Optimized Querying of Integrated Data over the Web

Andrea Calì and Diego Calvanese
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
lastname@dis.uniroma1.it

Abstract: Information Integration is the problem of providing a uniform access to multiple and heterogeneous data sources. The most common approach to this task, called *global-as-view*, consists in providing a global schema of the data, in which each relation is defined as a view over a set of data sources. Recent works deal with this problem in the case of limited source capabilities, where, in general, sources can only be accessed respecting certain binding patterns for their attributes. In this case, computing the answer to a user query over the global schema cannot be done by simply substituting the concepts appearing in the query with their definitions. Instead, it may require the evaluation of a suitable recursive Datalog program.

In this paper we study the evaluation of conjunctive queries in the global-as-view approach with limited source capabilities. We first present an algorithm for optimizing query answering which takes into account the structure of the query together with the binding patterns in order to compute an optimized query plan. The optimization allows for excluding from the query plan the sources that are not relevant for the answer. We then study online optimization of query answering by taking into account full inclusion and functional dependencies between sources. Such an optimization, at a certain step of the answering process, uses the dependencies together with the data retrieved so far to avoid unnecessary accesses to the sources.

Keywords: Data integration, global-as-view, query planning

1. INTRODUCTION

Information Integration is the problem of combining data residing at different, heterogeneous sources, by providing the user with a uniform access to the data (Hull, 1997). The integration system provides an integrated, reconciled view of the data, usually called *global schema*, in terms of which user queries are formulated. Thus the user is freed from the knowledge on where the data

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35614-3_21
C. Rolland et al. (eds.), *Engineering Information Systems in the Internet Context*
© IFIP International Federation for Information Processing 2002

are, how data are structured at the sources, and how the sources have to be accessed.

To specify the relation between the (materialized) data sources and the (virtual) global schema, two basic approaches have been used, called *Global-as-View* (GAV) and *Local-as-View* (LAV)) respectively (Ullman, 1997; Levy, 1999; Chawathe et al., 1994; Hull and Zhou, 1996; Abiteboul and Duschka, 1998; Levy et al., 1995). In the GAV approach, the global schema is expressed in terms of the data sources, by associating to every relation of the mediated schema, a view over the data sources specifying its meaning in terms of the data at the sources. In the LAV approach, the global schema is specified independently from the sources, and each source is defined as a view over the global schema.

When a user query is posed, the data integration system constructs a *query plan* with which sources are accessed; then the answers are assembled together in order to issue the final answer to the user. In the GAV approach the generation of the query plan is generally considered simple because usually it can be performed by *unfolding* the original query by replacing each global relation with the corresponding view (Ullman, 1997).

Information integration has typically been addressed in the relational setting, which is the one we consider in our paper. We assume that non-relational data sources are accessed via special programs, called *wrappers*, that export a relational view of the data.

A prominent application field for information integration is the integration of data over the Web (Florescu et al., 1998). Often in that context, no direct access to the underlying database is provided. Data are accessible only via forms, where typically certain fields are required to be filled in by the user in order to obtain a result. Limitations on how sources can be accessed significantly complicate query processing (Rajaraman et al., 1995; Li and Chang, 2000; Florescu et al., 1999; Duschka and Levy, 1997), since in this case simply unfolding the global relations in the query with their definitions is in general not sufficient. As shown in (Rajaraman et al., 1995; Li and Chang, 2000; Li and Chang, 2001), query answering in the presence of limited access patterns in general requires the evaluation of a recursive query plan, which can be suitably expressed in Datalog.

Since source accesses are costly, an important issue is how to minimize the number of accesses to the sources while still being guaranteed to obtain all possible answers to a query. (Li and Chang, 2000; Li and Chang, 2001) discuss several optimizations that can be made at compile time, during query plan generation. However, the presented techniques are not applicable in the case where user queries and view definitions are arbitrary conjunctive queries.

Moreover, an important point that has not been addressed before, is whether one can optimize query-plans at run-time, possibly exploiting additional in-

formation available about the sources. Indeed, by exploiting knowledge about integrity constraints present on the sources, one may detect during query evaluation that a certain access to a source is useless, in the sense that it may provide only answers that are already known from previous accesses. Relevant classes of integrity constraints that may be considered are functional dependencies and full inclusion dependencies (see, e.g., (Abiteboul et al., 1995), Chapters 8 and 9).

In this paper we address the problem of query plan optimization for sources with limited capabilities from various points of view. In particular we obtain the following results:

1. We propose a technique to optimize a query plan at the time of its generation. Such an optimization technique exploits the knowledge about the structure of the query and the binding patterns of the sources to compute a query plan which eliminates dependencies between sources and thus avoids unnecessary accesses which may be performed at query plan execution time. Moreover, it allows to exclude from the query plan those sources that cannot contribute to the result of the query. The technique is applicable in the case where both user queries and view definitions are given in terms of *conjunctive queries* (CQs).

