Navigational Plans For Data Integration

Marc Friedman
University of Washington
friedman@cs.washington.edu

Alon Levy
University of Washington
alon@cs.washington.edu

Todd Millstein
University of Washington
todd@cs.washington.edu

Abstract

We consider the problem of building data integration systems when the data sources are webs of data, rather than sets of relations. Previous approaches to modeling data sources are inappropriate in this context because they do not capture the relationships between linked data and the need to navigate through paths in the data source in order to obtain the data. We describe a language for modeling data sources in this new context. We show that our language has the required expressive power, and that minor extensions to it would make query answering intractable. We provide a sound and complete algorithm for reformulating a user query into a query over the data sources, and we show how to create query execution plans that both query and navigate the data sources.

Introduction

The purpose of data integration is to provide a *uniform* interface to a multitude of data sources. Data integration applications arise frequently as corporations attempt to provide their customers and employees with a consistent view of the data associated with their enterprise. Furthermore, the emergence of XML as a format for data transfer over the world-wide web is making data integration of autonomous, widely distributed sources an imminent reality. A data integration system frees its users from having to *locate* the sources relevant to their query, *interact* with each source in isolation, and manually *combine* the data from the different sources. The problem of data integration has already fueled significant research in both the AI and Database communities, e.g., (Ives *et al.* 1999; Cohen 1998b; Knoblock *et al.* 1998; Beeri *et al.* 1998; Friedman & Weld 1997; Duschka, Genesereth, & Levy 1999; Garcia-Molina *et al.* 1997; Haas *et al.* 1997; Levy, Rajaraman, & Ordille 1996; Florescu, Raschid, & Valduriez 1996; Adali *et al.* 1996), as well as several industrial solutions.

Data integration systems are usually built according to the following architecture. Each data source is modeled as a relation (or a set of relations). The user poses queries in terms of the relations and attributes of a *mediated database schema* as opposed to the schemas of the individual sources. The relations in the mediated schema are *virtual* in the sense that their extensions (i.e., the tuples of the relations) are not actually stored anywhere. The mediated schema is manually designed for a particular data integration application, and is intended to capture the aspects of the domain of interest to the users of the application. In addition to the mediated schema, the system has a set of *source descriptions* that specify the semantic mapping between the mediated schema and the source schemas. The data integration system uses these source descriptions to reformulate a user query into a query over the source schemas.

Two of the main approaches for specifying source descriptions use restricted forms of first-order logic sentences. In the *global-as-view* (GAV) approach (Garcia-Molina *et al.* 1997; Adali *et al.* 1996) Horn rules define the relations in the mediated schema in terms of the source relations. The *local-as-view* (LAV) approach (Duschka, Genesereth, & Levy 1999; Friedman & Weld 1997; Levy, Rajaraman, & Ordille 1996) is the opposite: the source relations are defined as expressions over the relations in the mediated schema.

As the WWW expands and sites become more complex, we observe a growing number of sources that can no longer be modeled as sets of relations, but rather as *webs* of data with a set of entry points. In this paper we consider the problem of modeling and answering queries using such sources. Our first observation is that modeling web sites requires combining the expressive power of GAV and LAV. There are two other characteristics distinguishing data webs from collections of relations: (1) linked pairs of pages contain related data, and (2) obtaining the data from the site may require navigation through a particular path in the site. These properties render previous formalisms inappropriate for incorporating data webs as sources in a data integration system. Previous works that considered such sources (e.g., the ARIADNE System (Knoblock *et al.* 1998)) modeled each page as a separate data source and assumed each page was an entry point.

This paper describes a formalism for modeling data webs, a formalism for incorporating them into a data integration system, and an algorithm for reformulating user queries into execution plans that both *query* and *navigate* the data sources. Our solution combines the following contributions.

First, we describe a formalism for modeling data webs. The formalism captures the contents of each page in the data web, the relationships between linked pages, and the constraints on the possible paths through the web.

Second, we describe GLAV, a language for source descriptions that is more expressive than GAV and LAV combined. We describe a query reformulation algorithm for sources described in GLAV and show that query answering for GLAV sources is no harder than it is for LAV sources. Furthermore, we show that in some sense, GLAV reaches the limits on the expressive power of a data source description language. Slight additions to the expressive power of GLAV would make query answering co-NP-hard in the size of the data in the sources. It should be noted that GLAV is also of interest for data integration independent of data webs, because of the flexibility it provides in integrating diverse sources.

