \end{aligned}$$

and consider what happens when algorithm **CreateBuckets** looks at Source 1 and the first subgoal of our query $CarForSale(c_1)$. The canonical augmentation of the content description of Source 1 is:

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

therefore, the algorithm will find the mapping $c \rightarrow c_1$ and check whether the following conjunction is satisfiable:[4]

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

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

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 $ProductReview(m_1, y_1, r_1)$. $\Box$

In the second step of the algorithm we consider every conjunctive query Q' of the form

$$Q' : q'(\bar{X}) \quad :- \quad V_1(\bar{Y}_1),\ \ldots,\ V_n(\bar{Y}_n), C_q$$

where V_i is the head of a description in the bucket of the ith subgoal of Q. Any minimal subset of Q' that is either

- contained in Q, or
- can be made to be contained in Q by adding subgoals of built-in predicates

is added to the list of semantically correct solutions. To test containment, we can use an extension of a containment algorithm for conjunctive queries with built-in predicates [LS93] that considers the the functional dependencies (as in [CM77]) and the inclusion dependencies. Although containment is known to be intractable in general, its intractability is in the size of the query (which tends to be small), and only occurs when queries have multiple occurrences of the same relations. Consequently, the complexity of containment is not a problem in practice.

Example 4.2 As shown in Example 3.2 the query resulting from combining sources 1 and 5 is contained in the original query, and is therefore a semantically correct plan. Note that as a first step in the containment check we propagate the functional dependencies enforced by the single-valued attributes, and then we remove multiple occurrences of identical conjuncts. $\Box$

Our algorithm considers rewritings that have at most one source from each bucket. Consequently, we consider only rewritings that have at most the number of subgoals in the query (not counting the built-in subgoals). The result of [LMSS95] implies that when functional dependencies are not present and built-in predicates do not appear in the content descriptions, it suffices to consider only rewritings of this length. This means that although they may be longer rewritings that produce semantically correct conjunctive plans, any answer that would be obtained from a longer rewriting would also be produced by a rewriting whose length is bounded by the size of the query. However, as shown in [LMSS95, RSU95], in theory, the bound on the size of the rewriting does not hold when either functional dependencies, built-in subgoals or binding patterns occur. In such cases, we would need to take more than one source from each bucket in order to guarantee that we find all solutions. It should be noticed that in practice we have not found that we missed solutions because of bounding the length of rewritings we consider.

4.2 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.

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.

Example 4.3 Consider the semantically correct plan for answering our sportscar query:

$$P_1 : Q(m,p,r) \leftarrow V_1(c), V_2(m,y,r), Model(c,m), Year(c,y), Category(c, \textsf{sportscar}), y \geq 1992.$$

The set of available bindings 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 $V_2(m, y, r)$ 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. $\square$

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. A proof sketch is given in the appendix.

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'.* $\square$

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 occuring 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 which parameters were chosen in order to satisfy its input requirements (note that we are referring to the names of the parameters, not their values!). 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 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. The proof of the theorem is given in the appendix.

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.* $\square$

5 Implementation

5.1 The Information Manifold System

The Information Manifold system uses the techniques described in the previous sections to provide a uniform query interface to structured information sources on the World Wide Web and internal sources at AT&T Bell Laboratories. Figure 5 shows the architecture of the Information Manifold system.