2. We address the problem of run-time query plan optimization in the case where both the queries defining relations in the global schema and the user query are *conjunctive queries*, and constraints over sources are asserted. We model such constraints by means of functional and full inclusion dependencies, and we show that the implication problem for such dependencies is decidable in polynomial time. We present a necessary and sufficient condition to determine, given the dependencies, whether during query evaluation a given source has to be accessed or not.

After having constructed an efficient query plan by result (1), by result (2) we can derive all possible full inclusion and functional dependencies that hold for a set of sources, and exploit such dependencies for run-time query plan optimization.

The rest of the paper is organized as follows. In Section 2 we present the technical preliminaries. In Section 3 we present a technique to construct an optimized query plan for query answering. In Section 4 we discuss implication of functional and full inclusion dependencies. In Section 5 we present the condition for minimizing run-time source accesses and prove its correctness and completeness. Finally, in Section 6 we conclude the paper.

2. PRELIMINARIES

We present the formal framework in which we address query optimization. We address information integration in the *global-as-view* (GAV) approach, where sources have access limitations, and both the queries establishing the mapping between relations in the global schema and the sources and the user query are conjunctive queries (CQs). A *conjunctive query* q of arity n over a set $\mathcal{R}$ of relations is written in the form

$$q(X_1, \ldots, X_n) \leftarrow conj(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$$

where $conj(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ is a conjunction of atoms involving the variables $X_1, \ldots, X_n, Y_1, \ldots, Y_m$ and constants, and the predicate symbols of the atoms are in $\mathcal{R}$. We assume that the query is *safe*, i.e., that each variable X_j appears in at least one atom in $conj$.

Given a database DB, the answer q^{DB} of q over DB is the set of tuples $(c_1, \ldots, c_n)$ of constants in DB such that there are constants $d_1, \ldots, d_m$ in DB, such that each atom in $conj(c_1, \ldots, c_n, d_1, \ldots, d_m)$ holds in DB.

To each attribute in a relation we associate a domain, which specifies the legal values for that attribute. Instead of using concrete domains, such as `Integer` or `String`, we deal with *abstract domains*, which have an underlying concrete domain, but represent information at a higher level of abstraction, which is needed to distinguish, e.g., strings representing person names from strings representing plate numbers.

Formally, we have:

- a set $\mathcal{S}$ of *relational sources*, each with an associated *arity*, a tuple of abstract domains, and a *binding pattern*. The binding pattern specifies which subset of the attributes of the relation must be bound by a constant in order to query the source. In the examples we underline the abstract domains in the positions of the source attributes that must be bound.
- a set $\mathcal{G}$ of *global relations*, each with an associated query, which is a CQ over $\mathcal{S}$;
- a *user query*, which is a CQ over $\mathcal{G}$.

The actual data are stored in the sources, whereas the relations in the global schema are not materialized. The user query is specified over the global schema, and in order to answer it, one has to compute a *query plan* specifying how to access the sources.

In the case where the sources do not have access limitation, computing the query plan in the GAV approach amounts to a simple unfolding of the global relations in the query with their definitions. This is shown in the following example, adapted from (Levy, 1999).

Example 1 Suppose we have two sources without access limitations: $\mathsf{s}_1(\mathit{Title}, \mathit{Year}, \mathit{Artist})$, which stores data about songs, and $\mathsf{s}_2(\mathit{Artist}, \mathit{Nation})$, which stores artists with their nationality. Let the global view be defined as follows:

$$\begin{array}{rcl} \mathsf{song}(T, Y, A) & \leftarrow & \mathsf{s}_1(T, Y, A) \\ \mathsf{italian}(A) & \leftarrow & \mathsf{s}_2(A, \mathit{italian}) \end{array}$$

The query

$$\mathsf{q}(T) \leftarrow \mathsf{song}(T, 1998, A),\ \mathsf{italian}(A)$$

asking for titles of songs produced in year 1998 and interpreted by an Italian artist, after unfolding becomes

$$\mathsf{q}'(T) \leftarrow \mathsf{s}_1(T, 1998, A),\ \mathsf{s}_2(A, \mathit{italian})$$

■

In the presence of access limitations on the sources, simple unfolding is in general not sufficient to extract all obtainable answers from the sources, as shown by the following example.