Finally, we show how to reformulate user queries into execution plans over the data sources. The reformulation consists of two parts. First we use our GLAV reformulation algorithm to obtain a query over the relations in the data webs. Then we augment the resulting query over the sources with the navigational instructions needed to interact with the data webs.

Incorporating Data Webs

In this section we describe how we represent data webs and incorporate them into a data integration system. We begin by recalling some basic terminology. Then we explain how we model a data web by a web schema, and finally we explain how to specify the relationship between the relations in web schemas and the mediated schema.

Preliminaries

In our discussion variables are denoted by capital letters and constants by lowercase letters. Overscores denote tuples of zero or more variables and constants (e.g., $\bar{X}$). An atom consists of a predicate symbol p followed by an argument list $(\bar{X})$. A Horn rule is a logical sentence of the form $r_1(\bar{X}_1) \wedge \ldots \wedge r_k(\bar{X}_k) \Rightarrow r(\bar{X})$, where $\bar{X} \subseteq \bigcup_i \bar{X}_i$. The variables in $\bar{X}$ are universally quantified, and all other variables are existentially quantified. Given the extensions of the relations appearing in the antecedent, a Horn rule defines a unique extension for the relation in the consequent. It should be noted that there is a direct correspondence between single Horn rules and select-project-join queries in relational databases (often called *conjunctive queries*).

Datalog programs are sets of Horn rules, in which the predicate symbols appearing in the consequents, called the intensional database (IDB) predicates, may also appear in any antecedents. A datalog program defines a unique extension for the IDB relations given the extensions of the other relations, called the extensional database (EDB) relations, as follows. We begin with empty extensions for the IDB relations and apply the rules, deriving new facts for the IDB relations, until no new facts can be derived.[1] We often distinguish one IDB predicate as the query predicate. The result of applying a datalog program to a database is the set of tuples computed for the query predicate.

Data Webs

A data web consists of pages and the links between them. In this paper we are concerned with the logical modeling of data webs. In practice, one also needs to address the labor-intensive problem of writing programs (called *wrappers*) that extract structured data from an HTML page. While several researchers have successfully automated wrapper construction (Cohen 1998a; Kushmerick, Doorenbos, & Weld 1997; Ashish & Knoblock 1997), the emergence of XML as a common data format will alleviate this obstacle considerably in the future.

In order to model a data web we need to represent the set of pages and links in the web, the data available at every page, whether each link is a hyperlink or a search form, and which pages can be accessed directly by name. We represent the structure of a data web with a *web schema*, a directed graph G with nodes representing sets of pages and directed edges representing sets of directed links between them. For example, Figure 1 shows web schemas for three different university webs. Nodes in G are annotated with:

- the node's name and unique id
- a parameter (a variable or constant)
- an *entry point* flag (an asterisk)
- a list of contents

Every node name defines a function symbol.[2] For example, consider node 1 in Figure 1, representing the home page of university u_1. Its name is *Univ*, with parameter u_1, a constant. $Univ(u_1)$ denotes the home page object of university u_1. Every web site has a set of entry points, *i.e.*, nodes that the integration system can access directly by URL (as opposed to those accessed via a link from some other node). We indicate them with an asterisk. For example, node 1 is an entry point to university u_1's data web. In order for an entry point node to be accessed directly, we require that its parameter be a constant.

There are three kinds of source relations appearing on a page $N(X)$. These correspond to ordinary contents of the page, outgoing edges from the page, and search forms on the page. (1) Tuples of a predicate p are listed as atoms of the form $p(Y_1, \ldots, Y_k)$ in the contents of $N(X)$. For instance, node 7, a department page, contains the source relation $chair(D, P)$, indicating that department pages list their department chairs.

[1]This unique model is known as the least fixed-point model of the Horn rules. Since our discussion only considers the derivation of positive atoms, the difference between the least fixed-point semantics and the classical first-order semantics is immaterial.

[2]We use only unary function symbols to simplify the exposition.

