Optimizing Query Planning with Limited Source Capabilities in the Presence of Inclusion and Functional Dependencies

Andrea Calì, 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 problem, called *Global-as-View*, consists in providing a global schema of data in which each relation of such a schema 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 query over the global schema cannot be done by simply unfolding the global relations with their definitions. Instead, it may require the evaluation of a suitable recursive datalog program. In this paper we study query evaluation in the Global-as-View approach with limited source capabilities in the presence of full inclusion and functional dependencies between sources. We present a polynomial time algorithm for implication of such kind of dependencies and we study how the presence of such dependencies enables one to minimize the number of accesses to the sources. We provide necessary and sufficient conditions for determining whether, during query evaluation, a given source has to be accessed or not.

1 Introduction

Information Integration is the problem of combining the data residing at different, heterogeneous sources, by providing the user with a uniform access to the data [8]. 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 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 [16, 10]. 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.

The LAV approach makes the maintenance of the integration system easier, since the addition of new sources does not require to modify the global schema and the queries associated to existing sources. On the other hand, processing queries over the global schema is complicated and costly [16,

1]: one possible approach is to first reformulate (rewrite) the query to obtain a *query plan* expressed in terms of the sources and then use such a query plan to access the sources and extract the data [11].

On the contrary, in the GAV approach, adding a new source typically requires changing the definition of the relations in the global schema, and hence may affect the whole integration system. The fundamental advantage of GAV lies however in the simplicity of query planning: one simply *unfolds* the original query by replacing each global relation with the corresponding view [16]. For this reason the GAV approach is in fact the most videly adopted in practice [4, 9], and in the rest of the paper we deal with the GAV approach only.

A prominent application field for information integration is the integration of data over the Web [7]. Often in that context, no direct access to the underlying database is provided. Data is accessible only via forms, where typically certain fields are required and need to be instantiated by the user in order to obtain a result. Consider for example the DBLP site[1], where, in order to obtain a list of publications extracted from an underlying database, either an author or a title has to be specified, while it is not possible to ask directly, e.g., for all publications of a certain conference[2].

Information integration has typically been addressed in the relational setting, which is the one we consider in our paper. To be able to incorporate in such a setting also non-relational data sources, such as web pages, these data sources are typically wrapped, i.e., accessed via special programs, called *wrappers*, that export a relational view of the data. A way to correctly represent the actual way in which the data source has to be accessed, is to associate a *binding pattern* to the relational view of the source [15, 12]. The binding pattern specifies which attributes must be bound when accessing the source. In such a setting a query plan is admissible only if all accesses to the sources specified in the plan respect the binding patterns

Limitations on how sources can be accessed significantly complicate query processing, since in this case simply unfolding the global relations in the query with their definitions is in general not sufficient. In fact, to answer queries over such sources one generally needs to start from a set of constants (provided e.g., by the user who fills a form, or taken from a source without access limitations) to be used to bind attributes. Such bindings are used to access sources and thus obtain new constants which in turn can be used for new accesses. Hence, as shown in [15, 12, 13], 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. [12, 13] discuss several optimizations that can be made at compile time, during query plan generation. However, an important point that has not been addressed before, is whether one can optimize query plans at run-time, possibly exploiting additional information available about the sources. Indeed, 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.

In this paper we address the problem of run-time query plan optimization for sources with limited capabilities in the presence of functional dependencies and full inclusion dependencies (see, e.g., [2], Chapters 8 and 9). *Functional dependencies* allow one to state that a certain set A of

[1] http://dblp.uni-trier.de/

[2] We do not consider here advanced search, which is also supported at DBLP.

attributes in a relation functionally determines another set B of attributes, in the sense that two tuples in the relation that coincide on A also have to coincide on B. They are an important type of integrity constraint, which generalize key constraints and thus are actually present in most relational data sources. *Full inclusion dependencies* are a limited form of inclusion dependencies, which allow one to state that an entire relation is included in another one (having the same arity). Full inclusion dependencies allow one to correctly model a data source accessible through different binding patterns, e.g., bibliographic data at DBLP accessible by author or by title. Such a data source can be modeled by means of two relations with different binding patterns, which are declared equivalent by means of two inclusion dependencies. Observe that this is a very common case which arises each time the same underlying data is accessible, e.g., on the web, via two different forms.