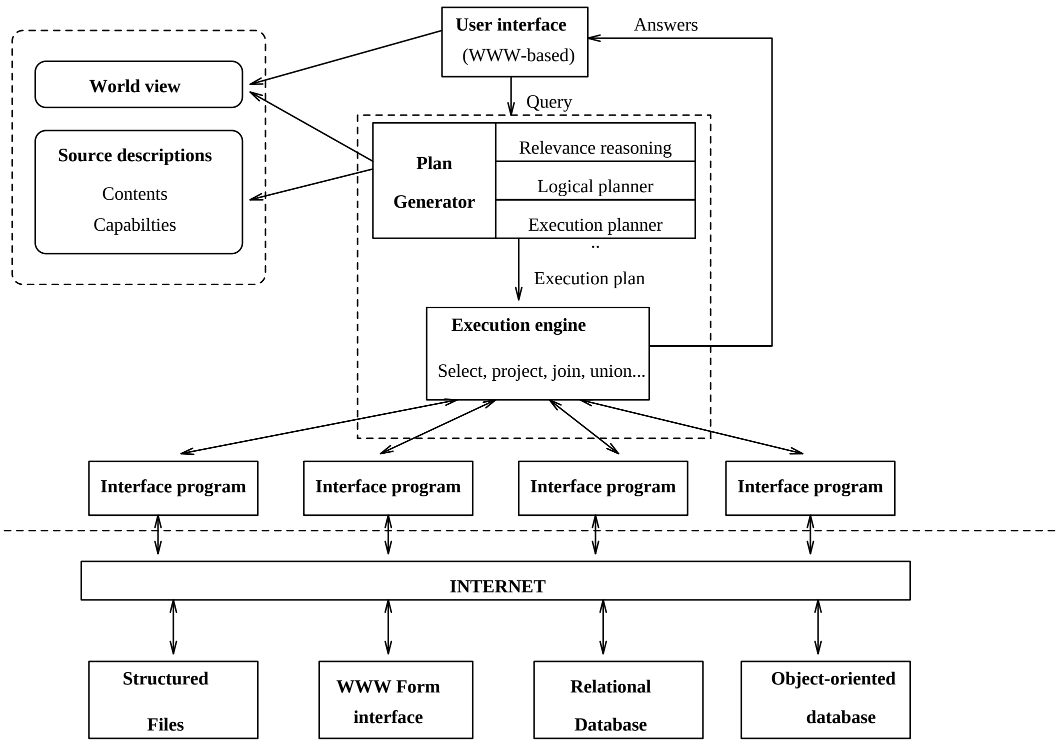

Figure 5: Architecture of the Information Manifold

Users interact with the Information Manifold through a web based interface. The interface enables users to browse the categories of information available (i.e., the world view), and to see which information sources are available. Users can formulate queries either using templates that are available for classes in the world-view, or by combining such templates into an arbitrary conjunctive query.

When a query is posed, the system uses the descriptions of information sources, as explained in the previous section, in order to decide which sources are relevant, and to compute the various ways in which answers to the query can be found (i.e., the executable solutions). 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. The plan executor also tries to access information sources in parallel, whenever the plan allows for parallelism.

The system currently provides access to over 100 information sources in various domains, including name servers, publication databases, market databases and entertainment sources (e.g., movie and video databases, CD stores). Our method of describing information sources has proved to be useful in practice by enabling us to quickly and accurately model a large number of sources, while leaving the world-view relatively stable. We have developed a set of tools which enable us to speed up the process of generating interface programs to information sources on the WWW. In particular, many sources on the WWW have a form-based interface. We have developed a tool in which we simply determine the correspondence between the variables used in the form and the

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.

world view, and then specify a grammar describing the format of the answers obtained from the source, and the bulk of the interface program is generated automatically. Several of the interface programs use an outerjoin-based technique [RU96] to convert hierarchically structured documents into relations.

5.2 Experimental Results

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:

- Query 1: *Find titles and years of movies featuring Tom Hanks.*
- Query 2: *Find titles and reviews of movies featuring Tom Hanks.*
- Query 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. All our experiments were run on a SGI Challenge computer with a clock speed of 150MHz. *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.

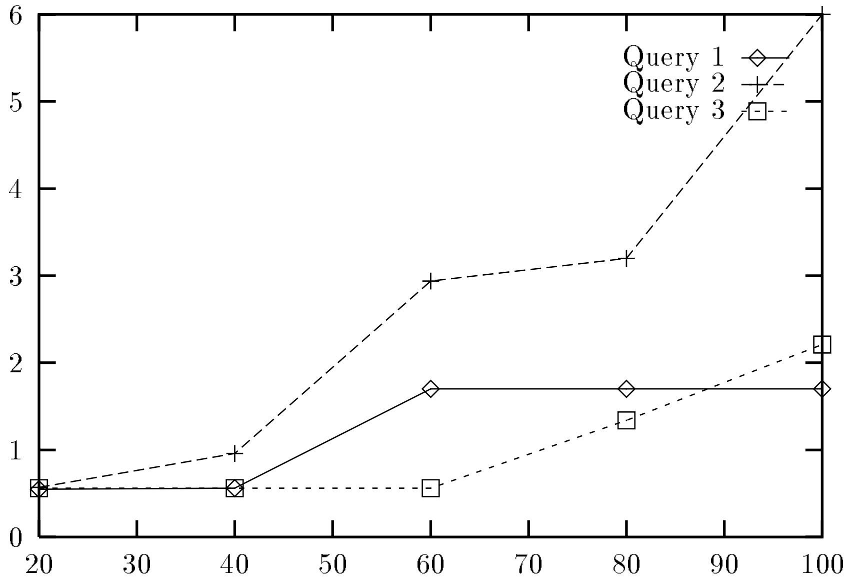

Figure 6: Total query planning time in seconds versus number of information sources.