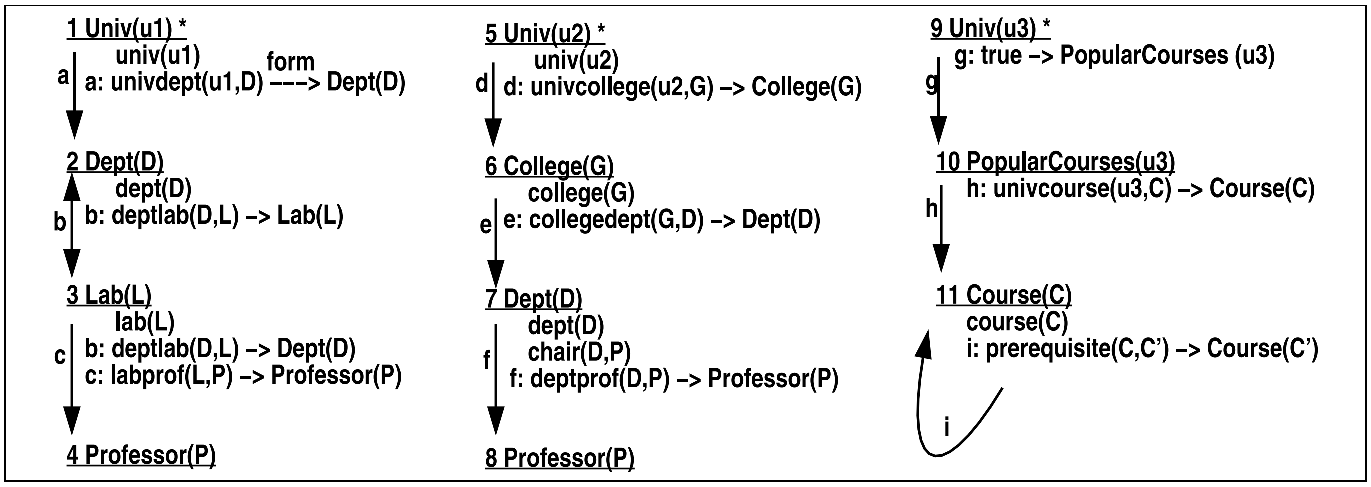
1 Univ(u1) *
univ(u1)
form
a: univdept(u1,D) ---> Dept(D)
a
2 Dept(D)
dept(D)
b: deptlab(D,L) -> Lab(L)
b
3 Lab(L)
lab(L)
b: deptlab(D,L) -> Dept(D)
c: labprof(L,P) -> Professor(P)
c
4 Professor(P)
5 Univ(u2) *
univ(u2)
d: univcollege(u2,G) -> College(G)
d
6 College(G)
college(G)
e: collegedept(G,D) -> Dept(D)
e
7 Dept(D)
dept(D)
chair(D,P)
f: deptprof(D,P) -> Professor(P)
f
8 Professor(P)
9 Univ(u3) *
g: true -> PopularCourses (u3)
g
10 PopularCourses(u3)
h: univcourse(u3,C) -> Course(C)
h
11 Course(C)
course(C)
i: prerequisite(C,C') -> Course(C')
i

Figure 1: Three university web schemas.

Typically X will appear as one of the Y_i's, but in general it may not. (2) Links from a page are often labelled with an identifier of the target page. If a link from page $N(X)$ to page $M(Y)$ is labelled with p, we indicate this by the expression $p(X,Y) \longrightarrow M(Y)$.[3] For instance, the $Univ(u_2)$ page (node 5) lists each college G satisfying source relation $univcollege(u_2, G)$, with a link to that college's page. (3) Search forms, much like links, map from binary relations to other pages, but the value of the target page parameter Y must be provided before accessing the link. We denote this by

$$p(X,Y) \xrightarrow{form} M(Y).$$

Note that in this case the value of Y is not available on the page $N(X)$. For instance, node 1 has a search form in which the user enters a department name, and the home page of that department is returned.

Mediated Schemas

A set of relations known as a mediated schema serves as a uniform query interface for all the sources. It is designed to represent the attributes of the domain relevant to the integration application and does not necessarily represent all of the attributes available in all of the sources. In our university domain, we use the following mediated schema.

$$\begin{array}{l} collegeOf(College, University) \\ deptOf(Department, College) \\ profOf(Professor, Department) \\ courseOf(Course, Department) \\ chairOf(Professor, Department) \\ prereqOf(Course, Course) \end{array}$$

As data webs are added, they need only be 'hooked' to the mediated schema, without reference to the other data webs. This is done via a *source description* language, which relates the source relations to the mediated schema relations.

[3]We rely on the wrappers to determine the label of a link from syntactic cues, either in the URL, the anchor text, or the position of the link in the document.