More precisely, we provide the following results:

1. We show that the implication problem for functional and full inclusion dependencies is decidable in polynomial time. This result is, to the best of our knowledge, not covered by the known results about implication of dependencies [6, 5].
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 *unions of conjunctive queries*, and functional and full inclusion dependencies are asserted over source relations. We present a condition to determine whether, during query evaluation, a given source has to be accessed or not, and prove that such a condition is necessary and sufficient.

Thus, by result (1) we can derive all possible full inclusion and functional dependencies that hold for a set of sources, and by result (2) we can 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 and discuss the query planning algorithm for GAV in the presence of binding patterns presented in [12]. In Section 3 we discuss implication of functional and full inclusion dependencies. In Section 4 we present the condition for minimizing run-time source accesses and prove its correctness and completeness. Finally, in Section 5 we conclude the paper.

2 Preliminaries

We present the formal framework in which we address run-time query plan optimization. We first introduce the notion of union of conjunctive queries. A *union of conjunctive queries* (UCQ) q of arity n over a set $\mathcal{R}$ of relations is written in the form

$$\begin{aligned} q(X_1, \ldots, X_n) &\leftarrow body_1(X_1, \ldots, X_n, Y_{1,1}, \ldots, Y_{1,n_1}) \\ &\vdots \\ q(X_1, \ldots, X_n) &\leftarrow body_k(X_1, \ldots, X_n, Y_{k,1}, \ldots, Y_{k,n_k}) \end{aligned}$$

where for each i, $body_i(X_1, \ldots, X_n, Y_{i,1}, \ldots, Y_{i,n_i})$ is a conjunction of atoms involving the variables $X_1, \ldots, X_n, Y_{i,1}, \ldots, Y_{i,n_i}$ and constants, and the predicate symbols of the atoms are in $\mathcal{R}$.

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, for some $i \in \{1, \ldots, k\}$, there are constants $d_1, \ldots, d_{n_i}$ in DB, such that each atom in $body_i(c_1, \ldots, c_n, d_1, \ldots, d_{n_i})$ holds in DB.

We address information integration in the GAV approach, where sources have access limitations and both the queries defining the relations in the global schema and the user query are UCQs. We use positional notation for relations, identifying attributes of relations with their positions. To each attribute (position) in a relation we associate an *abstract domain*, which specifies the legal values for that attribute. Abstract domains have underlying concrete domains (which we do not deal with in this paper), such as `Integer` or `String`, but they 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 is the subset of the attributes of the relation which must be bound by a constant in order to query the source. In the examples we specify relations with their abstract domains, and 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 an UCQ over $\mathcal{S}$;
- a *user query*, which is an UCQ 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 [10].

Example 1. Suppose we have the set of sources $\mathcal{S} = \{\mathsf{s}_1(\mathit{Title}, \mathit{Year}, \mathit{Artist}), \mathsf{s}_2(\mathit{Artist}, \mathit{Nation})\}$, where s_1 stores data about Italian songs, while s_2 stores artists with their nationality. Let the global view be defined as follows:

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

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

$$\begin{aligned} \mathsf{q}(T) &\leftarrow \mathsf{s}_1(T, 1998, A),\ \mathsf{s}_1(T, Y, A) \\ \mathsf{q}(T) &\leftarrow \mathsf{s}_1(T, 1998, A),\ \mathsf{s}_3(A, N),\ N = \mathit{italian} \end{aligned}$$

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. In this examples and in the following ones we assume that for each attribute of a relation an abstract domain with the same name is defined.

Example 2. Suppose we have two sources

$$\mathsf{s}_1(\underline{Owner}, Model, Color, PlateNo)$$
$$\mathsf{s}_1(Owner, Address, Rooms)$$

Source s_1 stores information about cars: given a person (required constant), s_1 provides model, plate number, and color of the cars owned by the person. Source s_2 provides the owner, address, and number of rooms of houses. The global schema is the following:

$$\mathsf{Car}(O, M, C, P) \leftarrow \mathsf{s}_1(O, M, C, P)$$
$$\mathsf{House}(O, A) \leftarrow \mathsf{s}_2(O, A, N)$$

Suppose we are searching for all the plate numbers of cars whose model is Ferrari by means of the following query:

$$\mathsf{q}(P) \leftarrow \mathsf{Car}(O, ferrari, C, P)$$

The unfolding of such a query generates a query plan involving only source s_1; such a plan yields the empty answer since to access source s_1 we need owner names. However, it is possible to obtain an answer to q by retrieving owner names from s_2, using them to query s_1, and selecting from the obtained tuples those in which the car model is Ferrari.