Example 2 Consider again Example 1, with the same sources redefined with access limitations: $\mathsf{s}_1(\mathit{Title}, \underline{\mathit{Year}}, \mathit{Artist})$ and $\mathsf{s}_2(\underline{\mathit{Artist}}, \mathit{Nation})$. In this case, given the same user query q, the unfolded query q' cannot be immediately evaluated over the sources, since s_2 requires the first attribute to be bound to a constant. Therefore, simple unfolding produces an empty answer to q, for any extension of the source relations. However, we could use artist names extracted from s_1 to access s_2 an extract tuples that may contribute to the answer. ■

Given a query over the data sources, an algorithm exists (Li and Chang, 2000) that retrieves all the *obtainable* tuples in the answer to the query. Such an algorithm consists in the evaluation of a suitable Datalog program which extracts all obtainable tuples starting from a set of initial values. The Datalog program is constructed by first unfolding the query in the traditional way, and then encoding in Datalog clauses the limitations on the sources that must be respected during evaluation of the query. The evaluation of the Datalog program is done as follows: starting from the initial values in the query, we access all the sources we can, according to their binding patterns. With the new tuples obtained (if any), we obtain new values with which to access the sources again, getting from them new tuples, and so on, until we have no way of doing accesses with new constants. The program extracts all tuples obtainable while respecting the binding patterns, but there may be tuples in the sources that cannot be retrieved.

3. QUERY PLANNING

Given an unfolded query q over a set of sources $\mathcal{S}$, we want to construct a *query plan* which allows us to retrieve all obtainable answers to q. We present now a technique which allows us to determine which subset of the sources in $\mathcal{S}$ is not relevant for q and to minimize the number of accesses to the relevant sources. On the basis of this optimization, we provide a query plan that provides the best set of answers to the query q. This problem has already been addressed in the case of a subclass of the class of conjunctive queries (Li and Chang, 2000); the improvements provided by our approach are the following:

- our optimization technique considers user queries and queries in the mapping to be in the full class of *conjunctive queries*;
- our technique for determining the relevant sources to a query is refined by exploiting the knowledge about the structure of the query.

In particular, we observe that, having extracted a number of values at a certain point of the query answering process, and given a source s to be accessed using the values extracted so far as bindings, not all the possible accesses to s are necessary in order to calculate the answer to the query. This is illustrated in the following example.

Example 3 Let $\mathcal{S} = \{s_1, s_2, s_3\}$ with

$$s_1(\underline{A}, B)$$
$$s_2(\underline{B}, C)$$
$$s_3(\underline{C}, B)$$

For simplicity, suppose we have a distinct abstract domain for each attribute name. Consider the following unfolded query:

$$q(C) \leftarrow s_1(a_0, B), s_2(B, C)$$

We easily observe that it is not useful to use the values obtained from s_2 to access s_3 in order to obtain new values of domain C with which to access s_2 again. In fact, due to the join condition between s_1 and s_2, the only tuples extracted from s_2 which can be used to construct a tuple of the answer to q are those obtained by binding the attribute B of s_2 with a value extracted from s_1. ■

In order to optimize the query plan, we are interested in keeping only those attribute that can actually contribute to the result of the query. More precisely, we say that an attribute is *relevant* for a query, if there exists an instance of the source database such that the values of the attribute for that instance can

provide values that are used for constructing the answer to the query or for accessing other sources that are in turn relevant.

First of all, we want to operate only with the *queryable* sources, i.e., the sources that can be accessed at least once for at least one instance of at least one source database, starting from the values in the query. For a description of how to calculate the queryable sources we refer to (Li and Chang, 2000), since the method described there still applies to the framework presented here. We exclude a priori all non-queryable sources from the construction of the optimized query plan.

We show how to construct a query plan for a CQ over the sources. To do so, we define the *dependency graph* of q wrt a set $\mathcal{S}$ of sources, denoted by $G_q^{\mathcal{S}}$. Such a graph allows us to eliminate unnecessary accesses to the sources.

In order to simplify the construction of the dependency graph, we take into account values appearing in the query as follows. For each value a appearing in q we introduce a new source s_a with a single free attribute, whose domain is that of the value; the content of s_a is the single tuple $\langle a \rangle$. We then replace all occurrences of a in the query with a fresh variable X_a, and we add the conjunct $s_a(X_a)$ to the body of q. The intuition is that a value acts as a source whose content is completely known, and amounts only to the value itself.

The set of nodes of $G_q^{\mathcal{S}}$ is determined as follows. For each atom in q, we have a set of nodes in $G_q^{\mathcal{S}}$, one for each attribute of the corresponding source relation. We call such nodes *black*. Moreover, for each source not appearing in q, we have a set of nodes, one for each attribute of the source. We call such nodes *white*. Both types of nodes have two labels, determined as follows:

- A *binding constraint*, which can be either bound or free. A node is marked as bound if the corresponding attribute is bound, and it is marked as free otherwise.
- An *abstract domain*, which is the one associated to the corresponding attribute.