GLAV Source Descriptions

Source description languages are necessary because the mediated schema relations do not match the source relations in a one-to-one fashion. There are two reasons for the mismatch. First, the source schemas often contain differing levels of *detail* from each other, and from the mediated schema. In Figure 1, university u_2 identifies the colleges in a university, a distinction that does not exist in university u_1. On the other hand, u_1 identifies laboratories within departments, a detail not in the mediated schema or in u_2.

The second reason is that even if the different schemas model the same information, they may split attributes into relations in different ways (in database terms, this corresponds to different normalizations of a database schema). For example, one schema may choose to store all the attributes of a person in a single relation, while another may decide to have a separate relation for every attribute.

The LAV and GAV source description languages only partially address these problems. LAV source descriptions have the form

$$v(\bar{X}) \Rightarrow r_1(\bar{X}_1, \bar{Z}_1) \wedge \ldots \wedge r_k(\bar{X}_k, \bar{Z}_k)$$

where v is a source relation, the r_i's are mediated schema relations, and $\bar{X} = \bigcup_i \bar{X}_i$. LAV descriptions handle the case in which the mediated schema contains details that are not present in every source, such as colleges. Statement 1 in Figure 2 is an example of a LAV source description.

The GAV language deals with the converse case, when the source contains details not present in the mediated schema. Descriptions in GAV have the form

$$v_1(\bar{X}_1, \bar{Y}_1) \wedge \ldots \wedge v_j(\bar{X}_j, \bar{Y}_j) \Rightarrow r(\bar{X}).$$

source description	#
u_1 :	
$univdept(u_1, D) \Rightarrow deptOf(D, G) \wedge collegeOf(G, u_1)$.	1
$deptlab(D, L) \wedge labprof(L, P) \Rightarrow profOf(D, P)$.	2
u_2 :	
$univcollege(u_2, G) \Rightarrow collegeOf(G, u_2)$.	3
$collegedept(G, D) \Rightarrow deptOf(D, G)$.	4
$deptprof(D, P) \Rightarrow profOf(P, D)$.	5
$chair(D, P) \Rightarrow chairOf(P, D)$	6
u_3 :	
$prereq(C, C')$:- $prerequisite(C, C')$.	7
$prereq(C, C')$:- $prereq(C, C'') \wedge prereq(C'', C')$.	
$prereq(C, C') \Rightarrow prereqOf(C', C)$.	

Figure 2: Sample source descriptions.

Statement 2 in Figure 2 is an example of a GAV source description.

Using either pure LAV or pure GAV source descriptions has undesirable consequences. In LAV, the mediated schema must contain all attributes shared by multiple source relations, whether or not they are of interest in the integration application. In our example, the lab name L is such an attribute. To relate departments to professors in LAV, a lab attribute must be in the mediated schema. To make matters worse, some sites use shared attributes that are only meaningful internally, such as URLs of intermediate pages or local record ids. In GAV, on the other hand, the mediated schema relations must all be relations present in the sources, or conjunctive queries over them, making the mediated schema contingent on which source relations are available.

Hence, we propose the GLAV language that combines the expressive power of both LAV and GAV, allowing flexible schema definitions independent of the particular details of the sources. Formally, a statement in GLAV is of the form

$$V(\bar{X}, \bar{Y}) \Rightarrow r_1(\bar{X}_1, \bar{Z}_1) \wedge \ldots \wedge r_k(\bar{X}_k, \bar{Z}_k)$$

where $(\cup_i \bar{Z}_i) \bigcap \bar{Y} = \emptyset$, and $V(\bar{X}, \bar{Y})$ is either a conjunction of source relations, or the distinguished query predicate of a datalog query over source relations. GLAV source descriptions for the university example are in Figure 2. GLAV combines the expressive power of GAV and LAV and allows source descriptions that contain recursive queries over sources. Recursion is useful when retrieving the desired information requires navigating arbitrarily long paths. University u_3 contains such a situation: in order to obtain the prerequisites of a given course, it may be necessary to traverse the prerequisite edge (which represents direct prerequisites) arbitrarily many times before finding them all. The multi-line Statement 7 illustrates this example in GLAV, where *prereq* is a new relation defined by a datalog program over the source relations.

Data Integration Domains

In summary, a set of web schemas and a set of source descriptions in GLAV form a *data integration domain*. Formally, a data integration domain D is a triple $\langle \mathcal{R}, \{G_i\}, \mathcal{SD} \rangle$, consisting of the set of mediated schema relations $\mathcal{R}$, web schemas G_i, and source descriptions $\mathcal{SD}$.