The example shows how an apparently useless source may provide constants with which useful information can be retrieved.

In [12] an algorithm is given which, given a UCQ over the data sources, retrieves all the *obtainable* tuples in the answer to the query. The algorithm consists in the evaluation of a suitable Datalog program which extracts all obtainable tuples starting from a set of initial bindings. 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. We illustrate how to construct the Datalog program associated to a query and how to evaluate it over the sources by means of an example, and refer to [12] for the details.

Example 3. Let us consider the following source relations

$$\mathsf{s}_1(\underline{A}, B)$$
$$\mathsf{s}_2(A, \underline{B})$$
$$\mathsf{s}_3(A, \underline{B}, C)$$

and the following unfolded query:

$$\mathsf{q}(C) \leftarrow \mathsf{s}_1(a_1, B),\ \mathsf{s}_3(a_1, B, C)$$
$$\mathsf{q}(C) \leftarrow \mathsf{s}_2(a_1, B),\ \mathsf{s}_3(a_1, B, C)$$

The datalog program corresponding to q is shown in Figure 1. It makes use of the auxiliary predicates

- $\mathsf{domA}(A)$, $\mathsf{domB}(B)$, and $\mathsf{domC}(C)$, called *domain predicates*, which represent the abstract domains of the sources;
- $\hat{\mathsf{s}}_1$, $\hat{\mathsf{s}}_2$, and $\hat{\mathsf{s}}_3$, which store the tuples extracted from s_1, s_2, and s_3.

$$
\begin{aligned}
\mathsf{q}(C) &\leftarrow \mathsf{s}_1(a_1, B), \mathsf{s}_3(a_1, B, C) \\
\mathsf{q}(C) &\leftarrow \mathsf{s}_2(a_1, B), \mathsf{s}_3(a_1, B, C) \\
\hat{\mathsf{s}}_1(A, B) &\leftarrow \mathsf{domA}(A), \mathsf{s}_1(A, B) \\
\mathsf{domA}(A) &\leftarrow \mathsf{domA}(A), \mathsf{s}_1(A, B) \\
\hat{\mathsf{s}}_2(A, B) &\leftarrow \mathsf{domB}(B), \mathsf{s}_2(A, B) \\
\mathsf{domA}(A) &\leftarrow \mathsf{domA}(A), \mathsf{s}_1(A, B) \\
\hat{\mathsf{s}}_3(A, B, C) &\leftarrow \mathsf{domB}(B), \mathsf{s}_3(A, B, C) \\
\mathsf{domA}(A) &\leftarrow \mathsf{domB}(B), \mathsf{s}_3(A, B, C) \\
\mathsf{domC}(C) &\leftarrow \mathsf{domB}(B), \mathsf{s}_3(A, B, C) \\
\mathsf{domA}(a_1) &\leftarrow
\end{aligned}
$$

Fig. 1. Datalog program that answers the query of Example 3

Observe that the $\hat{\mathsf{s}}_i$ predicates correspond to the source predicates but have no access limitations.

The evaluation of the Datalog program is done as follows: starting from the initial bindings 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 constants 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 bindings. At each step, the constants obtained so far are stored in the domain relations. The program extracts all tuples obtainable respecting the binding patterns, but there may be tuples in the sources that cannot be retrieved.

For the example, assume the sources have the extension shown in Figure 2. Starting from a_1, the only constant in the query, we access s_1 getting the tuples (a_1, b_1) and (a_1, b_2). Now we have b_1 and b_2 with which to access s_1 and s_2; from s_2 we get (a_2, b_1), while from s_2 we get nothing. With the new constant a_2 we access s_1 getting (a_2, b_3). Finally, we access s_3 with b_3 getting (a_4, b_3, c_1) (with b_3 we do not get any tuple from s_2). At this point, we have populated the relations $\hat{\mathsf{s}}_i$, from which we evaluate the query with the first two rules. The answer to q is therefore the tuple (c_1). Observe that (a_2, b_4) and (a_3, b_1) could not be extracted from s_2.

s_1 :

a_1	b_1
a_1	b_2
a_2	b_3
a_3	b_1

s_2 :

a_2	b_1
a_2	b_4

s_2 :

a_4	b_3	c_1

Fig. 2. Extension of sources of Example 3

3 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 $DB \models s_1 \subseteq s_2$ if $s_1^{DB} \subseteq s_2^{DB}$.

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 [6] 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 $s : \mathbf{B} \rightarrow \mathbf{A}$.
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 1. *Given a set $\mathcal{S}$ of sources with $s_1, s_2 \in \mathcal{S}$, a set Γ_i of full inclusion dependencies, and a set Γ_f functional dependencies, we have that $\Gamma_i \models s_1 \subseteq s_2$ iff $(\Gamma_i \cup \Gamma_f) \models s_1 \subseteq s_2$.*

Proof.

"$\Rightarrow$" Trivial.

"$\Leftarrow$" We show that (a) implies (b), where

(a) $\Gamma_i \not\models s_1 \subseteq s_2$, i.e., there exists a database DB such that $DB \models \Gamma_i$ and $DB \not\models s_1 \subseteq s_2$;

(b) $(\Gamma_i \cup \Gamma_f) \not\models s_1 \subseteq s_2$, i.e., there exists a database DB such that $DB \models \Gamma_i$ and $DB \models \Gamma_f$, but $DB \not\models s_1 \subseteq s_2$.

Assuming (a) to be true, we construct a database DB as follows. We start from empty source relations and we put a tuple t in s_2. Then we add t to any source s such that there exists a sequence

$$s \subseteq r_1 \subseteq \cdots \subseteq r_k \subseteq s_2$$

of full inclusion dependencies in Γ_i. At the end, we have a subset of sources containing t; At this point, we are sure that s_1 is empty in our database. In fact, if we had $t \in s_1$ at the end of the construction, we would necessarily have in Γ_i a chain of inclusion dependencies

$$s_1 \subseteq r'_1 \cdots \subseteq \cdots \subseteq r'_h \subseteq s_2$$

and by transitivity of inclusion dependencies [6] this negates (a). Therefore, the database DB we have constructed is such that $DB \models \Gamma_i$ by construction, and $DB \models \Gamma_f$ because each source contains at most one tuple. Thus (b) follows.

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

Theorem 2. *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 $s_i : \mathbf{A}_{ij} \rightarrow \mathbf{B}_{ij}$, for $i \in \{0, \ldots, n\}$ and $j \in \{1, \ldots, n_i\}$. Then $\Gamma \models s_0 : \mathbf{A} \rightarrow \mathbf{B}$ if and only if $\{s_0 : \mathbf{A}_{ij} \rightarrow \mathbf{B}_{ij} \mid i \in \{0, \ldots, n\}$ and $j \in \{1, \ldots, n_i\}\} \models s_0 : \mathbf{A} \rightarrow \mathbf{B}$.*

Proof. Let $\Gamma_0 = \{s_0 : \mathbf{A}_{ij} \rightarrow \mathbf{B}_{ij} \mid i \in \{0, \ldots, n\}$ and $j \in \{1, \ldots, n_i\}\}$

"$\Leftarrow$" It suffices to show that for each $i \in \{0, \ldots, n\}$ and $j \in \{1, \ldots, n_i\}$ we have that $\Gamma \models s_0 : \mathbf{A}_{ij} \rightarrow \mathbf{B}_{ij}$. This follows directly from inference rule 5.

"$\Rightarrow$" Suppose that $\Gamma \models s_0 : \mathbf{A} \rightarrow \mathbf{B}$ and $\Gamma_0 \not\models s_0 : \mathbf{A} \rightarrow \mathbf{B}$. Then we can construct a database DB by first putting two tuples t_1, t_2 in s_0^{DB} such that $t_1[\mathbf{A}] = t_2[\mathbf{A}]$, $t_1[\mathbf{B}] \neq t_2[\mathbf{B}]$, and all functional dependencies in Γ_0 are satisfied in DB. The we propagate the tuples t_1 and t_2 to $s_1, \ldots, s_n$ according to the inclusion dependencies in Γ. Now, from the fact that these tuples are the only ones in $s_1, \ldots, s_n$, that $s_0, s_1, \ldots, s_n$ all have the same abstract domains, and that the functional dependencies $s_i : \mathbf{A}_{ij} \rightarrow \mathbf{B}_{ij}$ involve the same attributes as the corresponding $s_0 : \mathbf{A}_{ij} \rightarrow \mathbf{B}_{ij}$, it follows that all dependencies in Γ are satisfied in DB. It follows that $DB \models \Gamma$, but $DB \not\models s_0 : \mathbf{A} \rightarrow \mathbf{B}$, which is a contradiction to $\Gamma \models s_0 : \mathbf{A} \rightarrow \mathbf{B}$.