As for the edges, $G_q^{\mathcal{S}}$ has an edge from a node u_1 to a node u_2 when

- u_1 and u_2 have the same abstract domain,
- u_1 is free, and
- u_2 is bound.

Intuitively, the edges denote dependencies between source attributes, indicating that a source with limited source capabilities needs values which can be retrieved from other sources (or can be in the query).

Our aim is to divide the edges of $G_q^{\mathcal{S}}$ into two types, which we call strong and weak edges. A *weak* edge (u_1, u_2) denotes that (the attribute associated

to) u_1 can provide values to be used in u_2 to retrieve useful tuples; a *strong* edge denotes a stronger dependency, i.e., that *all* the useful tuples that can be retrieved from the source to which u_2 belongs are extracted using *only* values coming from u_1. Intuitively, determining that an edge (u_1, u_2) is strong allows for reducing source accesses, since the bindings of u_2 can be restricted to values coming from u_1. On the graph G_q^S, when a node has an incoming strong edge, then all other incoming edges that are not strong themselves must be deleted.

To determine a maximal set of strong edges (which allows for deleting as many edges as possible), we first determine, based on the structure of the query, a set of edges that are *candidates* to become strong. An edge (u_1, u_2) is a candidate edge if both u_1 and u_2 are black and if in the query the same variable is associated to the corresponding attributes (i.e., the two attributes are joined). Such an edge can become strong only if the source to which u_2 belongs is not to provide arbitrary values to another source.

An edge which is not candidate, and from which no black node is reachable by traversing edges and moving among nodes belonging to the same source, can be deleted. Intuitively, we are avoiding accesses to sources that cannot provide values that can contribute to the answer.

Based on the above observations, we present in Figure 1 an algorithm to determine whether a candidate edge is strong. The algorithm makes use of two mutually recursive functions that perform a (partial) depth-first visit of the graph, having the side-effect of marking edges either as strong or as deleted. Function *isStrong* starts by assuming that the edge is strong and checks whether such an assumption is consistent with the necessary conditions for an edge to be strong. In visiting the graph, other candidate edges are marked as strong (recursive calls to *isStrong*) and certain non candidate edges are deleted (recursive calls to *deleteEdge*), and again the consistency of these assumptions is verified. The function *outEdges* takes a node u as input and returns the set of all edges whose origin is a node in the same source as u.

Since *isStrong* and *deleteEdge* perform a straightforward visit of G_q^S, never visiting an edge twice, they run in polynomial time in the size of G_q^S. Moreover, in order to determine all strong edges, one needs to make successive calls to *isStrong*. If a call returns **true**, edges that have been marked (either strong or deleted) can keep their mark. This is due the fact that "strong-ness" of edges is a monotone property: a strong edge can never become weak due to the fact that some other edge has been marked strong. On the contrary, if a call returns **false** this means that the assumptions about strong and deleted edges made during the visit were not consistent with the conditions on such edges, and hence need to be retracted.

isStrong $((u, u')$: candidate edge): **bool**
  if (u, u') is marked as strong **then return true**;
  mark (u, u') as strong;
  foreach $(v, v') \in outEdges(u')$
    if ((v, v') is candidate **and not** *isStrong*(v, v')) **or**
      ((v, v') is non-candidate **and not** *deleteEdge*(v, v'))
    then return false;
  return true;

deleteEdge $((u, u')$: non-candidate edge): **bool**
  if (u, u') is marked as deleted **then return true**;
  mark (u, u') as deleted;
  if u' is black
  then choose a candidate edge (v, u');
      if no such edge exists **then return false**;
      if not *isStrong*(v, u') **then return false**;
      mark all candidate edges (v', u') as strong;
      mark all non-candidate edges (v'', u') as deleted;
      return true;
  else foreach $(v, v') \in outEdges(u')$
       if not *deleteEdge*(v, v') **then return false**;
     return true;

Figure 1. Algorithm to determine whether an edge is strong

Example 4 Let $\mathcal{S} = \{s_1, s_2\}$, with

$$s_1(\underline{A}, B)$$
$$s_2(A, \underline{B})$$

Suppose again that we have a distinct abstract domain for each attribute name. Consider the unfolded query q, defined as follows:

$$q(X) \leftarrow s_1(a, X).$$

The dependency graph $\mathcal{G}_q^{\mathcal{S}}$ for q is shown in Figure 2. Note that the source s_a has been added to take into account the value a in q. Edge e_1 is candidate to be strong, and in fact this assumption is consistent, because, if we assume that e_3 is deleted, then e_2 is deleted as well, since no black node is reachable from them. The intuition is that source s_1 does not have to provide arbitrary values to s_2; in fact, due to the join condition in q, accessing s_1 with values provided by s_2 would not provide tuples that could be used to answer the query q. The optimized dependency graph, without deleted edges and without source s_2, is shown in Figure 3; the strong edge e_1 is denoted by a thick line. ■