Planning to Answer a Query

A user of a data integration system poses a query over the mediated schema relations, which the system answers using a query processor. We consider conjunctive queries in which the consequent is the new predicate symbol q, and the antecedents are mediated schema relations. For instance, to retrieve all chairs of history departments, a user could pose the query:

$$chairOf(Person, history) \Rightarrow q(Person).$$

To collect the answers to the query automatically, the integration system must translate this into a low-level procedural program, called an *execution plan*. For a relational query processor, this program is expressed in annotated relational algebra, which has operators to fetch relations and do basic relational operations such as project, select, join, and union. The annotations indicate operator implementations, memory allocations, and scheduling. In this work we augment relational algebra with an operation that traverses sets of links.

This section describes how to reformulate a query into an execution plan. We generate plans at progressively more detailed levels. First we construct a *logical plan* by reformulating the user's query into a query over the source relations in the data webs. Then we augment the logical plan with navigational information to describe how to locate the desired relations in the data webs, forming a *navigational plan*. Converting a navigational plan into an efficient execution plan is beyond the scope of this paper. See (Ives *et al.* 1999) for work on optimization of data integration queries. A non-recursive navigational plan can be converted straightforwardly into an execution plan in augmented relational algebra, though we do not provide the details here.

Logical Plans

A logical plan is a datalog program whose EDB relations are the source relations and whose answer predicate is q. The soundness and completeness of a logical plan can be defined in terms of logical entailment with respect to the source descriptions and contents of the data webs. Specifically, let T be the knowledge base containing the sentences in the source descriptions $\mathcal{SD}$, the ground atoms representing the extensions $\mathcal{I}$ of the source relations, and the query Q. Let P be the logical plan constructed for $\mathcal{SD}$ and Q, and let $P(\mathcal{I})$ be the set of facts derived by applying P to the database $\mathcal{I}$. The logical plan P is sound (resp. complete) if for every ground atom $q(\bar{a})$, $q(\bar{a}) \in P(\mathcal{I}) \Longrightarrow T \models q(\bar{a})$ (resp. $T \models q(\bar{a}) \Longrightarrow q(\bar{a}) \in P(\mathcal{I})$).

```
Given source descriptions $\mathcal{SD}$ and query $Q$,
$\Delta = \{Q\}$.
for each source description $s \in \mathcal{SD}$,
  let $s$ be $V(\bar{X},\bar{Y}) \Rightarrow r_1(\bar{X}_1,\bar{Z}_1) \wedge \ldots \wedge r_k(\bar{X}_k,\bar{Z}_k)$.
  $\Delta \mathrel{+}= V(\bar{X},\bar{Y}) \Rightarrow u_s(\bar{X})$,
    where $u_s$ is a new predicate symbol.
  let $\bar{Z} = \bigcup_i(\bar{Z}_i) = \{Z_1,\ldots,Z_m\}$.
  for each $Z_i$ in $\bar{Z}$,
    let $f_{s,i}$ be a new function symbol.
  for $l = 1$ to $k$,
    let $\bar{f_{s,l}}$ represent the vector of $f$'s
    corresponding to $\bar{Z}_l$.
    $\Delta \mathrel{+}= u_s(\bar{X}) \Rightarrow r_l(\bar{X}_l, \bar{f_{s,l}}(\bar{X}))$.
return $\Delta$.
```

Figure 3: Algorithm GlavInverse

```
Given logical plan $\Delta$ and web schemas $\{G_i\}$,
$\Delta' = \Delta$.
**location rules:**
for each source relation $r(\bar{X})$,
  for each node $N(Z)$ it appears on,
    $\Delta' \mathrel{+}= N(Z) \wedge on_{N,r}(Z,\bar{X}) \Rightarrow r(\bar{X})$.
    (Note $Z$ may or may not be in $\bar{X}$.)
**reachability rules:**
for each edge $e$ from $N(X)$ to $M(Y)$ with pred $r$,
  $\Delta' \mathrel{+}= N(X) \wedge r(X,Y) \overset{e}{\Rightarrow} M(Y)$.
for each entry point $N(X)$,
  $\Delta' \mathrel{+}= N(X)$.
return $\Delta'$.
```

Figure 4: Algorithm NavigationalPlan

Given a conjunctive query Q over the mediated schema relations, we construct a sound and complete logical plan for the query using the GlavInverse algorithm (Figure 3).[4] The key insight is that although the source descriptions are written in GLAV, an extension of the inverse rule method for LAV (Duschka, Genesereth, & Levy 1999) correctly produces the desired set of rules. Moreover, the low polynomial complexity of the inverse rules method is unchanged.