Finally, to deal with functional dependencies within one relation we can apply the following theorem [3, 14].

Theorem 3 ([3, 14]). *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.

Theorem 4. *Implication of full inclusion dependencies and functional dependencies can be decided in polynomial time.*

Proof. By Theorem 1, implication of an inclusion dependency amounts to reachability on the inclusion dependency graph, which can be decided in NLOGSPACE.

To decide implication of functional dependencies, by Theorem 2, we can first compute the transitive closure of full inclusion dependencies using inference rule 4. Then, we propagate all functional dependencies across full inclusion dependencies using rule 5. Finally, by Theorem 3, we can compute implication of functional dependencies within a single source in polynomial time.

4 On-line Optimization

In [12] 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 a further optimization technique, which can be applied during the query evaluation process. The technique takes into account constants already extracted from the sources at a certain step of the evaluation of the Datalog program Π associated to a query. It exploits full inclusion dependencies and functional dependencies on the source relations to know in advance, at a certain step of the evaluation of Π, whether an access is potentially useful for the answer, i.e. it could return tuples with new constants.

Observe that a full inclusion dependency between two sources can only hold when the two sources have not only the same arity but also the same abstract domains associated to the same attributes.

Example 4. Suppose we have the following sources:

$$\mathsf{person}(\mathit{Code}, \mathit{Surname}, \underline{\mathit{City}})$$
$$\mathsf{employee}(\underline{\mathit{Code}}, \underline{\mathit{Surname}}, \mathit{City})$$

with $\mathsf{employee} \subseteq \mathsf{person}$. 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 person and employee. Suppose that employee and person have both the following extension:

Code	*Surname*	*City*
2	*Brown*	*Sidney*
5	*Williams*	*London*
7	*Yamakawa*	*Tokyo*
1	*Yoshikawa*	*Tokyo*
9	*Peretti*	*Rome*

If our set of initial constants is *Rome* and *Tokyo*, at the first step we access person and we get the following tuples:

Code	*Surname*	*City*
7	*Yamakawa*	*Tokyo*
1	*Yoshikawa*	*Tokyo*
9	*Peretti*	*Rome*

Now we have six new constants: the three codes 1, 7, and 9 and the three surnames *Yamakawa*, *Yoshikawa*, and *Peretti*. With these constants we could access source person to try and get other constants (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 constants, we get a tuple we had already obtained from person. Therefore the access to employee is useless, once we have accessed person. Instead, if we get a code 2 and a surname *Brown* from another source, we could access person 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 constants.

Theorem 5. *Given two sources s_1, s_2 with $s_1 \subseteq s_2$, let $\mathbf{B}_1$ and $\mathbf{B}_2$ be the bound attributes of s_1 and s_2, respectively. Let without loss of generality $\mathbf{B}_1 = \{A_1, \ldots, A_{n_1}\}$ and let $(a_1, \ldots, a_{n_1})$ be an n_1-uple of constants, with a_i matching with domain A_i. Let further t be a tuple previously extracted from s_2 such that $t[\mathbf{B}_1] = (a_1, \ldots, a_{n_1})$.*

We have that, for any database we do not obtain new tuples by accessing s_1 with $(a_1, \ldots, a_{n_1})$, if and only if there exists a functional dependency

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

with

$$\mathbf{C} \subseteq \mathbf{B}_1$$
$$\mathbf{D} \supseteq \mathbf{B}_2$$

Proof. "$\Leftarrow$" Let Θ be a set of tuples of s_2, obtained by binding $\mathbf{B}_2$ one or more times. We want to show that, having a functional dependency $s_1 : \mathbf{C} \rightarrow \mathbf{D}$ as described above, we cannot use a set of constants $(a_1, \ldots, a_{n_1}) = t[\mathbf{B}_1]$, where $t \in \Theta$, to access s_1, and get a tuple $t' \notin \Theta$.

Let t' be a tuple extracted from s_1 by using $(a_1, \ldots, a_{n_1})$ as binding constants for $\mathbf{B}_1$. We have by construction $t'[\mathbf{B}_1] = t[\mathbf{B}_1]$, hence $t'[\mathbf{C}] = t[\mathbf{C}]$, being $\mathbf{C} \subseteq \mathbf{B}_1$. Therefore, because of the functional dependency, we have $t'[\mathbf{D}] = t[\mathbf{D}]$, hence $t'[\mathbf{B}_2] = t[\mathbf{B}_2]$, being $\mathbf{B}_2 \subseteq \mathbf{D}$. Now we observe that t' matches on $\mathbf{B}_2 = \mathcal{B}(s_2)$ with a tuple of Θ, so t' must have been obtained from s_1 at a previous step, that is $t' \in \Theta$.

"$\Rightarrow$" We have that for *any* database, accessing s_1 using $(a_1, \ldots, a_{n_1}) = t[\mathbf{B}_1]$ as binding constants, we do not get any new tuple. We want to show that there is a functional dependency $s_1 : \mathbf{C} \rightarrow \mathbf{D}$ with $\mathbf{C} \subseteq \mathbf{B}_1$ and $\mathbf{D} \supseteq \mathbf{B}_2$. In order to do so, we show that if one of these conditions does not hold, we can always construct an extension of s_1 and s_2, consistent with the dependencies, such that, by accessing s_1, we obtain a tuple $t' \notin \Theta$.

Let us consider the following two cases:

1. If $\mathbf{C} \not\subseteq \mathbf{B}_1$, we put in both s_1 and s_2 the tuple t and a tuple t' such that $t[\mathbf{B}_1] = t'[\mathbf{B}_1]$ and $t[\mathbf{C} - \mathbf{B}_1] \neq t'[\mathbf{C} - \mathbf{B}_1]$; moreover we choose $t[\mathbf{B}_2] \neq t'[\mathbf{B}_2]$.
2. If $\mathbf{D} \not\supseteq \mathbf{B}_2$, we put in both s_1 and s_2 the tuple t and a tuple t' such that $t[\mathbf{D}] = t'[\mathbf{D}]$ and $t[\mathbf{B}_2 - \mathbf{D}] \neq t'[\mathbf{B}_2 - \mathbf{D}]$.

Without loss of generality, we suppose $\Theta = \{t\}$. In both cases, the database we have constructed is consistent with the dependencies, and it is such that accessing s_1 with $t[\mathbf{B}_1]$ we get t', which is not in Θ. This proves our assertion.

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 6. *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 constants may provide new tuples.*

Proof. By Theorem 5 it is sufficient to check whether there exists a functional dependency $s_1 : \mathbf{C} \rightarrow \mathbf{D}$ with $\mathbf{C} \subseteq \mathbf{B}_1$ and $\mathbf{D} \supseteq \mathbf{B}_2$. From inference rules 2 and 3 for functional and full inclusion dependencies this condition is equivalent to $s_1 : \mathbf{B}_1 \rightarrow \mathbf{B}_2$. By Theorem 4 this can be checked in polynomial time.

5 Conclusions

We have studied the problem of query planning in the Global-as-View approach in the presence of source access limitation. We have provided a polynomial time algorithm for implication of full inclusion dependencies and functional dependencies. We have shown that the presence of such kinds of dependencies allows one to avoid unnecessary accesses to the sources, and have provided a necessary and sufficient condition to optimize the query evaluation process.

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

1. Serge Abiteboul and Oliver Duschka. 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, 1998.
2. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., Reading, Massachussetts, 1995.
3. Catriel Beeri and Philip A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Trans. on Database Systems*, 4(1):30–59, 1979.
4. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. of IPSI Conf. (IPSI'94)*, Tokyo (Japan), 1994.
5. S. S. Cosmadakis, P. C. Kanellakis, and M. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *J. of the ACM*, 37(1):15–46, January 1990.
6. Stavros S. Cosmadakis and Paris C. Kanellakis. Functional and inclusion dependencies - A graph theoretical approach. In P. C. Kanellakis and F. P. Preparata, editors, *Advances in Computing Research, Vol. 3*, pages 163–184. JAI Press, 1986.
7. Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.
8. Richard Hull. 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)*, 1997.
9. Richard Hull and Gang Zhou. 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, 1996.
10. Alon Y. Levy. Answering queries using views: A survey. Technical report, University of Washinghton, 1999.
11. Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*, pages 95–104, 1995.
12. Chen Li and Edward Chang. Query planning with limited source capabilities. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 401–412, 2000.

13. Chen Li and Edward Chang. On answering queries in the presence of limited access patterns. In *Proc. of the 8th Int. Conf. on Database Theory (ICDT 2001)*, 2001.
14. David Maier. Minimum covers in the relational database model. *J. of the ACM*, 27(4):664–674, 1980.
15. Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*, 1995.
16. Jeffrey D. Ullman. 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-Verlag, 1997.