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 pruning 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). This pruning is extremely important to ensure the *scalability* of our system, since we have to do an expensive query containment check for each enumerated plan.

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 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.

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

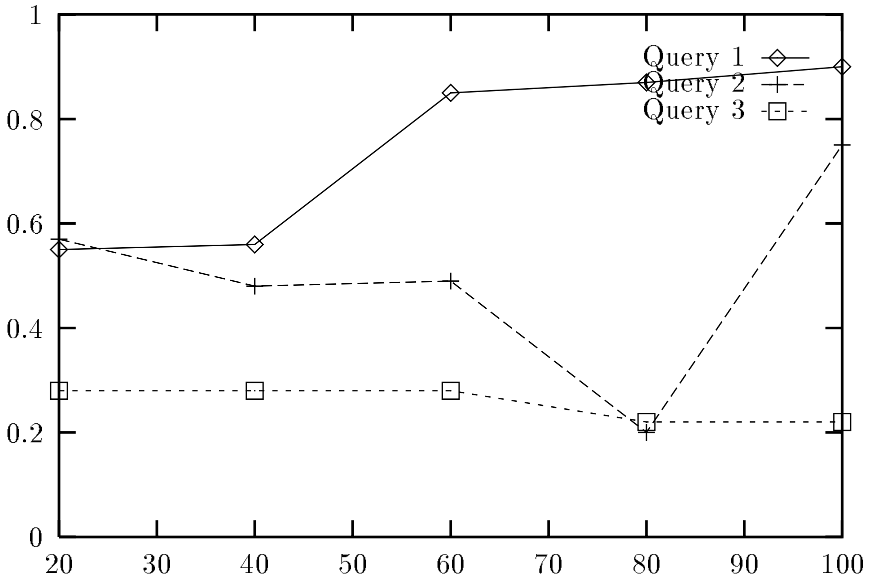

Figure 7: Average time per plan in seconds versus number of information sources.

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. For example, executing a query plan for Query 2, which involves querying multiple sources, parsing their answers, and computing a join, takes as much as 30 seconds with typical wide area network speeds.

6 Related Work

Our approach to integrating multiple information sources can be seen as treating the sources as a multidatabase or as a federated database (e.g., [ASD$^+$91, FHM94, HBP94, LMR90]). In fact, the content descriptions of information sources can be viewed as a generalization of *exported schemas* in multidatabases. However, in multidatabases the correspondence between the contents of the individual databases and the global schema is more direct. As one example, in the Pegasus system [ASD$^+$91] every external database is modeled as a class in the class hierarchy, which is disjoint from other classes. It is then possible to define superclasses that represent *unions* of databases. In the Information Manifold the contents of an information source can be defined as an arbitrary conjunctive query on classes and relations in the world-view. Therefore we do not have to create a class for every source, and we are able to make more fine-grained distinctions between the contents of sources. This difference has proven very important in practice in order to quickly add information sources. In Pegasus, determining the set of databases to access in order to answer a query is simple, because it is immediate from the query. One of the key difficulties in the Information Manifold is determining which information sources are relevant to a query. Finally, source capabilities are not considered in the multidatabase literature. The upshot of these differences is that in the Information Manifold the user can more easily specify *what* he or she wants, rather than having to formulate a query in a way that would guarantee that all the relevant information sources are accessed. On the other hand, it should be noted that since our goal is only to provide a *query* interface to the information sources, we are not concerned with issues of consistency and updating the information sources as in multidatabases.

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.

Several systems (e.g., TSIMMIS [CGMH+94, PGGMU95], SIMS [ACHK94], HERMES [SAB+95], CARNOT [WAC+93], DISCO [FRV95], Nomenclator [Ord93, OM93]) for integrating multiple information sources are being built on the notion of a *mediator* [Wie92]. 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 information 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. Furthermore, it is not clear how their techniques will scale up to large numbers of information sources. In contrast, most of the algorithms we use are efficient, and the sources of complexity (e.g., containment checks) are well understood and are limited in practice. The advantage of general purpose planning techniques is the flexibility in dealing with unexpected problems during query evaluation [Kno95].

Finally, there are several indexing systems for finding information on the World Wide Web (e.g., Harvest [BDMS94], Gloss [GGMT94], Yahoo, Lycos). However, all these systems are based on keyword searches on contents or annotations of documents, and are only able to retrieve documents that match these keywords. They cannot answer semantically meaningful queries that require considering the contents of the sources. The 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 (that can be checked using existing Unix utilities). W3QS is a very 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 contributions include practical algorithms for deciding which information sources are relevant to a query, and how to combine them in a way that adheres to the capabilities of the sources and exploits them when possible. 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.

The world-view with which the user interacts is designed with respect to the set of information sources to which we expect to provide access. An important issue is designing tools to obtain descriptions of information sources easily. In [LO95] we describe one such fielded tool designed to obtain descriptions of name server sources. The tool is based on asking the administrators of the name servers to annotate example entries from their databases so that we can infer the content descriptions of the source from the annotations.

There are several important areas of research we are currently pursuing. First, we are 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. The second extension we are considering is extending our modeling mechanism and algorithms to be able to deal partially with unstructured information sources.

Acknowledgments

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

Appendix

Proof Sketch of Theorem 4.1: The main observation underlying the proof is that if there is an ordering of the subgoals of Q', and a subgoal V_i appears in the i'th position, then its input requirements will still be satisfied even if other subgoals are pushed in front of it. This property is because the set of available parameters increases monotonically.

Given this observation, suppose there is an executable ordering $V_1, \ldots, V_n$ of the subgoals of Q'. The subgoal V_1 will be added at some point to the ordering by the algorithm because its input requirements are already satisfied by the explicit bindings in the query. Inductively, if the subgoals $V_1, \ldots, V_{i-1}$ are added at some point to the ordering, then the algorithm will ultimately add V_i to the ordering, since its input requirements are satisfied after $V_1, \ldots, V_{i-1}$ have been added. Therefore, since all subgoals will be added at some point, some ordering will be found. $\Box$

Proof of Theorem 4.2: The problem is in NP because we can simply guess an ordering, and check whether it satisfies the requirements of the information sources in polynomial time.

We show the NP-hardness by reducing the satisfiability problem of 3CNF formulas to our problem. Let Δ be a set of 3CNF formulas, with the propositional variables $p_1, \ldots, p_n$ and clauses $c_1, \ldots, c_m$. We construct a semantically correct plan with $n + m$ subgoals. Each subgoal uses a different information source, and all subgoals have exactly one variable, X. For each of the first n subgoals, we have the following capability record: $(\emptyset, \{p_i^1(X)\}, \emptyset, 0, 0)$, $(\emptyset, \{p_i^2(X)\}, \emptyset, 0, 0)$ that is, there are no requirements on the input or selection parameters, but there are two possible outputs, either the p_i^1 or p_i^2 parameter. Intuitively, the choice of output determines the truth value of the proposition p_i. The next m subgoals are one for each clause. Suppose the i'th clause is $\{L_1, L_2, L_3\}$, where each L_i is a literal. The $(n+i)$'th subgoal has one capability record. The input set S_{in} contains three elements: for $1 \leq j \leq 3$ it contains $p_i^1(X)$ if $L_i = p_i$ and $p_i^2(X)$ if $L_i = \neg p_i$. The minimum number of inputs is 1 and the maximal number is 3. There are no output or selection parameters.

Clearly, if Δ is satisfiable, there is a way to satisfy the capability requirements in the query. We choose the capabilities of the first n subgoals to produce exactly the satisfying assignment to the variables of Δ (i.e., $p_i^1(X)$ if p_i is assigned $True$, and $p_i^2(X)$ otherwise). The next m subgoals are satisfied, since each of the clauses in Δ are satisfied by the truth assignment. Similarly, if there is a way to satisfy the requirements of the information sources then we can build a satisfying assignment in a straightforward fashion. Finally, our reduction is polynomial because the query we produced is polynomial in the size of Δ. $\Box$

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.

[ACM93] Serge Abiteboul, Sophie Cluet, and Tova Milo. Querying and updating the file. In *Proceedings of the 19th VLDB Conference, Dublin, Ireland*, 1993.

[ASD+91] Rafi Ahmed, Phillippe De Smedt, Weimin Du, William Kent, Mohammad A. Ketabchi, Witold A. Litwin, Abbas Rafii, and Ming-Chien Shan. The Pegasus heterogeneous multidatabase system. *IEEE Computer*, pages 19–26, December 1991.

[BDMS94] C. Mic Bowman, Peter B. Danzig, Udi Manber, and Michael F. Schwartz. Scalable internet resource discovery: Research problems and approaches. *CACM*, 37(8):98–107, August 1994.

[CGMH+94] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogenous information sources. In proceedings of IPSJ, Tokyo, Japan, October 1994.

[CKPS95] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Proceedings of International Conference on Data Engineering*, 1995.

[CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[EW94] Oren Etzioni and Dan Weld. A softbot-based interface to the internet. *CACM*, 37(7):72–76, 1994.

[FHM94] Douglas Fang, Joachim Hammer, and Dennis McLeod. The identification and resolution of semantic heterogeneity in multidatabase systems. In *Multidatabase Systems: An Advanced Solution for Global Information Sharing*, pages 52–60. IEEE Computer Society Press, Los Alamitos, California, 1994.

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

[GGMT94] Luis Gravano, Hector Garcia-Molina, and Anthony Tomasic. The effectiveness of gloss for the text database discovery problem. In *Proceedings of SIGMOD-94*, pages 126–137, 1994.

[HBP94] A. R. Hurson, M. W. Bright, and S. H. Pakzad, editors. *Multidatabase Systems: An Advanced Solution for Global Information Sharing.* IEEE Computer Society Press, Los Alamitos, California, 1994.

[Kno95] Craig A. Knoblock. Planning executing, sensing and replanning for information gathering. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[KS95] David Konopnicki and Oded Shmueli. W3QS: A query system for the WWW. In *Proceedings of the 21st VLDB Conference, Zurich, Switzerland*, 1995.

[LMR90] Witold Litwin, Leo Mark, and Nick Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22 (3):267–293, 1990.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, CA*, 1995.

[LO95] Alon Y. Levy and Joann J. Ordille. Integrating internet information sources. In *Working Notes of the AAAI Fall Symposium on AI Applications in Knowledge Navigation*, 1995.

[LRU96] Alon Y. Levy, Anand Rajaraman, and Jeffrey D. Ullman. Answering queries using limited external processors. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Montreal, Canada (to appear)*, 1996.

[LS93] Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *Proceedings of the 19th VLDB Conference, Dublin, Ireland*, pages 171–181, 1993.

[LSK95] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval*, 5 (2), September 1995.

[Mor88] K. A. Morris. An algorithm for ordering subgoals in NAIL! In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 82–88, 1988.

[OM93] Joann J. Ordille and Barton 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, Pittsburgh, USA*, pages 120–129, 1993.

[Ord93] Joann J. Ordille. *Descriptive Name Services for Large Internets*. PhD thesis, University of Wisconsin, Madison, WI, 1993.

[PGGMU95] Yannis Papakonstantinou, Ashish Gupta, Hector Garcia-Molina, and Jeffrey D. Ullman. A query translation scheme for rapid implementation of wrappers. In *Proceedings of the Conference on Deductive and Object Oriented Databases, (DOOD)*, 1995.

[Qia95] Xiaolei Qian. Query folding. Technical Report SRI-CSL-95-09, Stanford Research Institute, Menlo Park, California, 1995.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, CA*, 1995.

[RU96] Anand Rajaraman and Jeffrey D. Ullman. Integrating information by outerjoins and full disjunctions. To appear in the Proceedings of the Fifteenth Symposium on Principles of Database Systems (PODS). Available by anonymous ftp from `db.stanford.edu` as the file `pub/rajaraman/1995/outerjoin.ps`, 1996.

[SAB+95] V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. Lu, A. Rajput, T. Rogers, R. Ross, and C. Ward. HERMES: A heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1995.

[WAC+93] Darrel Woelk, Paul Attie, Phil Cannata, Greg Meredith, Amit Seth, Munindar Sing, and Christine Tomlinson. Task scheduling using intertask dependencies in Carnot. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 491–494, 1993.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, 1992.

[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.