The algorithm converts the theory T into a datalog program. The theory T differs from an ordinary datalog program in two ways. (1) Not all of the rules in T are Horn rules, since the source descriptions may have a conjunction of atoms in the consequent. The algorithm converts each such source description into several Horn rules whose conjunction is equivalent to the original rule. (2) Source descriptions may have existential variables in the consequent. The algorithm converts these variables into terms containing function symbols. Although the resulting logical plan contains function symbols, their form is limited in such a way that naive datalog evaluation of the plan on a database will terminate (Duschka, Genesereth, & Levy 1999).

Theorem 1 *Let $D = \langle \mathcal{R}, \{G_i\}, \mathcal{SD} \rangle$ be an information integration domain. Let Q be a conjunctive query. Then the logical plan Δ returned by* GlavInverse *is sound and complete.*

Proof Sketch: Let $\mathcal{I}$ be the extensions of the source relations in D, and $T = \mathcal{SD} + Q + \mathcal{I}$. It suffices to show that any ground fact $q(\bar{X})$ entailed by T is a result of evaluating Δ on database $\mathcal{I}$, and vice versa. **(1)** u **predicates:** (Soundness) It is easy to see that $\Delta + \mathcal{I}$ entails only consequences of T. In particular, any proof tree for a fact x in $\Delta + \mathcal{I}$ can be converted into a proof tree for x in T by applying modus ponens to rules in Δ in order to remove uses of the u predicates. (Completeness) Since all facts in T are positive, and all rules have non-null consequent, T entails no negative facts. A sufficient deduction rule to obtain all *consequences* of T is to nondeterministically apply modus ponens on any rule and any facts. Consider a proof tree of a fact x from T. A proof tree for x in $\Delta + \mathcal{I}$ can be constructed by splicing in nodes for the u predicates. **(2) function symbols:** Function symbols are usually manipulated as constants. Since we don't consider a fact containing a function symbol an answer to a query, however, function symbols behave exactly like existential variables. Since there is one for each existential variable, the conversion to function symbols is just a renaming of variables, which has only a superficial effect on the proof trees.

Navigational Plans

Logical plans by themselves cannot be executed, since they do not explain how to populate the source relations from the data webs. (Lambrecht & Kambhampati 1998; Friedman & Weld 1997) explore how to optimize and execute logical plans when the source relations are directly accessible. In data webs, though, we have access only to the page relations and links, from which we need to generate source relation atoms. A navigational plan will be a logical plan augmented with rules to do exactly that.

Algorithm NavigationalPlan in Figure 4 produces a navigational plan Δ' given a logical plan Δ and the web schemas. We write $on_{N,p}(X,\bar{Y})$ to denote the fact that source relation $p(X,\bar{Y})$ appears on node $N(X)$ in the web schema. First the algorithm adds location rules which associate a source relation with a node on which it appears. For example, the relation *chair* appears on only one page, so we add the single rule

$$Dept(D) \wedge on_{Dept,chair}(D,D,P) \Rightarrow chair(D,P).$$

Next we add reachability rules, indicating how to reach each node. For instance, the rule

$$College(G) \wedge collegedept(G,D) \overset{e}{\Rightarrow} Dept(D)$$

indicates that we can reach node $Dept(D)$ if we are at node $College(G)$ by taking edge e. Edge labels like e

[4] A sound and complete logical plan is equivalent to the definition of a maximally-contained query plan in (Abiteboul & Duschka 1998).

A	$chairOf(P, history) \Rightarrow q(P)$.
B	$chair(D, P) \Rightarrow chairOf(P, D)$.
C	$Dept(D) \wedge on_{Dept,chair}(D, D, P) \Rightarrow chair(D, P)$.
D	$College(G) \wedge collegedept(G, D) \overset{e}{\Rightarrow} Dept(D)$.
E	$Univ(U) \wedge univcollege(U, G) \overset{d}{\Rightarrow} College(G)$.
F	$Univ(u_2)$.

Figure 5: Plan to find history department chairs

do not affect semantics, but are useful when extracting a path from the derivation tree of a tuple. The fact $Univ(u_2)$ indicates that we can reach entry point $Univ(u_2)$ unconditionally.