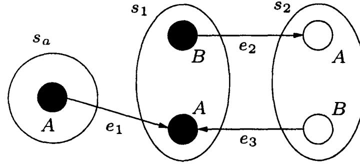

Figure 2. Dependency graph for Example 4

From the resulting optimized dependency graph we construct the optimized query plan, expressed in Datalog notation, as follows: For each source atom over predicate s of arity n in $\mathcal{S}$ (respectively for each source relation s of arity n not appearing in the query), we introduce two predicates s, and $\hat{s}$, both of arity n. The predicate s corresponds to the actual data source, and it has access limitations; the predicate $\hat{s}$ is a sort of *cache* in which we store, during the query answering process, all the tuples extracted from s.

Using these predicates, we construct the optimized query plan as follows.

- The unfolded query mantains its stucture, but it is expressed over the caches $\hat{s}_i$.
- For each predicate s of arity n, we introduce a set of nonrecursive Datalog rules. Each rule corresponds to a choice of exactly one incoming edge in the graph for each of the bound attributes of the atom. The rule has the form

  $$\hat{s}(\mathbf{A}) \leftarrow s(\mathbf{A}), \hat{s_1}(\mathbf{A}_1), \ldots, \hat{s_k}(\mathbf{A}_k)$$

  where

  - $\hat{s_1}(\mathbf{A}_1), \ldots, \hat{s_k}(\mathbf{A}_k)$ are the atoms whose nodes are predecessors of the bound nodes of $s(\mathbf{A})$ wrt to the edges selected in the choice;
  - the variables in the body of the rule reflect the joins expressed by the edges of the graph.

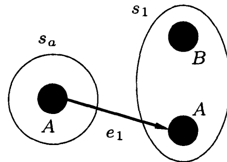

Figure 3. Optimized dependency graph for Example 4

Example 5 We refer to Example 4. From the optimized dependency graph relative to q we construct the following plan:

$$\begin{array}{rcl} q(X) & \leftarrow & \hat{s}_a(X), \hat{s}_1(X,Y) \\ \hat{s}_a(X) & \leftarrow & s_a(X) \\ \hat{s}_1(X) & \leftarrow & s_1(X,Y), \hat{s}_a(X) \end{array}$$

■

Such a Datalog program ensures that the bindings for the bound attributes of s are obtained from values retrieved from the relations s_i and stored in the caches $\hat{s}_i$.

The following theorem states that the optimized query plan is indeed sound, i.e., it returns only tuples that are in the answer to the query, and complete, i.e., it does not miss any answer obtainable from the sources, taking into account the binding patterns.

Theorem 6 *Given an unfolded query q and a set of sources with binding patterns, the Datalog program constructed from q as specified above computes all the answers to q obtainable from the sources, given the binding patterns on them.*

4. FUNCTIONAL AND FULL INCLUSION DEPENDENCIES

We formally introduce the kind of integrity constraints that we use for source access optimization. In the following we denote sets of attributes (i.e., positions) with boldface letters, and we use $\mathcal{A}(s)$ to denote the set of attributes of source s. Given a relation s, a set of attributes $\mathbf{A} \subseteq \mathcal{A}(s)$, and a tuple t in the extension of s, we denote with $t[\mathbf{A}]$ the projection of t over $\mathbf{A}$. Finally, given a database DB, we denote the extension of s in DB with s^{DB}.

A *full inclusion dependency* between two sources s_1 and s_2, which must be of the same arity, has the form

$$s_1 \subseteq s_2$$

Such an inclusion dependency is *satisfied* in a database DB, written $DB \models s_1$, if $s_1^{DB} \subseteq s_1^{DB}$.

A *functional dependency* over a source s has the form

$$s : \mathbf{A} \rightarrow \mathbf{B}$$

with $\mathbf{A}, \mathbf{B} \subseteq \mathcal{A}(s)$. Such a dependency is *satisfied* in a database DB if for any pair of tuples $t_1, t_2 \in s^{DB}$ we have that $t_1[\mathbf{A}] = t_2[\mathbf{A}]$ implies that $t_1[\mathbf{B}] = t_2[\mathbf{B}]$.

