

Scalable Query Reformulation Using Views in the Presence of Functional Dependencies

Qingyuan Bai, Jun Hong, and Michael F. McTear

School of Computing and Mathematics, University of Ulster at Jordanstown
Newtownabbey, Co. Antrim, BT37 0QB, UK

{q.bai, j.hong, mf.mctear}@ulster.ac.uk

Abstract. The problem of answering queries using views in data integration has recently received considerable attention. A number of algorithms, such as the bucket algorithm, the SVB algorithm, the MiniCon algorithm, and the inverse rules algorithm, have been proposed. However, integrity constraints, such as functional dependencies, have not been considered in these algorithms. Some efforts have been made in some inverse rule-based algorithms in the presence of functional dependencies. In this paper, we extend the bucket-based algorithms to handle query rewritings using views in the presence of functional dependencies. We build relationships between views containing no subgoal of a given query and the query itself. We present an algorithm which is scalable compared to the inverse rule-based algorithms. The problem of missing query rewritings in the presence of functional dependencies that occurs in the previous bucket-based algorithms is avoided. We prove that the query rewritings generated by our algorithm are maximally-contained rewritings relative to functional dependencies.

1 Introduction

In data integration, user queries over a mediated schema need to be reformulated into queries over the schemas of data sources using source descriptions. There are two main approaches to query reformulation in data integration, i.e., Global As View (GAV for short) and Local As View (LAV for short). In the LAV approach, data sources are described by views that are defined over a mediated schema, and user queries are also posed in terms of the mediated schema. The LAV approach to query reformulation is closely related to the problem of answering queries using views, which has recently received considerable attention because of its relevance to a wide variety of data management problems: query optimization, maintenance of physical data independence, data integration, and data warehouse and web-site design [11]. Informally speaking, the problem is as follows. Suppose we are given a query Q over a database schema, and a set of view definitions V_1, \dots, V_m over the same schema. Is it possible to answer query Q using only the answers to V_1, \dots, V_m , and if so, how?

In data integration, views describe a set of distributed, autonomous, and heterogeneous data sources. Thus, in this context we can usually find only maximally-

contained rewritings that provide the best possible answers for a given query. Many algorithms including the bucket algorithm [12],[13], the inverse rules algorithm [16],[3], the Shared Variables Bucket (SVB for short) algorithm [14], and the MiniCon algorithm [15], have been proposed. These algorithms can generate all the maximally-contained query rewritings in the absence of functional dependencies. However, they have not considered the problem of answering query using views in the presence of functional dependencies in a mediated schema. As a result, they might sometimes miss query rewritings in this case.

Example 1 [4]. Suppose that there are three relations in a mediated schema:

Conference(P, C), Year(P, Y), and Location(C, Y, L),

where, attributes P, C, Y, and L refer to Paper, Conference, Year, and Location respectively. A paper is only presented at a conference and published in a year. Also, in a given year a conference is held at a specific location. Therefore, we have the following functional dependencies:

Conference: $P \rightarrow C$; Year: $P \rightarrow Y$; Location: $C, Y \rightarrow L$.

Suppose that there are two data sources (views):

$V_1(P, C, Y)$:- Conference(P, C), Year(P, Y).

$V_2(P, L)$:- Conference(P, C), Year(P, Y), Location(C, Y, L).

Assume that a query asks where PODS'89 was held:

$Q(L)$:-Location ("PODS", "1989", L).

For the sake of simplicity, we change the query with constants into the following form:

$Q(L)$:-Location (C, Y, L), C= "PODS", Y= "1989".

All the bucket-based algorithms would try to find the mappings between the subgoals of the query and those in views. Thus, V_1 would not appear in any query rewriting generated by these algorithms, because it does not contain the subgoal Location. Therefore all the previous bucket-based algorithms would fail to generate the following correct rewriting in the presence of functional dependencies:

$Q'(L)$:- $V_1(P, C, Y), V_2(P, L), C= "PODS", Y= "1989"$. □

In [4], an algorithm based on use of inverse rules is proposed to solve this problem by using binary relations to describe functional dependencies. The papers [6],[7],[9] also consider query rewriting using views in the presence of functional dependencies. However, no bucket-based algorithm has so far been developed to address this issue. In this paper, we present a bucket-based algorithm to solve this problem. Our main contribution is the extension of the MiniCon algorithm in the presence of functional dependencies.