Figure 5 shows all of the relevant parts of the navigational plan produced from the history chair query. We start with the logical plan, rules A and B. Rule C indicates where the source relation *chair* is located. Rules D and E trace the path from the entry point, indicated by rule F, and the edge labels d and e describe the specific path taken.

Let $D = \langle \mathcal{R}, \{G_i\}, \mathcal{SD} \rangle$ be an information integration domain. Let Δ' be a navigational plan for D, and let Δ be the underlying logical plan, obtained by removing all rules in Δ' that refer to nodes in $\{G_i\}$. Let $\mathcal{W}$ be a set of data webs consistent with $\{G_i\}$, and let $\mathcal{I}$ be the extension of the source relations in $\mathcal{W}$. We say that a derivation of some fact in $\Delta(\mathcal{I})$ is *navigationally sound* if each source atom in the derivation is located on some page in $\mathcal{W}$ reachable from an entry point. Δ' is sound whenever Δ is sound, and for any $q(\bar{a})$ derivable from $\Delta'(\mathcal{W})$, $q(\bar{a})$ is also derivable from $\Delta(\mathcal{I})$ and this derivation is navigationally sound. Δ' is complete whenever Δ is complete, and for any $q(\bar{a})$ derivable from $\Delta(\mathcal{I})$ such that this derivation is navigationally sound, $q(\bar{a})$ is also derivable from $\Delta'(\mathcal{W})$.

Theorem 2 *The navigational plan Δ' produced by* NavigationalPlan *is sound and complete.*

Navigational plans ensure that links and page relations can only be accessed after the page they are on is proved to be reachable from an entry point. The antecedent ordering used in NavigationalPlan guarantees that that is always the case when the goals are solved left-to-right. Search forms actually require more machinery than shown here. In that case, domain rules and domain predicates need to be added to the plan to enforce binding pattern restrictions (Duschka, Genesereth, & Levy 1999).

Optimization and Execution

In principle, it is straightforward to give navigational plans operational semantics and to execute them with an interpreter. However in a traditional database system, logical plans are converted into trees of relational algebra expressions, optimized, and executed by a query processor. Navigational plans can be compiled into expressions in relational algebra augmented with the operator $traverse(Source, Edge, Target)$, which returns the target page given a source page and edge. The query optimization methods discussed in (Florescu *et al.* 1999; Mecca, Mendelzon, & Merialdo 1998) can be applied in this context as well.

The Complexity of GLAV

In this section we show that GLAV reaches the limits of the tradeoff between expressive power and tractability of query answering in data integration systems. Two measures of complexity are traditionally used for query processing problems: query complexity and data complexity. Query complexity measures the query answering time in terms of the size of the query Q, holding the size of the data fixed. High query complexity (NP-complete or worse), which is quite common for practical database languages, is not considered a serious impediment to implementation, because queries are generally considered to be very small compared with the size of the data. Data complexity measures the query answering time in terms of the size of the data. Since the data (data webs in this case) can be quite large, data complexity is by far the more important measure. In our discussion we model accessing a page, fetching a tuple of a relation, and traversing a link as unit-time operations.

Figures 3 and 4 result in a method for query answering over data webs using GLAV. This method has exactly the same complexity as current query answering methods for relational sources using LAV (Duschka, Genesereth, & Levy 1999). Both methods construct a plan and evaluate it on the data. Our next theorem summarizes the complexity of our algorithms, mirroring those for the traditional relational LAV case (Duschka, Genesereth, & Levy 1999).

Theorem 3 *Given data integration domain D with GLAV source descriptions, conjunctive query Q, and data webs W, (i) the query complexity of generating navigational plans is polynomial, (ii) the query complexity of answering queries is exponential, and (iii) the data complexity of answering queries is polynomial.*

In answering queries using views, there are three inputs of arbitrary size: the query, the view definitions, and the data. As a result, view complexity – the complexity as a function of the number and size of the view definitions – becomes interesting. Indeed, it is not as important as data complexity, since the data in the views should be larger than the view definitions in practice. However, view definitions can still be large and numerous. We show that for even very restricted cases of inequalities in the query, complexity is $co\text{-}\mathcal{NP}$-hard in the size of the largest view.