Full inclusion dependencies turn out to be essential for modeling the common case of real data sources that can be accessed in different ways, e.g., a database relation that can be accessed from a Web site using different forms. We can represent in our model such a real data source as a set of distinct sources $s_1, \ldots, s_n$, one for each different way of accessing it, with a binding pattern that reflects the access modality. The fact that the sources $s_1, \ldots, s_n$ represent the same data is expressed by means of a pair of full inclusion dependencies $s_i \subseteq s_j$ and $s_j \subseteq s_i$ between each pair of sources s_i and s_j. More generally, by means of an inclusion dependency we can capture the case of a Web site in which a form gives access to a subset of the data contained in the site.

A (full inclusion or functional) dependency γ is *implied* by a set of dependencies Γ, if for every database DB satisfying Γ also γ is satisfied.

We discuss now implication of functional and full inclusion dependencies. The following inference rules are the specialization of the more general sound (but not complete) inference rules for (arbitrary) inclusion and functional dependencies (Cosmadakis and Kanellakis, 1986) to the case where all inclusion dependencies are full. We show that for such a case these inference rules are not only sound but also complete.

1 If $\mathbf{A} \subseteq \mathbf{B}$, with $\mathbf{A}, \mathbf{B} \subseteq \mathcal{A}(s)$, then $s : \mathbf{A} \rightarrow \mathbf{B}$.

2 If $\mathbf{A} \rightarrow \mathbf{B}$, then $\mathbf{AC} \rightarrow \mathbf{BC}$.

3 If $s : \mathbf{A} \rightarrow \mathbf{B}$ and $s : \mathbf{B} \rightarrow \mathbf{C}$, then $s : \mathbf{A} \rightarrow \mathbf{C}$.

4 If $s_1 \subseteq s_2$ and $s_2 \subseteq s_3$, then $s_1 \subseteq s_3$.

5 If $s_1 \subseteq s_2$ and $s_2 : \mathbf{A} \rightarrow \mathbf{B}$, then $s_1 : \mathbf{A} \rightarrow \mathbf{B}$.

We note that the only rule that makes full inclusion and functional dependencies interact is rule 5.

The following theorem shows that functional dependencies do not influence the implication of a full inclusion dependency. Hence, an inclusion dependency can be derived *only* from the set of available inclusion dependencies.

Theorem 7 *Given a set $\mathcal{S}$ of sources with $s, s' \in \mathcal{S}$, a set Γ_i of full inclusion dependencies, and a set Γ_f of functional dependencies, we have that $\Gamma_i \models s_1 \subseteq s_2$ iff $(\Gamma_i \cup \Gamma_f) \models s_1 \subseteq s_2$.*

Next, we show that full inclusion and functional dependencies interact only in a limited form.

Theorem 8 *Let $\mathcal{S} = \{s_0, s_1, \ldots, s_n\}$ be a set of sources, let Γ be a set of full inclusion dependencies of the form $s_0 \subseteq s_i$, and of functional dependencies of the form $r_i : \mathbf{A}_{ij} \rightarrow \mathbf{B}_{ij}$, for $i \in \{0, \ldots, n\}$ and $j \in \{1, \ldots, n_i\}$. Then*

$\Gamma \models r_0 : \mathbf{A} \rightarrow \mathbf{B}$ *if and only if* $\{r_0 : \mathbf{A}_{ij} \rightarrow \mathbf{B}_{ij} \mid i \in \{0, \ldots, n\}$ *and* $j \in \{1, \ldots, n_i\}\} \models r_0 : \mathbf{A} \rightarrow \mathbf{B}$.

Finally, to deal with functional dependencies within one relation we can apply the following theorem (Beeri and Bernstein, 1979; Maier, 1980).

Theorem 9 (Beeri and Bernstein, 1979; Maier, 1980) *The inference rules 1, 2, and 3 are sound and complete for implication of functional dependencies within one relation.*

From the previous results we can prove the following theorem, which, to the best of our knowledge, is not covered by the known results about implication of dependencies (Cosmadakis and Kanellakis, 1986; Cosmadakis et al., 1990).

Theorem 10 *Implication of full inclusion dependencies and functional dependencies can be decided in polynomial time.*

5. ON-LINE OPTIMIZATION

In (Li and Chang, 2000) an optimization technique is presented, which can be applied at query plan generation, and which identifies the sources that are relevant for a query, thus avoiding useless accesses to non-relevant sources.

Instead, here we introduce further optimization techniques, which can be applied during the query evaluation process. Such techniques take into account tuples already extracted from the sources at a certain step of the evaluation of the Datalog program associated to a query. They exploit full inclusion dependencies and functional dependencies on the source relations to know in advance, at a certain step of the evaluation of the Datalog program, whether an access is potentially useful for the answer, i.e. it could return tuples with new values.