In Section 2, we have a brief look at the previous algorithms. In Section 3, the preliminaries are given. In Section 4, we first show how our algorithm works using an example and then present our algorithm to handle query rewritings using views in the presence of functional dependencies in a mediated schema. In Section 5, we prove the correctness and computational complexity of our algorithm. At the end of the paper, we conclude and discuss our future work.

2 Related Work

We divide all the algorithms for query rewriting into two categories. The algorithms of the first category are based on use of buckets while the ones of the second category are based on use of inverse rules.

2.1 The Algorithms Based on Use of Buckets

The bucket algorithm [12],[13] proceeds in two stages. Initially, a bucket is created for each subgoal of a query. A view is put in the bucket if it can be unified with a subgoal in the query. Next, candidate query plans are generated by picking one view from each bucket. These plans are then verified using containment tests.

A non-distinguished variable that appears in more than one subgoal of a query is called a shared variable. In the SVB algorithm [14], given a query Q , two types of buckets are created. The first type of buckets, the single-subgoal buckets are built in the same way as the bucket algorithm. The second type of buckets, the shared-variable buckets are created by checking the containment mapping from a set of subgoals in Q containing a shared variable to some subgoals in a view. Once all the buckets are created, the algorithm constructs rewritings by combining views from buckets which contain disjoint sets of subgoals of Q .

The MiniCon algorithm [15] proceeds in principle in the same way as the SVB algorithm. Both algorithms focus on the roles of distinguished variables and shared variables in a query. In the first phase of the MiniCon algorithm, a MiniCon Description (MCD for short) for a query Q over a view V is formed to contain a set of subgoals in Q and the mapping information. The MCDs and the minimum MCDs in the MiniCon algorithm correspond to the single-subgoal buckets and the shared-variable buckets in the SVB algorithm respectively. In the second phase, the MiniCon algorithm combines the MCDs to generate query rewritings. In [15], it is shown that the MiniCon algorithm significantly outperforms the bucket and the SVB algorithms and scales up to hundreds of views.

2.2 The Algorithm Based on Use of Inverse rules

The key idea underlying the inverse rules algorithm [16],[3] is to first construct a set of rules called inverse rules that invert the view definitions, and then replace existential variables in the view definitions with Skolem functions in the heads of the inverse rules. The rewriting of a query Q using the set of views V is simply the composition of Q and the inverse rules for V by the transformation method in [3] or the unification-join method in [16].

A key advantage of the inverse rules algorithm is its conceptual simplicity. The inverse rules can be constructed in advance in polynomial time, independent of a particular query. However, even if the computational cost of constructing rewriting is polynomial, the rewriting contains rules that may cause accessing irrelevant views. The problem of eliminating irrelevant rules has exponential time complexity. Another drawback is that evaluating the inverse rules over the source extension may invert some useful computation done to produce the views. Hence, much of the

computational advantage of exploiting the materialized view is lost due to recomputing the extensions of the database relations.

3 Preliminaries

Queries and views. We consider the problem of answering conjunctive queries using views. A conjunctive query has the form:

$$Q(\bar{X}): -R_1(\bar{X}_1), \dots, R_k(\bar{X}_k)$$

where $R_1(\bar{X}_1), \dots, R_k(\bar{X}_k)$ are the subgoals referred to database relations. $Q(\bar{X})$ is the head of the query. The tuples $\bar{X}, \bar{X}_1, \dots, \bar{X}_k$ contain either variables or constants. We require that the query be safe, i.e., $\bar{X} \subseteq \bar{X}_1 \cup \dots \cup \bar{X}_k$. The variables in \bar{X} are the distinguished variables, and others are existential variables. We use $\text{Vars}(Q)$, $Q(D)$ to refer to all variables in Q and the evaluating result of Q over the database D respectively.

A view is a named query. If the query results are stored, we refer to them as a materialized view, and we refer to the result set as the extension of the view.