Theorem 4 *For conjunctive queries with constraints of the form $v \neq c$, for conjunctive views, the view complexity of determining whether a tuple is a certain answer, under the open world assumption, is $co\text{-}\mathcal{NP}$-hard, even when the query has one relational conjunct.*

This theorem is also true when modified to permit constraints of the form $v < c$ and $v > c$ rather than $v \neq$

c. In addition, the next theorem shows that constants are not the source of complexity.

Theorem 5 *For conjunctive queries with inequality, for conjunctive views, the view complexity of determining whether a tuple is a certain answer under the open world assumption, is co-$\mathcal{NP}$-hard, even when the query has one relational conjunct and no constants.*

Conclusions

We have shown how to extend data integration systems to incorporate data webs. We define a formalism for modeling data webs and a language for source descriptions (GLAV) that make querying multiple web-structured sources possible. In addition, GLAV pushes the envelope of expressive power with efficient reasoning. We present an algorithm for answering queries using GLAV source descriptions that can be used independently of data webs.

One important line of future work is to extend the query-answering algorithm and complexity results to the case where additional constraints are stated on the mediated schema using description logics, using techniques described in (Calvanese, Giacomo, & Lenzerini 1998).

References

Abiteboul, S., and Duschka, O. 1998. Complexity of answering queries using materialized views. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*.

Adali, S.; Candan, K.; Papakonstantinou, Y.; and Subrahmanian, V. 1996. Query caching and optimization in distributed mediator systems. In *Proc. of ACM SIGMOD Conf. on Management of Data*.

Ashish, N., and Knoblock, C. A. 1997. Wrapper generation for semi-structured internet sources. *SIGMOD Record* 26(4):8–15.

Beeri, C.; Elber, G.; Milo, T.; Sagiv, Y.; O.Shmueli; N.Tishby; Y.Kogan; D.Konopnicki; Mogilevski, P.; and N.Slonim. 1998. Websuite-a tool suite for harnessing web data. In *Proceedings of the International Workshop on the Web and Databases*.

Calvanese, D.; Giacomo, G. D.; and Lenzerini, M. 1998. On the decidability of query containment under constraints. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*.

Cohen, W. 1998a. A web-based information system that reasons with structured collections of text. In *Proc. Second Intl. Conf. Autonomous Agents*, 400–407.

Cohen, W. 1998b. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. of ACM SIGMOD Conf. on Management of Data*.

Duschka, O.; Genesereth, M.; and Levy, A. 1999. Recursive query plans for data integration. *To appear in Journal of Logic Programming, special issue on Logic Based Heterogeneous Information Systems.*

Florescu, D.; Levy, A.; Manolescu, I.; and Suciu, D. 1999. Query optimization in the presence of limited access patterns. In *Proc. of ACM SIGMOD Conf. on Management of Data*.

Florescu, D.; Raschid, L.; and Valduriez, P. 1996. A methodology for query reformulation in CIS using semantic knowledge. *Int. Journal of Intelligent & Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems* 5(4).

Friedman, M., and Weld, D. 1997. Efficient execution of information gathering plans. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Garcia-Molina, H.; Papakonstantinou, Y.; Quass, D.; Rajaraman, A.; Sagiv, Y.; Ullman, J.; and Widom, J. 1997. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems* 8(2):117–132.

Haas, L.; Kossmann, D.; Wimmers, E.; and Yang, J. 1997. Optimizing queries across diverse data sources. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*.

Ives, Z.; Florescu, D.; Friedman, M.; Levy, A.; and Weld, D. 1999. An adaptive query execution system for data integration. In *Proc. of ACM SIGMOD Conf. on Management of Data*.

Knoblock, C. A.; Minton, S.; Ambite, J. L.; Ashish, N.; Modi, P. J.; Muslea, I.; Philpot, A. G.; and Tejada, S. 1998. Modeling web sources for information integration. In *Proceedings of the 15th National Conference on Artificial Intelligence*.

Kushmerick, N.; Doorenbos, R.; and Weld, D. 1997. Wrapper induction for information extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*.

Lambrecht, E., and Kambhampati, S. 1998. Optimization strategies for information gathering plans. TR 98-018, Arizona State University Department of Computer Science.

Levy, A. Y.; Rajaraman, A.; and Ordille, J. J. 1996. Query answering algorithms for information agents. In *Proceedings of AAAI*.

Mecca, G.; Mendelzon, A. O.; and Merialdo, P. 1998. Efficient queries over web views. In *Proc. of the Conf. on Extending Database Technology (EDBT)*.