Example 11 Suppose we have a source $s(\underline{A_1}, \underline{A_2}, A_3)$ (supposing for simplicity to have a distinct abstract domain for each attribute) with the functional dependency

$$s: A_1 \rightarrow A_2, A_3$$

Let t be a tuple previously extracted from s, having a_1 as component of attribute A_1. Now, if we have another value a_2 with which to bind attribute A_2, and we try to access s using (a_1, a_2) as bindings, the access does not provide any tuple (or provides just t itself if $a_2 = t[A_2]$) because of the functional dependency. ■

We can generalize the observation made in Example 11 with the following result. We denote the bound attributes of a source s with $\mathcal{B}(s)$.

Theorem 12 *Given a source s, let* $\mathbf{B} = \mathcal{B}(s)$ *and let*

$$s : \mathbf{K} \rightarrow \mathcal{A}(s)$$

be a functional dependency over s, with $\mathbf{K} \subseteq \mathbf{B}$*. Let b be a binding for s (i.e., a set of values matching with the attributes of* $\mathbf{B}$*). Then we have that for any database, by accessing s with b, we do not get any tuple which was not previously extracted from s, if and only if there exists a tuple t, previously extracted from s, such that* $b[\mathbf{K}] = t[\mathbf{K}]$.

Proof. "$\Leftarrow$" If there exists t extracted from s such that $b[\mathbf{K}] = t[\mathbf{K}]$, all tuples that may be extracted from s using b have the same values over $\mathbf{K}$, due to the fact that $\mathbf{K} \subseteq \mathbf{B}$. Then, because of the functional dependency $s : \mathbf{K} \rightarrow \mathcal{A}(s)$, any tuple extracted from s using b has the same values as t over $\mathcal{A}(s)$, i.e., it coincides with t.

"$\Rightarrow$" Accessing s using b as binding, we obtain at most one tuple, being $\mathbf{K}$ the key of s and $\mathbf{K} \subseteq \mathbf{B}$. If we obtain no tuple, the theorem is true. If we obtain one, this must have been previously extracted from s, and it is the tuple t we were searching for. □

Now we illustrate a further optimization technique which exploits also full inclusion dependencies.

Example 13 Suppose we have the following sources:

$$s_2(\mathit{Code}, \mathit{Surname}, \underline{\mathit{City}})$$
$$s_1(\underline{\mathit{Code}}, \underline{\mathit{Surname}}, \mathit{City})$$

where s_1 stores data about employees and s_2 stores data about persons, with $s_1 \subseteq s_2$. The attribute with domain *City* represents the city where the corresponding person (or employee) lives. We also have the functional dependency

$$\mathit{Code} \rightarrow \mathit{City}, \mathit{Surname}$$

on both s_2 and s_1. Suppose that s_1 and s_2 have both the following extension:

Code	*Surname*	*City*
2	*brown*	*sidney*
5	*williams*	*london*
7	*yamakawa*	*kyoto*
1	*wakita*	*kyoto*
9	*marietti*	*rome*

If our set of initial values is *rome* and *kyoto*, at the first step we access s_2 and we get the following tuples:

Code	*Surname*	*City*
7	*yamakawa*	*kyoto*
1	*wakita*	*kyoto*
9	*marietti*	*rome*

Now we have six new values: the three codes 1, 7, and 9 and the three surnames *yamakawa*, *wakita*, and *marietti*. With these values we could access source s_2 to try and get other values (we may get only cities, as other attributes are bound). But we can easily observe that, because of the functional dependency cited above, if we bind the attribute with domain *Code* with one of the known values, we get a tuple we had already obtained from s_2. Therefore the access to s_1 is useless, once we have accessed s_2. Instead, if we get a code 2 and a surname *brown* from another source, we could access s_2 and get new tuples. ∎

The following theorem provides a characterization, in terms of a necessary and sufficient condition, of the source accesses that are useless, in the sense that they do not provide new values.

Theorem 14 *Given two sources s_1 and s_2, let $\mathbf{B}_1 = \mathcal{B}(s_1)$ and $\mathbf{B}_2 = \mathcal{B}(s_2)$. Suppose we have the following dependencies:*

$$s_1 \subseteq s_2$$
$$s_1 : \mathbf{C} \rightarrow \mathbf{D}$$

with $\mathbf{C} \subseteq \mathbf{B}_1$ and $\mathbf{D} \supseteq \mathbf{B}_2$. Let b be a binding for s_1 (i.e., a set of values matching with the attributes of $\mathbf{B}_1$). Then we have that, for any database, by accessing s_1 with b, we do not obtain any tuple which was not previously extracted from s_2 if and only if there exists a tuple t, previously extracted from s_2, such that $b[\mathbf{C}] = t[\mathbf{C}]$.