Query containment and equivalence. The concepts of query containment and equivalence enable us to compare between queries and rewritings. We say that a query Q_1 is contained in the Q_2 , denoted by $Q_1 \subseteq Q_2$, if the answers to Q_1 are a subset of the answers to Q_2 for any database instance. Containment mappings provide a necessary and sufficient condition for testing query containment. A mapping ϕ from $\text{Vars}(Q_2)$ to $\text{Vars}(Q_1)$ is a containment mapping if

- (1) ϕ maps every subgoal in the body of Q_2 to a subgoal in the body of Q_1 , and
- (2) ϕ maps the head of Q_2 to the head of Q_1 .

The query Q_2 contains Q_1 if and only if there is a containment mapping from Q_2 to Q_1 . The query Q_1 is equivalent to Q_2 if and only if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.

Answering queries using views. Given a query Q and a set of view definitions $V = V_1, \dots, V_m$, a rewriting of Q using the views is a query expression Q' whose body predicates are only from V_1, \dots, V_m .

Note that the views are not assumed to contain all the tuples in their definitions since the data sources are managed autonomously. Moreover, we cannot always find an equivalent rewriting of the query using the views because data sources may not contain all of the answers to the query. Instead, we consider the problem of finding maximally-contained rewritings as follows.

Definition 1 (Maximally-contained rewriting [15]) Q' is a maximally-contained rewriting of a query Q using the views $V = V_1, \dots, V_m$ with respect to a query language L if

1. for any database D , and extensions v_1, \dots, v_m of the views such that $v_i \subseteq V_i(D)$, for $1 \leq i \leq m$, then $Q'(v_1, \dots, v_m) \subseteq Q(D)$ for all i ,

2. there is no other query Q_1 in the language L , such for every database D and extensions v_1, \dots, v_m as above (1) $Q'(v_1, \dots, v_m) \subseteq Q_1(v_1, \dots, v_m)$ and (2) $Q_1(v_1, \dots, v_m) \subseteq Q(D)$, and there exists at least one database for which (1) is a strict subset.

The MiniCon Algorithm [15]. Since our aim in this paper is to extend the MiniCon algorithm in the presence of functional dependencies, let us here have a detailed look at it.

In the MiniCon algorithm, a MCD plays an important role like a bucket in the bucket algorithm. It contains information about the unification and containment of subgoals between a given query and a view.

Definition 2 (MiniCon Descriptions) A MCD C for a query Q over a view V is a tuple of the form $(h_c, V(\bar{Y})_C, \varphi_c, G_c)$ where:

- h_c is a head homomorphism on V ,
- $V(\bar{Y})_C$ is the result of applying h_c to V , i.e., $\bar{Y} = h_c(\bar{A})$, where \bar{A} are the head variables of V ,
- φ_c is a partial mapping from $\text{Vars}(Q)$ to $h_c(\text{Vars}(V))$,
- G_c is a set of subgoals in Q which are covered by some subgoal in $h_c(V)$ and φ_c .

A head homomorphism h on a view V is a mapping h from $\text{Vars}(V)$ to $\text{Vars}(V)$ that is the identity on the existential variables, but may equate distinguished variables.

The MiniCon algorithm proceeds in the following two steps.

Step 1: Forming the minimum MCDs. For each subgoal R of Q and each subgoal R' of view V , find a least restrictive head homomorphism h on V such that there exists a mapping φ such that $\varphi(R) = h(R')$. If h and φ exist, then extend the domain of φ to variables in a minimum set G of subgoals of Q such that,

- (a) G is mapped to a subgoal of $h(V)$ by φ ;
- (b) each head variable in G is mapped to a head variable in $h(V)$;
- (c) if an existential variable x in G is mapped to an existential variable of $h(V)$, then (i) each subgoal of Q involving x is in G ; (ii) all variables in the comparisons B of Q that involve x are in the domain of φ and comparisons of Q and $h(V)$ are consistent.

Step 2: Generating rewritings [17]. If there are minimum MCDs $(h_i, v_i(Y_i), \varphi_i, G_i), i=1, \dots, k$, such that $i \neq j, G_i \cap G_j = \emptyset$, and $G_1 \cup G_2 \cup \dots \cup G_k = \text{subgoals}(Q)$, then

- (1) If φ_i maps two or more variables x_1, \dots, x_s in G_i to the same argument, then choose one of the variable as a representative variable, denote the representative variable of x_j by $EC_i(x_j)$. For each x in Q , if $EC_i(x) \neq EC_j(x)$ ($1 \leq i, j \leq k$), then let $EC(x)$ be one of them but consistently across all y for which $EC_i(y) = EC_i(x)$.
- (2) For each $y \in Y_i$, if exists x such that $\varphi_i(x) = y$, then let $\psi_i(y) = x$, otherwise let $\psi(y)$ be a distinct new variable;
- (3) Create the conjunctive rewriting: $Q'(EC(X)) :- v_1(EC(\psi_1(Y_1))), \dots, v_k(EC(\psi_k(Y_k)))$.

Functional Dependencies. An instance of a relation R satisfies the functional dependency $A_1, \dots, A_n \rightarrow B$ if for every two tuples t and u in R with $t.A_i = u.A_i$ for $i=1, \dots, n$, also $t.B = u.B$. If $A \rightarrow B$ and $B \rightarrow C$ hold, then $A \rightarrow C$ holds.

When the relations satisfy a set of functional dependencies Σ , we define our notion of query containment as query containment relative to Σ . Query Q_1 is contained in query Q_2 relative to Σ , denoted $Q_1 \subseteq_{\Sigma} Q_2$, if for each database D satisfying the functional dependencies in Σ , $Q_1(D) \subseteq Q_2(D)$. Maximal query containment relative to Σ can be defined accordingly.

4 Query Rewriting in the Presence of Functional Dependencies

Now we consider query rewriting in the presence of functional dependencies. In the context of databases, any database schema has some integrity constraints, such as functional dependencies, and/or inclusion dependencies. Thus, this issue has practical significance.

We note that the MiniCon algorithm fails to form a MCD for a given query Q over a view V if

- (1) V contains no subgoal in Q , or
- (2) there is a violation on containment mapping from $\text{Vars}(Q)$ to $\text{Vars}(V)$, for example, a distinguished variable of Q is mapped into an existential variable of V , or
- (3) there is any inconsistency between comparison predicates in Q and ones in V .

In this paper, we address the above problem (1) by making use of functional dependencies. We first use an example to give a detail description about the idea underlying our algorithm, and we then present our algorithm.

Continue with Example 1. The following query rewriting is correct in the presence of functional dependencies:

$Q'(L):- V_1(P,C,Y), V_2(P,L), C = \text{"PODS"}, Y = \text{"1989"}$.

Informally, this query rewriting is generated as follows. It first finds some paper presented at PODS'89 using V_1 , and then finds the location of the conference at which the paper was presented using V_2 . This plan is correct only because every paper is presented at one conference and in one year. Note that V_1 is needed in the query rewriting even though V_1 does not contain any subgoal in Q . In the previous algorithms, only views that contain subgoals appearing in the query are considered. In this example, the MiniCon algorithm would not generate any query rewriting containing V_1 . As a result, it would fail to find the above rewriting. In other words, all bucket-based algorithm encounters the problem of missing query rewritings in the presence of functional dependencies in a mediated schema.

In V_1 , attributes P , C , and Y are distinguished variables and we have in *Conference*, $P \rightarrow C$, in *Year*, $P \rightarrow Y$. There are two constants to appear in the form of $C = \text{"PODS"}$, $Y = \text{"1989"}$ in the query Q . According to the definition of functional dependencies, there are some values (maybe single value) of P in *Conference* for $C = \text{"PODS"}$, and some values of P in *Year* for $Y = \text{"1989"}$. This indicates that V_1 containing *Conference* and *Year* might be useful for answering the query because it can provide information

about attribute P, which is also a distinguished variable in other views given C=“PODS”, Y=“1989”.

We try to find out this kind of relationships between V_1 and Q in the presence of functional dependencies. We first identify views that contain no subgoal of the query and then check whether these views can provide any useful information for the query. Among the subgoals in an identified view there may exist a set of functional dependencies that have transitive relationships with the ones among the subgoals in the query. For example, V_1 contains no subgoal of Q, and there is a set of functional dependencies $\{P \rightarrow C, P \rightarrow Y\}$ on its subgoals. In Q, there is a functional dependency $\{C, Y \rightarrow L\}$. We find that there exists a relationship between V_1 and Q, i.e., implicit functional dependency.

In V_2 , attributes P and L are distinguished variables, but attributes C and Y are shared variables. The MiniCon algorithm can unify the subgoal *Location* in Q with one in V_2 and create a MCD for *Location* over V_2 as follows (here, \rightarrow_m means the mapping through this paper):

V(Y)	h	ϕ	G
$V_2(P,L)$	$P \rightarrow_m P, L \rightarrow_m L$	$P \rightarrow_m P, C \rightarrow_m C, Y \rightarrow_m Y$	Location

However, attributes C and Y in V_2 are not distinguished variables. Therefore, V_2 can not answer Q alone given C=“PODS”, Y=“1989”. Note that attributes P and L are distinguished variables in V_2 and functional dependency $\{P \rightarrow L\}$ holds, which means if some values of P can be passed from V_1 to V_2 , then the value of L can be determined using V_2 . Thus, we can combine V_1 and V_2 to form a query rewriting of Q as follows:

$Q'(L):- V_1(P,C,Y), V_2(P,L), C=“PODS”, Y=“1989”.$ □

Assume that a mediated schema consists of a set of relations R_1, R_2, \dots, R_m . The data sources are described by views V_1, \dots, V_n that are defined in the mediated schema. Without loss of generality, we assume that there exists at most one functional dependency, $A \rightarrow B$ in a relation R, where B is a single attribute in R, A may be a set of attributes in R. We denote a set of functional dependencies in a mediated schema as $\Sigma = \{A_i \rightarrow B_i, A_i, B_i \in R_i, 1 \leq i \leq m\}$.

Given a conjunctive query Q over a mediated schema:

$Q(\bar{X}) :- R_1(\bar{X}_1), R_2(\bar{X}_2), \dots, R_k(\bar{X}_k)$

Assume that there are k_1 ($1 \leq k_1 \leq k$) functional dependencies among R_1, \dots, R_{k_1} , i.e., $\Sigma_Q = \{A_i \rightarrow B_i, A_i, B_i \in R_i, 1 \leq i \leq k_1\}$. Given views V_1, \dots, V_n , we divide them into two groups: $V^1 = \{V_j^1, j=1, 2, \dots, n_1\}$ and $V^2 = \{V_j^2, j=1, 2, \dots, n_2\}$, where each view in V^1 contains at least one subgoal of Q while each view in V^2 contains no subgoal of Q. We check each view V_j^2 ($1 \leq j \leq n_2$) in V^2 to see whether it can make any contribution to answering Q in the presence of functional dependencies.

Assume that there is a set of functional dependencies: $\{C_j \rightarrow D_j, C_j, D_j \in \text{subgoal } R \text{ in the body of } V_j^2, 1 \leq j \leq n_2\}$. If

(1) there exists $\{R_i: A_i \rightarrow B_i\} \in \Sigma_Q$ such that D_j is identical to A_i , and A_i appears in Σ_Q ,

- (2) C_j is a distinguished variable in V_j^2 , and
(3) A_i appears as a constant in Q ,

then, V_j^2 might be used for answering Q . V_j^2 is put into a bucket as follows:

V(Y)	FD	SQ(subgoal of Q)
V_j^2	$C_j \rightarrow A_j$	R_i

For each view V_i^1 in V , $1 \leq i \leq n$, the MCDs are formed using the MiniCon algorithm. The MCDs of the form $(h_C, V(\bar{Y})_C, \varphi_C, G_C)$ are divided into two sets as $C^1 = \{C_j^1, 1 \leq j \leq m_1\}$ and $C^2 = \{C_j^2, 1 \leq j \leq m_2\}$. A MCD is put in C^1 if its component $V(\bar{Y})$ contains the distinguished variables C_j and B_j , where C_j and B_j are distinguished variables in some view, say V_j^2 , in V^e and in Q respectively, and satisfy:

- (4) $C_j \rightarrow D_j$ holds in the body of V_j^2 ,
(5) $A_j \rightarrow B_j$ holds in the body of Q , and
(6) D_j is identical to A_j .

Thus, such a view, say V_i^1 , can participate in join operations on C_j with V_j^2 to get the values of B_j . The rest of the MCDs is put in C^2 .

Now we have a set of buckets for V_j^2 ($j=1,2,\dots$), two sets of the MCDs, C^1 and C^2 . The MCDs in C^2 can be used to generate query rewritings alone. The buckets for V_j^2 ($j=1,2,\dots$) are combined with the MCDs in C^1 to generate other query rewritings.

Algorithm Bucket-Based Query Rewriting in the Presence of Functional Dependencies

Input: A set of the relations in a mediated schema, R_1, R_2, \dots, R_m and a set of functional dependencies $\Sigma = \{A_i \rightarrow B_i, A_i, B_i \in R_i, 1 \leq i \leq m\}$; A set of the views $V = \{V_1, V_2, \dots, V_n\}$; A conjunctive query in the form of

$Q(\bar{X}) : -R_1(\bar{X}_1), R_2(\bar{X}_2), \dots, R_k(\bar{X}_k)$; A set of functional dependencies in the body of Q :

$\Sigma_Q = \{A_i \rightarrow B_i, A_i, B_i \in R_i, 1 \leq i \leq k\}$.

Output: Q 's, the maximally-contained rewritings of Q relative to functional dependency Σ .

Method:

We divide all views into two groups V^1 and V^e as described above.

Step 1: Check whether each view in V^e satisfies the above conditions (1), (2), and (3). If so, the view is put into a bucket defined above.

Step 2: Form a MCD for each view in V^A using the MiniCon algorithm [15]. The MCDs are divided into two sets C^1 and C^2 .

Step 3: Generate all possible query rewritings Q 's. Some query rewritings are generated in the same way as the MiniCon algorithm using the MCDs in C^2 . The others are generated by combining the MCDs in C^1 with the obtained buckets for views in V^B as follows.

- For every subset $\{C_j^1, 1 \leq j \leq m_1\}$ of C^1 such that:
 $G_1^1 \cup G_2^1 \cup \dots \cup G_{m_1}^1 = \text{subgoals}(Q)$ and for every $i \neq j, G_i^1 \cap G_j^1 = \emptyset$.
- For every subset of buckets for views in V^B such that the union of a set of functional dependencies in these buckets and a set of functional dependencies in the MCDs in C^1 can imply the functional dependencies of Q .
- Combine the views in $C_j^1, 1 \leq j \leq m_1$ with the views in buckets in the same way as in the MiniCon algorithm to generate query rewritings. \square

We end this section by the following example which shows the complete procedure of our algorithm.

Example 2. Suppose that there are three data sources:

$V_1(P,C,Y)$:-Conference(P,C),Year(P,Y).

$V_2(P,L)$:-Conference(P,C),Year(P,Y),Location(C,Y,L).

$V_3(C,Y,L)$:-Location(C,Y,L).

Conference, Year, and Location as well as functional dependencies are the same as in Example 1. We also have the same query:

$Q(L)$:-Location (C,Y,L),C= "PODS",Y= "1989".

We divide these views into $V^A = \{V_2, V_3\}$ and $V^B = \{V_1\}$. By checking the above conditions (1)-(3), a bucket is built for V_1 .

Using the MiniCon algorithm, we get the following MCDs over the views in V^A :

$V(Y)$	h	ϕ	G
$V_2(P,L)$	$P \rightarrow_m P, L \rightarrow_m L$	$P \rightarrow_m P, C \rightarrow_m C, Y \rightarrow_m Y$	Location
$V_3(C,Y,L)$	$C \rightarrow_m C, Y \rightarrow_m Y, L \rightarrow_m L$	$C \rightarrow_m C, Y \rightarrow_m Y, L \rightarrow_m L$	Location

The first MCD belongs to C^1 and the second is in C^2 , because in V_2 of the first MCD, P is a distinguished variable in a view in V^B and L is a distinguished variable in Q , and functional dependency $\{P \rightarrow L\}$ holds. Hence, we get a query rewriting of Q from C^2 alone.

$Q'(L)$:- $V_3(C,Y,L)$, C= "PODS",Y= "1989".

Another rewriting is generated by combining V_1 with V_2 .

$Q''(L)$:- $V_1(P,C,Y), V_2(P,L)$,C= "PODS",Y= "1989". \square

5 Computational Complexity and Correctness of Our Algorithm

In this section, we illustrate that our algorithm is a scalable extension of the MiniCon algorithm by taking into account functional dependencies in query reformulation and prove the correctness of our algorithm.

5.1 Computational Complexity of the Algorithm

In the MiniCon algorithm, any views in either \mathcal{V}^l or \mathcal{V}^e are considered for forming the MCDs. In Step 1 of our algorithm, the algorithm checks for each view in \mathcal{V}^e by testing above three conditions (1)-(3). In the worst case the complexity of this process is $n^{|\Sigma_Q| \cdot |\Sigma_{VQ}|}$, where n is the number of the views in the second group, $|\Sigma_Q|$ and $|\Sigma_{VQ}|$ are the numbers of functional dependencies in the query and in each view in \mathcal{V}^e respectively. In Step 2 of our algorithm, the computation is the same as in the MiniCon algorithm when forming the MCDs for each view in \mathcal{V}^l . The computation is increased when dividing the MCDs into two groups. In the worst case the complexity of this process is $n^{|\Sigma_Q| \cdot |\Sigma_{VQ}|}$. In Step 3, the complexity of our algorithm is the same as the MiniCon algorithm when generating the query rewritings from the MCDs in \mathcal{C}^2 . Due to generating additional query rewritings by combining the MCDs in \mathcal{C}^1 and the buckets for views in \mathcal{V}^e , the computation of this step is exponential in terms of $|\mathcal{V}^e| \cdot |\mathcal{C}^1|$. However, in the worst case, the computation of the MiniCon algorithm is exponential in terms of the number of the views. In [15], it is shown that the MiniCon algorithm can scale up to hundreds of views. Our algorithm has the same scale-up.

5.2 Correctness of the Algorithm

Theorem 1. Given a conjunctive query Q , conjunctive views \mathcal{V} , and a set of functional dependencies Σ in a mediated schema, our algorithm produces the union of conjunctive queries that is the maximally-contained rewriting of Q using \mathcal{V} relative to Σ .

Proof: In our algorithm, some query rewritings are generated from the MCDs in \mathcal{C}^2 along in the same way as the MiniCon algorithm. Therefore, these rewritings are maximally-contained rewritings. The others are generated by combining the MCDs in \mathcal{C}^1 with the buckets of views in \mathcal{V}^e . We show that these rewritings are also maximally-contained rewritings.

As shown above, the MCDs of \mathcal{C}^1 are needed to answer the query if the other information is required. We claim that (1) the conditions (1)-(3) show the buckets of views in \mathcal{V}^e have useful information for answering the query, and (2) the conditions (4)-(6) guarantee the combination of the MCDs of \mathcal{C}^1 with the buckets of \mathcal{V}^e in Step 3 of our algorithm can exactly provide the answers to the query. The claim (1) is valid for the following reasons: The conditions (1) and (3) mean that for evaluating attribute A_i in a subgoal R_i of a query, the value of another attribute C_j in a subgoal R_j of view

V is needed even though V does not contain any subgoal of the query. The condition (2) guarantees that the required value of C_j can be transferred to the query because C_j is a distinguished variable. The claim (2) is valid because for the MCDs in C^1 , their components $V(\bar{Y})$ contain the distinguished variables C_j, B_j , and the conditions (4)-(6) guarantee that the transitive property of functional dependencies holds among the query, the views in C^1 and the views in V^e . Therefore, the combination of the MCDs of C^1 with the buckets of V^e in the same way as the MiniCon algorithm can provide the answers to the query in the presence of functional dependencies. According to the proofs given in [15], if there exist some query rewritings, then these rewritings are the maximally-contained rewritings. \square

6 Conclusion and Future Work

In this paper, we have considered the query rewriting problem in data integration by using the approach of answering query using views. The algorithms for this problem can be divided into two groups, bucket-based algorithms and inverse rules-based algorithms. The algorithms based on inverse rules have considered the problem of query rewriting in the presence of functional dependencies and inclusion dependencies in a mediated schema. However, there has been no bucket-based algorithm for this problem. We consider query rewriting in the presence of functional dependencies in a mediated schema. We give three conditions to check whether a view which does not contain any subgoal of a query is needed for answering the query when there are a set of functional dependencies in a mediated schema. We divide all the MCDs in the MiniCon algorithm into two groups. The MCDs in one of groups can not answer the query alone. We combine these MCDs with the buckets created for views identified by the above three conditions, and then generate the query rewritings which are missed by the previous bucket-based algorithms. The problem of missing query rewritings in the presence of functional dependencies that occurs in the previous bucket-based algorithms is avoided. Our algorithm is scalable compared to the inverse rule-based algorithms for query rewriting in the presence of functional dependencies.

In the future, we will study the problem of query rewriting in the presence of functional dependencies in the case of join lossless decomposition. Inclusion dependencies are another important type of integrity constraints in databases. There has been no bucket-based algorithm for query rewriting in the presence of inclusion dependencies. This would be another issue to be addressed in the future.

References

1. S.Abiteboul, R.Hull, and V.Vianu. Foundations of Databases, Addison-Wesley Publishing Company, 1995.
2. A.V.Aho, Y.Sagiv, and J.D.Ullman. Equivalences among relational expressions, SIAM Journal on Computing, 8(3): 218-246, May 1979.

3. O.M.Duschka and M.R.Genesereth. Answering Recursive Queries Using Views, Proc. of 16th ACM Conference on Principles of Database Systems, PODS, Tucson, AZ, May 1997.
4. O.M.Duschka, M.R.Genesereth, and A.Y.Levy. Recursive Query Plans for Data Integration. Journal of Logic Programming, special issue on Logic Based Heterogeneous Information Systems,43(1),49-73, 2000.
5. P.Godfrey, J.Grant, J.Gryz, and J.Minker. Integrity Constraints: Semantics and Applications. Logics for Databases and Information Systems, J.Chomicki and G.Saake, Kluwer, 1998.
6. J. Grant and J.Minker. A logic-based approach to data integration, TLP 2(3):323-368, 2002.
7. Jarek Gryz. An Algorithm for Query Folding with Functional Dependencies. Intelligent Information Systems VII Proceedings of the Workshop, Malbork, Poland, June, 1998.
8. Jarek Gryz. Query Folding with Inclusion Dependencies. Proceedings of ICDE'98, Orlando, Florida, USA. IEEE Computer Society 1998, ISBN 0-8186-8289-2.
9. Jarek Gryz. Query rewriting using views in the presence of functional and inclusion dependencies. Information System, 1999, 24(7):597-612.
10. D.S. Johnson and A. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. Journal of Computer and system Sciences 28, 167-189, 1984.
11. A.Y.Levy. Answering Queries Using Views: A Survey. VLDB Journal,10(4),270-294,2001.
12. A.Y.Levy, A.Rajaraman, and J.J.Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. Proceedings of the 22nd VLDB Conference,1996.
13. A.Y.Levy, A.Rajaraman, and J.J.Ordille. Query-Answering Algorithms for Information Agents. Proceedings of the AAAI 1996 .
14. Prasenjit Mitra. An Algorithm for Answering Queries Efficiently Using Views. In Proceedings of the 12th Australasian Database Conference, 2001.
15. Rachel Pottinger and Alon Y. Levy. A Scalable Algorithm for Answering Queries Using Views. Proc. of the 26th International Conference on Very Large Data Bases(VLDB), Cairo, Egypt,2000.
16. X. Qian. Query folding. In Proceedings of the 12th IEEE International Conference on Data Engineering(ICDE'96),48-55,1999.
17. J.Wang, M.Maher, and R.Topor. Rewriting General Conjunctive Queries Using Views. Proceedings of 13th Australasian Database Conference (ADC2002),2002.