Proof. "$\Leftarrow$" If there exists a tuple t extracted from s_2 such that $b[\mathbf{C}] = t[\mathbf{C}]$, then, being $\mathbf{C} \subseteq \mathbf{B}_1$ and because of the dependency $s_1 : \mathbf{C} \rightarrow \mathbf{D}$, we have that any tuple t' extracted from s_1 using b is such that $t'[\mathbf{D}] = t[\mathbf{D}]$. Now, being $\mathbf{B}_2 \subseteq \mathbf{D}$, we obviously have that $t'[\mathbf{B}_2] = t[\mathbf{B}_2]$. Observe that, as t was extracted from s_2, all tuples of s_2 coinciding with t on $\mathbf{B}_2$ have been extracted as well. Due to the inclusion dependency $s_1 \subseteq s_2$, we can conclde that every tuple t' that we may get from s_1 using b as binding was already extracted from s_2.

"$\Rightarrow$" If we access s using b as binding, we get in general a set of tuples. If this set is empty, the theorem is proved. Instead, consider any tuple t' extracted from s_1 using b as binding; obviously, $t'[\mathbf{C}] = b[\mathbf{C}]$. By hypothesis t' must

have been previously extracted from s_2, and thus it is the tuple we are searching for. □

From the previous theorem we can prove the following upper bound for verifying whether a source access may be useful for getting new tuples.

Theorem 15 *One can check in polynomial time in the number of functional and full inclusion dependencies and the number of attributes in all sources, whether accessing a source with a certain tuple of values may provide new tuples.*

6. CONCLUSIONS

We have studied the problem of query planning in the global-as-view approach, where user queries and view definitions are CQs, and in the presence of source access limitation. We have presented a novel query planning technique, applicable to conjunctive queries, in order to obtain an optimized query plan which exploits the structure of the user query to avoid unnecessary accesses to the sources. We have provided a polynomial time algorithm for implication of full inclusion dependencies and functional dependencies, which we use to model data sources accessible through Web forms. We have shown that the presence of such kinds of dependencies allows one to avoid unnecessary accesses to the sources, and we have provided a necessary and sufficient condition to optimize the query evaluation process at query evaluation time.

We are currently implementing the query planning and query evaluation algorithms, and we are working on incorporating the proposed optimization techniques in the query evaluation phase.

REFERENCES

Abiteboul, S. and Duschka, O. (1998). Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 254–265.

Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts.

Beeri, C. and Bernstein, P. A. (1979). Computational problems related to the design of normal form relational schemas. *ACM Trans. on Database Systems*, 4(1):30–59.

Chawathe, S. S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. D., and Widom, J. (1994). The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. of the 10th Meeting of the Information Processing Society of Japan (IPSJ'94)*, pages 7–18.

Cosmadakis, S. S. and Kanellakis, P. C. (1986). Functional and inclusion dependencies - A graph theoretical approach. In Kanellakis, P. C. and Preparata, F. P., editors, *Advances in Computing Research, Vol. 3*, pages 163–184. JAI Press.

Cosmadakis, S. S., Kanellakis, P. C., and Vardi, M. (1990). Polynomial-time implication problems for unary inclusion dependencies. *J. of the ACM*, 37(1):15–46.

Duschka, O. M. and Levy, A. Y. (1997). Recursive plans for information gathering. In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pages 778–784.

Florescu, D., Levy, A., and Mendelzon, A. (1998). Database techniques for the World-Wide Web: A survey. *SIGMOD Record*, 27(3):59–74.

Florescu, D., Levy, A. Y., Manolescu, I., and Suciu, D. (1999). Query optimization in the presence of limited access patterns. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 311–322.

Hull, R. (1997). Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*.

Hull, R. and Zhou, G. (1996). A framework for supporting data integration using the materialized and virtual approaches. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 481–492.

Levy, A. Y. (1999). Answering queries using views: A survey. Technical report, University of Washinghton.

Levy, A. Y., Mendelzon, A. O., Sagiv, Y., and Srivastava, D. (1995). Answering queries using views. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*, pages 95–104.

Li, C. and Chang, E. (2000). Query planning with limited source capabilities. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 401–412.

Li, C. and Chang, E. (2001). On answering queries in the presence of limited access patterns. In *Proc. of the 8th Int. Conf. on Database Theory (ICDT 2001)*, pages 219–233.

Maier, D. (1980). Minimum covers in the relational database model. *J. of the ACM*, 27(4):664–674.

Rajaraman, A., Sagiv, Y., and Ullman, J. D. (1995). Answering queries using templates with binding patterns. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*.

Ullman, J. D. (1997). Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer.