Finding Equivalent Rewritings in the Presence of Arithmetic Comparisons

Foto Afrati[1,⋆], Rada Chirkova[2,⋆⋆], Manolis Gergatsoulis[3], and Vassia Pavlaki[1,⋆]

[1] Department of Electrical and Computing Engineering,
National Technical University of Athens (NTUA), 15773 Athens, Greece
{afrati, vpavlaki}@softlab.ntua.gr
[2] Computer Science Department, North Carolina State University,
Campus Box 7535, Raleigh, NC 27695-7535
chirkova@csc.ncsu.edu
[3] Department of Archive and Library Sciences, Ionian University,
Palea Anaktora, Plateia Eleftherias, 49100 Corfu, Greece
manolis@ionio.gr

Abstract. The problem of rewriting queries using views has received significant attention because of its applications in a wide variety of data-management problems. For select-project-join SQL (a.k.a. conjunctive) queries and views, there are efficient algorithms in the literature, which find equivalent and maximally contained rewritings. In the presence of arithmetic comparisons (ACs) the problem becomes more complex. We do not know how to find maximally contained rewritings in the general case. There are algorithms which find maximally contained rewritings only for special cases such as when ACs are restricted to be semi-interval. However, we know that the problem of finding an equivalent rewriting (if there exists one) in the presence of ACs is decidable, yet still doubly exponential. This complexity calls for an efficient algorithm which will perform better on average than the complete enumeration algorithm. In this work we present such an algorithm which is sound and complete. Its efficiency lies in that it considers fewer candidate rewritings because it includes a preliminary test to decide for each view whether it is potentially useful in some rewriting.

1 Introduction

The problem of answering queries using views (i.e. rewriting queries using views) is as follows. Suppose we are given a query Q over a database schema, and a set of view definitions $V_1, V_2, \ldots, V_k$ over the same schema. We want to know whether and how we can answer the query Q using only the answers to the views

⋆ The project is co-funded by the European Social Fund (75%) and National Resources (25%)- Operational Program for Educational and Vocational Training II (EPEAEK II) and particularly the program PYTHAGORAS.

⋆⋆ This author's work on this material has been supported by the National Science Foundation under Grant No. 0307072.

Y. Ioannidis et al. (Eds.): EDBT 2006, LNCS 3896, pp. 942–960, 2006.

$V_1, V_2, \ldots, V_k$. The problem has recently received significant attention because of its applications in a wide variety of data management problems, query optimization, maintenance of physical data independence, data integration, data warehousing, global information systems and mobile computing.

When answering queries using views we often need either find *equivalent rewritings* for a query or *maximally contained rewriting* (MCR). In data integration, where views describe a set of autonomous heterogenous data sources, we search for a maximally-contained rewriting, which provides the best answer, given the available sources. In query optimization or maintenance of physical data independence we search for a solution that uses the views and is equivalent (instead of contained) to the original query. When the query and views are conjunctive (i.e., select-project-join) without comparison predicates, the maximally-contained rewriting is a union of conjunctive queries over the views [2].

The original definition of conjunctive queries does not allow for comparisons between data values. However, in practice users often ask select-project-join queries that do involve comparisons in the selection condition (e.g. $price \leq 100$). For this reason, we extend the class of conjunctive queries by allowing built-in predicates which are arithmetic comparisons (ACs). So the problem of answering queries using views in the presence of arithmetic comparisons becomes more important, yet more complex. The following example illustrates this complexity.

Example 1. Consider the following query Q and set of views V_1, V_2:

$Q: \ q(X, X) \text{ :- } a(X, X), b(X), X < 7$
$V_1: \ v_1(T, U) \text{ :- } a(S, T), b(U), T \leq S, S \leq U$
$V_2: \ v_2(T, U) \text{ :- } a(S, T), b(U), T \leq S, S < U$

The query $Q': \ q(A, A) \text{ :- } v_1(A, A), A < 7$ is an equivalent rewriting of Q using V_1. To see why, suppose we expand Q' by replacing the view subgoal $v_1(A, A)$ by its definition. We get the expansion $Q'^{exp}: \ q(A, A) \text{ :- } a(S, A), b(A), A \leq S, S \leq A, A < 7$. By equating S and A we see that the expansion is equivalent to Q. Notice that the definitions of the views V_1, V_2 differ only on their second inequalities. However V_2 can not be used to answer Q. Thus, it is the comparison predicate that affects the existence of the rewriting.

Equivalent and contained rewritings use the containment test. Several algorithms have been proposed for testing containment in the presence of arithmetic comparisons [12, 10, 25, 4]. Some of these algorithms [10, 25] first normalize the queries by replacing constants and shared variables, each with new unique variables, and add arithmetic comparisons to equate those new variables to the original constants or shared variables. The containment is tested by checking a logical implication using multiple containment mappings. Another containment test existing in the literature is based on canonical databases [17, 12].

The problem of finding an equivalent rewriting (if there exists one) in the general case of ACs is decidable, yet still doubly exponential [3]. This complexity calls for an efficient algorithm which will perform better on average than the complete enumeration algorithm.

In this work we present an algorithm that, given a query and a set of views which are conjunctive queries with arithmetic comparisons, finds an equivalent rewriting if there exists one. The algorithm is sound and complete. Its efficiency lies in that it considers fewer candidate rewritings because it includes a preliminary test to decide for each view whether it is potentially useful in some rewriting. One of the challenges of our work consists in finding the relationship between the two problems; a) finding equivalent rewritings in the case of conjunctive queries with arithmetic comparisons and b) finding equivalent rewritings in the case of simple conjunctive queries. Such relation would allow us to leverage on existing algorithms for the latter problem. However this is not easy as we explain in detail in Subsection 3.1.

Another challenge comes from the following observation. In the case of conjunctive queries, if an equivalent rewriting exists in the language of union of conjunctive queries, then there exists one which is a single conjunctive query. However, in the case of conjunctive queries with arithmetic comparisons this property does not hold. Indeed even for very simple cases of conjunctive queries and views with arithmetic comparisons, it is often not possible to find equivalent rewritings in the form of a single conjunctive query with arithmetic comparisons. Instead, it is possible to find equivalent rewritings in the form of unions of conjunctive queries with arithmetic comparisons, as the following example illustrates.

Example 2. Consider the following query Q and set of views V_1, V_2:

$Q: \ q() \text{ :- } p(X), X \geq 0$
$V_1: \ v_1() \text{ :- } p(X), X = 0$
$V_2: \ v_2() \text{ :- } p(X), X > 0$

It is easy to see that there is no conjunctive query which is an equivalent rewriting of Q using V_1, V_2. Instead, the following union of conjunctive queries is an equivalent rewriting:

$r_0() \text{ :- } v_1()$
$r_0() \text{ :- } v_2()$

1.1 Related Work

The problem of answering queries using views is closely related to the problem of testing for query containment. Chandra and Merlin [6] have shown that the problems of containment, minimization, and equivalence of conjunctive queries are NP-complete. Klug in [12] showed that the containment problem for the class of conjunctive queries with arithmetic comparisons is in Π_2^P which is the second level of the polynomial hierarchy introduced by Stockmeyer [23]. In the same work was also proved that when only left (or right) semi-interval comparisons are used, the containment problem is shown to be in NP. In a more recent work Afrati et al. [4] showed more classes of conjunctive queries with arithmetic comparisons for which the problem of query containment is in NP. Van der Meyden in [24] proved Klug's conjecture that containment for conjunctive queries with inequality arithmetic comparisons is Π_2^P-complete. He also pointed out

that the containment problem for conjunctive queries with inequalities ($\neq$) is also Π_2^P-complete. The work in [13] studies the computational complexity of the query containment problem of queries with inequality ($\neq$). In fact, Kolaitis et al. proved that the complexity for the containment problem for safe conjunctive queries with inequalities ranges between coNP and Π_2^P-completeness depending on how many times each database predicate occurs in the body of the contained query. They also showed that when one of the two queries is fixed the problem can be DB-complete, where DB is the class of all decision problems that are the conjunction of a problem in NP and a problem in coNP.

The problem of finding whether there exists an equivalent rewriting for a query using views was studied in [14]. An efficient algorithm for finding equivalent rewritings with the smallest number of subgoals is given in [5]. The work in [16] considers the problem of answering conjunctive queries using infinite sets of views and they extend their results to cases when the query and the views use the built-in predicates $<, \leq, =$ and $\neq$.

The work in [1] shows how to find a Datalog maximally-contained rewriting (MCR) for a special case of Datalog queries and views that are unions of conjunctive queries. Several algorithms have been developed for finding rewritings of queries using views. The bucket algorithm [9, 15], the inverse-rule algorithm [8, 21, 1], the MiniCon algorithm [20], and the Shared-Variable-Bucket algorithm [18] are some of them (see [11] for a survey.) These algorithms aim at generating contained rewritings for a query that compute a subset of the answer to the query, and take the open-world assumption.

Afrati et al. in [2, 3] study the problem of query rewriting in the presence of arithmetic comparisons. They show that it is decidable to tell whether there exists an equivalent rewriting which is the union of conjunctive queries with arithmetic comparisons. They also investigate the existence of maximally contained rewritings in the presence of arithmetic comparisons and prove that for a special case of semi-interval comparisons there is a maximally contained rewriting.

2 Preliminaries

In this section we review the problem of query rewriting using views and summarize results in the literature on conjunctive queries with arithmetic comparisons. In the remainder of the paper we shall use names beginning with lower-case letters for constants and relations, and names beginning with upper-case letters for variables. We use $V, V_1, \ldots, V_m$ to denote views that are defined by conjunctive queries on the base relations. Moreover, for the sake of simplicity, we use "CQ" to represent "conjunctive query", "AC" for "arithmetic comparison", and "CQAC" for "conjunctive query with arithmetic comparisons".

2.1 Answering Queries Using Views

We start by reviewing the problem of answering queries using views for conjunctive queries (i.e., select-project-join queries). A conjunctive query CQ is a query of the form: $h(\overline{X})$:- $e_1(\overline{X}_1), \ldots, e_k(\overline{X}_k)$, where the head $h(\overline{X})$ represents the

results of the query, and $e_1 \ldots e_k$ are database relations. Each atom in the body of a conjunctive query is said to be a *subgoal*. Every argument in the subgoal is either a variable or a constant. The variables in $\overline{X}$ are called *head* or *distinguished* variables, while the variables in $\overline{X}_i$ are called *body* variables of the query. A conjunctive query is said to be *safe* if all its distinguished variables also occur in its body. A query Q_1 *is contained* in a query Q_2, denoted $Q_1 \sqsubseteq Q_2$, if for any database D of the base relations, the answer computed by Q_1 is a subset of the answer computed by Q_2, i.e., $Q_1(D) \subseteq Q_2(D)$. The two queries are *equivalent*, denoted $Q_1 \equiv Q_2$, if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$.

Chandra and Merlin [6] show that a conjunctive query Q_1 is contained in another conjunctive query Q_2 if and only if there is a containment mapping from Q_2 to Q_1. The containment mapping maps the head and all the subgoals in Q_2 to Q_1. It maps each variable to either a variable or a constant, and maps each constant to the same constant. Concerning unions of CQs, the following containment test is from [22]; a union of CQs $P_1 \cup \ldots \cup P_k$, is contained in a union of CQs $Q_1 \cup \ldots \cup Q_n$, denoted $P_1 \cup \ldots \cup P_k \sqsubseteq Q_1 \cup \ldots \cup Q_n$, iff for all P_i there exists some Q_j such that $P_i \sqsubseteq Q_j$.

Let Q be a query defined on a database schema S, V be a set of views defined on S, and D be a database with the schema S. A query R is a rewriting of the query Q using the views in V if the subgoals of R are only view predicates defined in V or interpreted predicates. The *expansion* of a query P on a set of views V, denoted by P^{exp}, is obtained from P by replacing all the views in P with their corresponding base relations. Note that in the case of union of CQs the following holds: if $R = \cup R_i$, then $R^{exp} \equiv \cup(R_i^{exp})$.

Given a query Q and a view set V, a query P is a *contained rewriting* of query Q using V if P uses only the views in V, and $P^{exp} \sqsubseteq Q$. That is, P computes a partial answer to the query. Given a rewriting language L (e.g., unions of conjunctive queries), we call P an *equivalent rewriting* of Q using V w.r.t. L if P is in L, and $P^{exp} \equiv Q$. We call P a *maximally-contained rewriting (MCR)* of Q w.r.t. L if (1) P is a contained rewriting (in L) of Q, and (2) there is no contained rewriting P_1 (in L) of Q such that P_1 properly contains P.

2.2 Conjunctive Queries with Arithmetic Comparisons

In this work we study the problem of rewriting a query using views when both the query and the views are of the following form:

$$h(\overline{X}) \text{ :- } e_1(\overline{X}_1), \ldots, e_k(\overline{X}_k), C_1, \ldots, C_m$$

where each C_i is an arithmetic comparison in the form $A_1 \theta A_2$, where A_1 and A_2 are variables or constants. The operator θ is one of the following: $<, \leq, =, >,$ or $\geq$. We call an arithmetic comparison *open* if its operator is $<$ or $>$ and *closed* if its operator is $\leq$ or $\geq$. We call the e_i's *ordinary subgoals*, and the C_i's *arithmetic comparison subgoals* (AC subgoals). In addition, the following assumptions must hold:

1) Values for the arguments in the arithmetic comparisons are chosen from an infinite, totally densely ordered set, such as the rationals or reals.

2) The arithmetic comparisons are not contradictory; that is, there exists an instantiation of the variables such that all the arithmetic comparisons are true.

3) All the comparisons are safe, i.e., each variable in the comparisons also appears in some ordinary subgoal.

2.3 Testing Containment of CQACs

When the queries and views are expressed as conjunctive queries (without arithmetic comparisons), we know how to find equivalent rewritings (if they exist) and maximally-contained rewritings (MCRs) that are unions of conjunctive queries (see [11] for a survey). However, arithmetic comparisons introduce many complications to the problem. In particular, both the containment mapping theorem [6] and the theorem for unions of CQs [22] no longer hold.

Let Q_1 and Q_2 be two conjunctive queries with arithmetic comparisons (CQACs). To test whether $Q_2 \sqsubseteq Q_1$ there are two most popular methods: a) the test of canonical databases [17, 12] and b) the test of Gupta and Zhang-Ozsoyoglu [10, 25]. In the following paragraphs we shortly review the first test, which we use extensively throughout the paper. Due to space limit, we refer the reader to [4] for more details about the second test. Before presenting the test, we briefly explain how to obtain a canonical database D given a query Q: we turn each ordinary subgoal into a fact by replacing each variable in the body by a distinct constant, and treating the resulting subgoals as the only tuples in D.

We now describe the test of canonical databases [17, 12]. When dealing with CQACs we must consider the set of values in the database as belonging to a totally ordered set, e.g. the rationals or reals. This test produces an exponential number of canonical databases any one of which could be a counterexample to the containment. Suppose we want to test $Q_1 \sqsubseteq Q_2$. We do the following:

1) Consider all partitions of the variables of Q_1. For each partition P consider all possible total orders of the members of the partition and assign to each member b_i of P a unique positive integer n_i such that if b_k, $b_l \in P$ and $b_k < b_l$, then $n_k < n_l$. Then, substitute (freeze) every variable in each member b_i of P by the corresponding constant n_i. Thus we obtain a number of canonical databases $D_1, D_2, \ldots, D_n$, one database for each different order in each partition. Each D_i consists of the frozen subgoals of Q_1 excluding the subgoals having comparison predicates.

2) Test whether for all D_i that make the body of Q_1 true, $Q_2(D_i)$ includes the frozen head of Q_1. The frozen head of Q_1 is obtained by making the same substitution of constants for variables that yielded D_i.

3) $Q_1 \sqsubseteq Q_2$ if and only if (2) holds.

2.4 Known Decidability Results

The following two theorems from [2] prove the decidability of the problem we study in this work.

Theorem 1. *(CQAC equivalent rewritings) For a query and views that are conjunctive queries with arithmetic comparisons, it is decidable whether there is an*

equivalent rewriting for the query using the views, where the rewriting is also a conjunctive query with arithmetic comparisons. If such an equivalent rewriting exists, there is an algorithm to find it.

Theorem 2. *(Union of CQAC equivalent rewritings) For a query and views that are conjunctive queries with arithmetic comparisons, it is decidable whether there is an equivalent rewriting for the query using the views, where the rewriting is a finite union of conjunctive queries with comparisons. If such an equivalent rewriting exists, there is an algorithm to find it.*

2.5 Technical Details

This subsection contains some technical points that are needed to understand the details of our algorithm. Let D be the canonical database of the query Q when ignoring the ACs and let $V(D)$ be the result of applying the view definitions V on database D. For each tuple in $V(D)$, we "unfreeze" each introduced constant back to the original variable of Q, and obtain a set of view tuples $T(V)$.

A head homomorphism [20] of the head variables in a view is a partitioning of these variables, such that all the variables in each member of the partition are equated to a single variable. For a specific view, different head homomorphisms result in different view tuples.

Now we consider containment mappings from the ordinary subgoals of the query to the ordinary subgoals of the view. Let μ be one such mapping from some query subgoals to view subgoals. The definition of the *shared variable property* for μ is the following: whenever a query variable X is mapped on a nondistinguished view variable, then all query subgoals that contain X are in the domain of the mapping.

Definition 1. *We assume that the sets of variables in the query and the view definitions are disjoint. An MCD mapping (MiniCon Description [20]) μ is an one-to-one*[1] *containment mapping from the ordinary subgoals of the query to the ordinary subgoals of view V which satisfies the shared variable property. Let S be the set of query variables that are mapped to head variables of view V under μ. We rename each variable X in $\mu(S)$ to $\mu^{-1}(X)$. Let v be the head of view V after this renaming. Then, we say that μ is an* MCD mapping for view tuple v.

Intuitively, an MCD mapping represents a fragment of a containment mapping from the query to the expansion of the rewriting. The way in which MCDs are constructed guarantees that these fragments can be combined seamlessly.

Definition 2. *Let v_i and v_j be view tuples of V such that there is a containment mapping from v_i to v_j. We say that v_i is a* more relaxed *form of v_j.*

[1] This is the only difference with the algorithm in [20]. Here we consider one-to-one mappings because we are searching for equivalent rewritings whereas in [20] they are searching for MCR's.

Definition 3. *We call a nondistinguished variable X in a view V* exportable *if there is a head homomorphism h for V, such that the inequalities in $h(V)$ imply that X is equal to a distinguished variable in V.*

To find exportable nondistinguished variables in a view V, we use the ACs in V to construct its inequality graph [12], denoted by $G(V)$. That is, for each variable or constant A in ACs we create a node in the graph labelled with A. Then, for every comparison predicate $A\theta B$ where θ is $<$ or $\leq$, we introduce an edge labeled θ from A to B. If there is a path from node A to C, we have $A \leq C$. If there is a $<$-labeled edge on any path between A and C, then $A < C$. We need the following concepts to show how to export a nondistinguished view variable.

Definition 4. *Let X be a nondistinguished variable in a view V. The leq-set (less-than-or-equal-to set) of X, denoted by $S_{\leq}(V, X)$, includes all distinguished variables Y of V that satisfy the following conditions. There exists a path from Y to X in the inequality graph $G(V)$, and all edges on all paths from Y to X are labeled $\leq$. In addition, in all paths from Y to X, there is no other distinguished variable except Y.*

Correspondingly, we define the geq-set (greater-than-or-equal-to set) of a variable X, denoted by $S_{\geq}(V, X)$. The following lemma from [2] can help us decide if a variable in a view V is exportable.

Lemma 1. *A nondistinguished variable X in view V is exportable if and only if both $S_{\leq}(V, X)$ and $S_{\geq}(V, X)$ are nonempty.*

3 Finding Equivalent Rewritings of CQAC Queries Using CQAC Views

In the following paragraphs we present an algorithm that finds an equivalent rewriting (if there exists one) for queries that are CQAC using views that are also CQAC. Our algorithm consists of two phases. In the first phase we find all candidate rewritings that contain the query, while in the second phase we add constraints to the rewritings (obtained in the first phase) and we check whether these rewritings are contained in the query.

The efficiency of our algorithm is mainly based on the observations that if there exists an equivalent rewriting then there exists one which uses view subgoals out of a restricted search space of potentially useful view subgoals. These useful view subgoals are found by using techniques for finding rewritings of queries and views without arithmetic comparisons. In more detail, we use chase-like techniques [7, 19, 5] to find candidate useful subgoals and then we prune the space even further by using techniques used in finding maximally contained rewritings [20].

The main challenge of our algorithm however comes from the presence of arithmetic comparisons and the complications in testing query containment in

this case. Due to these complications, existing algorithms cannot be used without modification as the discussion in the next subsection shows.

3.1 Technical Challenges

In the first phase of our algorithm we find rewritings using the views V that contain the query Q. We begin by considering query Q' and view V' which result from Q and V after dropping the ACs. Then, we find maximally contained rewritings of Q' using V' and we ensure that these are also equivalent rewritings of Q' using V' by deleting the view tuples that are not more relaxed. In particular, we use the algorithm proposed in [20] adjusted to our setting as described in Subsection 3.2. Other known algorithms which compute either equivalent rewritings or maximally contained rewritings might also be used. In any case it is not straightforward how they can be useful. The reason is that these algorithms focus on rewritings which do not use redundant view subgoals or that are containment minimal [5]. The following two examples illustrate this point.

Example 3. Consider query Q and set of views $V = \{V_1, V_2, V_3\}$:

$$\begin{aligned}
Q:\ & q() \text{ :- } a(X_1,X_2), a(X_2,X_3), a(X_3,X_4), a(X_4,X_5), a(X_5,X_6), a(X_6,X_7),\\
& \quad a(X_7,X_1),\ X_2 > 5,\ X_7 < 8\\
V_1:\ & v_1(X_1,X_4) \text{ :- } a(X_1,X_2), a(X_2,X_3), a(X_3,X_4), a(X_4,X_5), a(X_5,X_6),\\
& \quad a(X_6,X_7), a(X_7,X_1), X_3 > 5\\
V_2:\ & v_2(X_3,X_5) \text{ :- } a(X_1,X_2), a(X_2,X_3), a(X_3,X_4), a(X_4,X_5), a(X_5,X_6),\\
& \quad a(X_6,X_7), a(X_7,X_1), X_4 < 8\\
V_3:\ & v_3(X,Y) \text{ :- } a(X,X_2), a(X_2,Y)
\end{aligned}$$

Note that Q evaluates to *true* whenever there exists a closed path of length 7 in the database D such that the conditions shown in Figure 1(a) hold for that path. We consider also the query Q' which is defined as Q with the ACs dropped and the views V_1', V_2', and V_3' (with predicates v_1', v_2' and v_3' respectively) which are the views V_i without the ACs in their definition. For this last query Q' the CoreCover algorithm [5] will find an equivalent rewriting R' where:

$$R':\ r() \text{ :- } v_1'(X,Y)$$

However, if we use this rewriting and simply add ACs, we will not find an equivalent rewriting of the original query Q using views V_i. Note that such an equivalent rewriting R does exist and is the following:

$$R:\ r() \text{ :- } v_1(X,Y), v_2(Z,X), v_3(Y,Z)$$

This comes easily from Figure 1(b) which shows the two heptagons corresponding to the (expansions of the) atoms $v_1(X,Y)$ and $v_2(Z,X)$ with a common vertex labelled X. Notice also the path formed by the arcs $Y \rightarrow X_2''$ and $X_2'' \rightarrow Z$ corresponding to the (expansion of the) atom $v_3(Y,Z)$. Thus the Figure 1(b) represents the expansion of R. It is easy to see that $Q \sqsubseteq R$ since whenever Q evaluates to true then so does R (we can check it by considering twice the heptagon corresponding to instance of the body of Q). To check

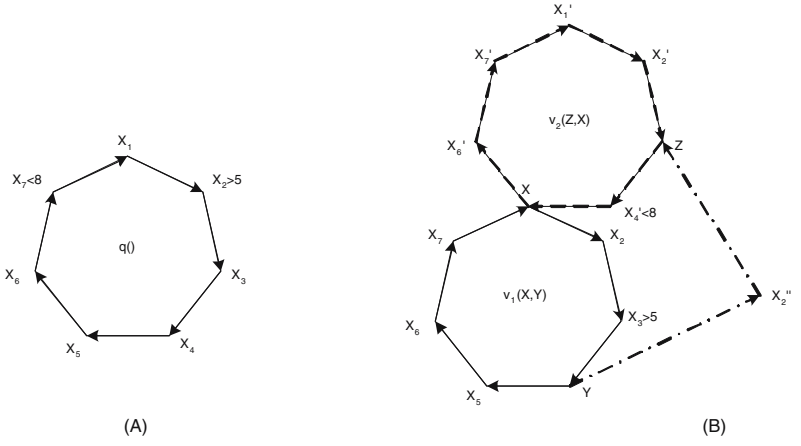

Fig. 1. Example 3

that $R \sqsubseteq Q$ notice that the heptagon with vertices $Z, X_4', X, X_2, X_3, Y, X_2''$ which is formed by the expansion of R satisfies the properties required by the query Q. It is not straightforward that the heptagon fulfills the conditions of Q.

The rewriting:$R'' :\ r() \text{ :- } v_1'(X,Y), v_2'(Z,X), v_3'(Y,Z)$ is an equivalent rewriting of Q' using V' and in fact it is the rewriting that our algorithm needs to use in order to find an equivalent rewriting of the given CQAC Q using the views V. However, this rewriting would not have been computed by the existing algorithms since it contains the redundant subgoals $v_2'(Z,X)$ and $v_3'(Y,Z)$.

Example 4. Suppose we are given the following query and set of views:

$Q :\ q(X,Y) :- a(X,Z_1), a(Z_1,2), b(2,Z_2), b(Z_2,Y), Z_1 < 5, Z_2 > 8$
$V_1 :\ v_1(X,Y) :- a(X,Z_1), a(Z_1,2), b(2,Z_2), b(Z_2,Y), Z_1 < 5$
$V_2 :\ v_2(X,Y) :- a(X,Z_1), a(Z_1,2), b(2,Z_2), b(Z_2,Y), Z_2 > 8$

Note that an equivalent rewriting is

$R :\ r(X,Y) \text{ :- } v_1(X,Y'), v_2(X',Y)$

We consider the query Q' which is defined as query Q with the ACs dropped and the views V_1' and V_2' which are the two views again without the ACs in their definition. In this case the rewriting of Q' using V' does not contain redundant subgoals. Still, it is not a containment minimal rewriting [5], i.e. there is another equivalent rewriting of Q' using V' which is the following:

$R' :\ r(X,Y) \text{ :- } v_1(X,Y).$

However we cannot obtain from R' an equivalent rewriting of Q using V. Therefore we cannot use the algorithm in [5].

3.2 Phase 1: Construct Rewritings that Contain the Query

In Phase 1 we begin by creating all canonical databases of the query. For this we consider all total orders of the variables of the query and the constants of both the query and the views. Thus we obtain a number of canonical databases. Notice that the number of canonical databases is exponential in the number of variables. From these canonical databases we keep only those that compute the head of the query, or if the query is boolean, that make the body of the query true. Suppose $D_1, D_2, \ldots, D_k$ are these canonical databases. For every $D_i, i = 1, \ldots, k$ we compute the view tuples $T_i(V)$ by applying the view definitions V on D_i and restoring back the variables in the tuples. Note that the total order of each canonical database must satisfy the ACs of views; otherwise we omit the view tuples corresponding to the specific canonical database and view definition.

Example 5. Suppose we are given the following query Q and the view V:

$Q:\ q(A) \text{ :- } r(A), s(A, A), A \leq 8$
$V:\ v(Y, Z) \text{ :- } r(X), s(Y, Z), Y \leq X, X \leq Z$

(Note that $P:\ p(A) \text{ :- } v(A, A), A \leq 8$ is an equivalent rewriting of Q).
To compute the sets of view tuples we first construct the canonical databases of Q by considering all variables of Q and all constants of both query and views:

$D_1 = \{r(a), s(a, a)\} : a < 8$
$D_2 = \{r(a), s(a, a)\} : a = 8$
$D_3 = \{r(a), s(a, a)\} : a > 8$

From these canonical databases we keep only D_1, D_2 as they compute (taking also into account the comparison predicates) the head of the query. To compute the view tuples corresponding to D_1 we apply the view definitions to D_1. We get $V(D_1) = \{v(a, a)\}$. Then, by restoring the constant a back to the variable A we get the set of view tuples $T_1(V) = \{v(A, A)\}$. Similarly, for the canonical database D_2, we get $T_2(V) = \{v(A, A)\}$.

Having computed $T_i(V)$ we proceed as follows. Let Q_0 be the query obtained by deleting the comparisons from Q, and let V_0 be the view obtained by deleting the comparisons from V and exporting in the head of the view definition the *exportable variables* (Subsection 2.5, or see [4] for more details). Due to the different ways of exporting variables, it is possible that to one view in V may correspond more than one view in V_0. The following example illustrates this point.

Example 6. Suppose we are given the following view definition:

$V:\ v(X, Y, W) \text{ :- } a(X, Z_1), a(Z_1, Z_2), b(Z_2, Y, W), X \leq Z_1, W \leq Z_1, Z_1 \leq Y.$

By equating variable X to variable Y we obtain the view tuple v_1 and by equating variable Y to variable W we obtain the view tuple v_2. In both cases we export variable Z_1, in v_1 by equating Z_1 to X and in v_2 by equating Z_1 to Y. That is:

$V_1:\ v_1(X, X, W) \text{ :- } a(X, X), a(X, Z_2), b(Z_2, X, W)$
$V_2:\ v_2(X, Y, Y) \text{ :- } a(X, Y), a(Y, Z_2), b(Z_2, Y, Y)$

We continue with an overview of the algorithm presented by Pottinger and Levy in [20] in order to make clear the contribution of our work. The algorithm in [20] consists of two phases. The first phase computes MCDs and populates the buckets accordingly. In the second phase the algorithm combines the content of the buckets to create MCRs. Our algorithm starts as the first phase of [20] but after this we do not proceed directly to the second phase. First, we delete those view tuples in the buckets that are not more relaxed forms of view tuples in $T_i(V)$. Then, we proceed to the second phase of [20] but only to get an answer to whether there exists an MCR. If it does not exist, our algorithm stops. If it does exist, then we output a rewriting PR_i consisting of a conjunctive query with subgoals the content of all buckets. So to every canonical database corresponds only one rewriting.

The above procedure is repeated for every canonical database. If there exists a canonical database D_i for which there is no maximally contained rewriting, then the algorithm stops and there is no equivalent rewriting of the query. If there is at least one maximally contained rewriting, then the output of the first phase of our algorithm is a set of *Pre-Rewritings* (denoted $PR_1, PR_2, \ldots, PR_k$), one for each canonical database. Figure 2 summarizes the steps of the first phase of our algorithm.

Example 7. (Continued from Example 5) There are two Pre-Rewritings PR_1, PR_2 corresponding to the two canonical databases D_1, D_2:

$PR_1(A) :- v(A, A)$
$PR_2(A) :- v(A, A)$

Procedure Pre-Rewritings:
Input: A CQAC Q and a set V of CQAC views.
Output: A set of Pre-Rewritings $PR_1, PR_2, ..., PR_k$ together with the corresponding canonical databases $D_1, D_2, ..., D_k$.
Method:
(1) Construct all canonical databases for Q by taking into account the variables of Q and all constants of the query and views. Construct also query Q_0 which is Q with the ACs dropped, and a set V_0 of CQ views which is V with the ACs dropped.
(2) Keep only those canonical databases which compute the head of Q.
(3) For every canonical database D_i do:

1. Compute the view tuples $T_i(V)$ by applying the view definitions V on D_i.
2. If for a canonical database D_i it holds $T_k(D_k) = \emptyset$ then stop (as there is no rewriting).
3. Run the first phase of [20] with respect to Q_0 and V_0 which populates the buckets.
4. Delete from the buckets those tuples that are not more relaxed forms of view tuples in the $T_i(V)$.
5. Run the second phase of [20]. If it produces an MCR continue, otherwise stop.
6. Produce a Pre-Rewriting whose subgoals are all view tuples contained in the buckets.
7. Output the Pre-Rewriting together with the corresponding canonical database.

Fig. 2. Phase 1 of our algorithm

In Proposition 1 we prove that each canonical database D_i of the query must correspond to one CQAC P_j^{exp} of P which computes the query head on this canonical database.

Proposition 1. *Let Q be a CQAC query. If there exists a union of CQAC $P = \cup P_i$ which is an equivalent rewriting of Q, then for every canonical database D_i of Q, there exists a P_j such that P_j^{exp} computes the head of Q in D_i.*

Proof. (sketch) The reason is that for every canonical database D_i of the query, P^{exp} must compute the head of the query on this canonical database. Therefore, there must exist a P_j such that P_j^{exp} computes the head of the query.

The view subgoals in the body of P_j (the corresponding CQAC of canonical database D_i) as a consequence of Proposition 1 are necessarily more relaxed forms of view tuples in $T_i(V)$. Therefore, it suffices to restrict our search to view tuples in more relaxed forms than tuples in $T_i(V)$. Proposition 2 shows that by restricting ourselves to view tuples, that we compute in Phase 1, we do not lose solutions.

Proposition 2. *Let Q be a CQAC query. Suppose there is an equivalent rewriting P of Q in the language of unions of CQACs using a set of CQAC views V. Then, there is a $P' = \cup P_i'$ which is an equivalent rewriting of Q using views V with the following property. There exists a canonical database D on which Q computes the head tuple such that any view (hence ordinary) subgoal of P_i' maps on a view tuple in D.*

Proposition 3 shows that by restricting ourselves to view tuples in their more relaxed form that are part of an MCR CQAC we do not lose solutions.

Proposition 3. *Let Q be a CQAC query. Suppose there is an equivalent rewriting P of Q in the language of union of CQACs using a set of CQAC views V. Then there is a P' which is an equivalent rewriting of Q using views V with the following property. Let P_i' be a CQAC in P'. Let $P_i' = P_{i,0}' + \beta_i$. Then $P_{i,0}'$ is a CQ in the MCR of Q_0 using V_0 possibly with redundant subgoals.*

Propositions 2 and 3 are partial results of the completeness of our algorithm and Lemma 2 is a partial result of soundness so far.

Lemma 2. *Let Q be a CQAC query and V a set of CQAC views. Let D_i, with $i = 1, \ldots, k$, be the canonical databases and PR_i, with $i = 1, \ldots, k$, the corresponding Pre-Rewritings obtained by procedure of Figure 2. Let $PR_i^{exp,V}$ be the expansion of PR_i wrt V. Then $Q \sqsubseteq \cup PR_i^{exp,V}$.*

Proof. The proof of the lemma follows from the containment test for CQACs.

3.3 Phase 2: Construct Rewritings that Are Contained in the Query

The second phase performs two tasks: a) it constructs the candidate rewritings by adding constraints to the Pre-Rewritings PR_i obtained in Phase 1, still preserving that the union of the new Pre-Rewritings still contains the query, b) it

checks that the candidate rewritings are also contained in the query. In task a) to every PR_i we add the constraints of the canonical database of Q to which this Pre-Rewriting corresponds. We call these new Pre-Rewritings PR'_i. Then, in task b) we check the containment in the query by considering the expansions of all PR'_is w.r.t. V and constructing the canonical databases of these expansions. We keep only those canonical databases that compute the head of the expansion (or if the expansion is boolean, that make the body true). Note that the expansion contains constraints coming from the bodies of the view definitions too. So fresh variables may also appear. However these variables are used only for checking the containment in the query.

Example 8. (Continued from Example 7). To those Pre-Rewritings obtained in Phase 1 we add the total order of the corresponding canonical database. So we have the following Pre-Rewritings:

$PR'_1(A)$:- $v(A, A), A < 8$
$PR'_2(A)$:- $v(A, A), A = 8$

We then consider the expansion of PR'_1, and PR'_2:

$$PR_1^{'exp}(A) \text{ :- } r(X), s(A, A), A < 8, A \leq X, X \leq A$$

which simplifies to

$$PR_1^{'exp}(A) \text{ :- } r(A), s(A, A), A < 8$$

and,

$$PR_2^{'exp}(A) \text{ :- } r(X), s(A, A), A = 8, A \leq X, X \leq A$$

which simplifies to

$$PR_2^{'exp}(A) \text{ :- } r(A), s(A, A), A = 8$$

We proceed to the construction of the canonical databases of every $PR_i^{'exp}$ by considering all variables and constants of the expansion. Here, both PR_i's have the same set of canonical databases.

$D_{1,1} = \{r(a), s(a, a)\} : a < 8$
$D_{1,2} = \{r(a), s(a, a)\} : a = 8$
$D_{1,3} = \{r(a), s(a, a)\} : a > 8$
$D_{2,1} = \{r(a), s(a, a)\} : a < 8$
$D_{2,2} = \{r(a), s(a, a)\} : a = 8$
$D_{2,3} = \{r(a), s(a, a)\} : a > 8$

We keep only the canonical databases that compute the head of the expansion of the rewriting. In this example we keep only the canonical databases $D_{1,1}$, $D_{2,2}$.

The last step of Phase 2 consists in checking the constraints for each PR'_i through a two-column tableau constructed as follows. Each row corresponds to a canonical database of the expansion of PR'_i. We apply the query Q on this canonical database and if the expansion head is computed, we place the

Procedure Equivalent_rewritings:
(1) Input: A set of Pre-Rewritings $PR_1, PR_2, ..., PR_k$ together with the corresponding canonical databases $D_1, D_2, ..., D_k$
(2) Output: An equivalent rewriting R
(3) Method:

1. For each PR_i do:
   (a) Construct PR_i' by adding the ACs of the canonical database D_i to which PR_i corresponds.
   (b) Consider the expansion $PR_i^{'exp}$ wrt V of PR_i' and all its canonical databases.
   (c) Make a two-column tableau as follows: in the left column place the total order of all canonical databases created from the $PR_i^{'exp}$ in which Q computes the head variable of $PR_i^{'exp}$. In the right column place the total order of the canonical databases created from the $PR_i^{'exp}$s in which Q does not compute the head variable of $PR_i^{'exp}$.
2. If a constraint appears on the right column of the tableau, then the algorithm fails (there is no rewriting). If not, then output $R = \cup PR_i'$.

Fig. 3. Phase 2 of our algorithm

constraint corresponding to the total order of the canonical database in the left column of the tableau. Otherwise, we place the constraint in the right column. In the end, if there is at least one constraint on the right column of the tableau there is no equivalent rewriting to the query. Otherwise, the equivalent rewriting of Q is the union of PR_i'. Figure 3 presents the steps of Phase 2.

Example 9. (Continued from Example 8). For every canonical database that we finally keep, we check the corresponding total order through the following tableau:

	Q satisfies db	Q does not satisfy db
$D_{1,1}$:	$a < 8$	
$D_{2,2}$:	$a = 8$	

Since no constraint appears on the right column of the tableau, then the equivalent rewriting R to the query Q consists of the union:

$r(A)$:- $v(A, A), A < 8$
$r(A)$:- $v(A, A), A = 8$

Example 10. This example illustrates the case when the algorithm detects that there is no equivalent rewriting and stops. Consider the query and view:

$Q : q(A)$:- $r(A), s(A, A), A \leq 8$
$V :\ v(Y, Z)$:- $r(X), s(Y, Z), Y \leq X, X < Z$

Phase 1: We construct the canonical databases of Q by considering all variables of Q and all constants of the query and views:

$D_1 = \{r(a), s(a, a)\} : a < 8$
$D_2 = \{r(a), s(a, a)\} : a = 8$
$D_3 = \{r(a), s(a, a)\} : a > 8$

We keep those canonical databases on which we compute the head of the query. That is, we keep only D_1, D_2. As $V(D_1) = V(D_2) = \emptyset$, the algorithm would stop in Phase 1, and the query has no equivalent rewriting.

3.4 Soundness and Completeness

To prove soundness and completeness of our algorithm we use Lemma 2 and Propositions 2 and 3 from Phase 1.

Proposition 4. *Let PR_i be the Pre-Rewriting computed in Phase 1 of the algorithm corresponding to the canonical database D_i of Q. Then, every $PR_i^{'exp}$ constructed in Phase 2 still computes the head of Q in D_i. Hence, $Q \sqsubseteq \cup PR_i^{'exp}$.*

Proof. (sketch) The PR_i's in Phase 2 are constructed from PR_is by adding the constraints implied by the total order of the corresponding canonical database D_i of Q. So the new constraints do not harm, and $\forall i$ $PR_i^{'exp}$ still computes the head of Q in D_i.

So far we have proved that our algorithm is complete i.e. if there are rewritings equivalent to Q with respect to the views in V, then our algorithm finds at least one. Lemma 3 proves that whenever our algorithm produces a rewriting then this rewriting is equivalent to the query.

Lemma 3. *Let Q be a CQAC query and V a set CQAC views. Let $PR = \cup PR_i$ be the set of Pre-Rewritings. When the algorithm does not fail then the output R of the algorithm in Figure 3 is an equivalent rewriting of Q using V.*

Theorem 3. *Given a query and views that are CQACs, our algorithm finds an equivalent rewriting (if there exists one) in the language of unions of CQACs.*

Proof. (sketch) Completeness: a consequence of Propositions 2, 3 and 4.
Soundness: a consequence of Lemma 3.

4 Experimental Results

In this section we present some of the experiments conducted to evaluate the efficiency of our algorithm. All the experiments were run on a machine with 3GHz Intel Pentium 4 processor with 512MB RAM and a 80GB hard disk, running the Windows XP operating system. Figure 4(a), (b) and (c) show that the runtime of the algorithm depends strongly on the number of distinct variables and constants in the CQAC queries and CQAC views rather than on the number of views.

Note that a completely naive full-enumeration algorithm would not have a chance because it would have to enumerate thousands of combinations of view tuples for a typical query. In simple words, we would not be able to draw the curves in the graphs as they would go nearly vertically.

In more detail, Figure 4(a) shows the dependence of the runtime on the number of views where the number of variables is kept constant (6 variables and

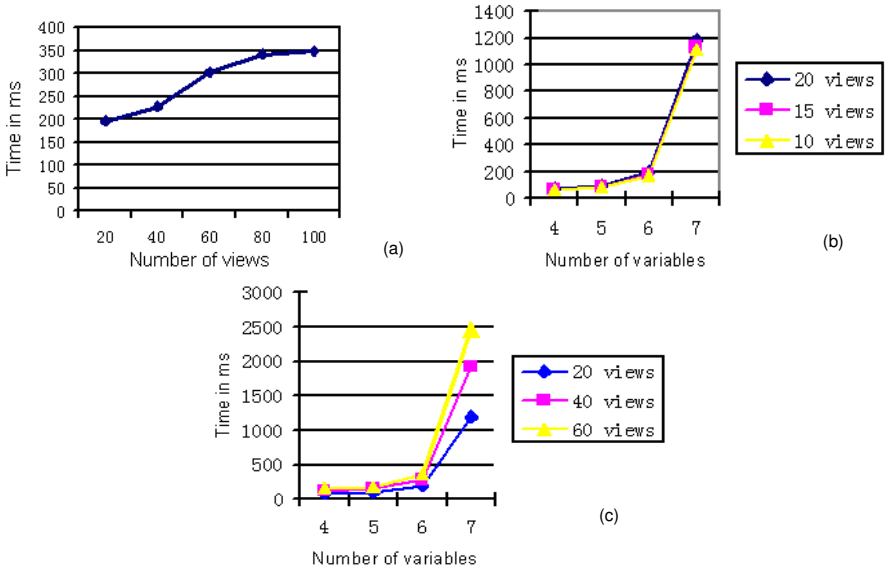

Fig. 4. Experimental results

constants). The graphs in Figures 4(b) and 4(c) present the dependence of our algorithm on both the number of the variables and the number of views. To be more precise, graph (b) gives the dependence for 10-20 views whereas graph (c) for 20-60 views.

5 Conclusions

The problem of rewriting queries using views in the presence of arithmetic comparisons is an important problem since users often need to pose queries containing inequalities. However the presence of arithmetic comparisons adds more complexities. The problem of finding an equivalent rewriting (if there exists one) in the presence of ACs is decidable. The doubly exponential complexity though calls for an efficient algorithm which will perform better on average than the complete enumeration algorithm.

In this work we present an algorithm which finds equivalent rewritings for conjunctive queries with arithmetic comparisons, and prove its correctness. Its efficiency lies in that it considers fewer candidate rewritings because it includes a preliminary test to decide for each view whether it is potentially useful in some rewriting. Experiments conducted to evaluate our algorithm proved its efficiency. In future work it would be interesting to investigate special cases in which our algorithm may have lower complexity, such as acyclic queries.

Acknowledgments. The authors would like to thank the students Manik Chandrachud and Dongfeng Chen who ran the experiments presented in this work.

References

1. F. Afrati, M. Gergatsoulis, and T. Kavalieros. Answering queries using materialized views with disjunctions. In *ICDT*, pages 435–452, 1999.
2. F. Afrati, C. Li, and P. Mitra. Answering queries using views with arithmetic comparisons. In *PODS*, pages 209–220, 2002.
3. F. Afrati, C. Li, and P. Mitra. Rewriting queries using views in the presence of arithmetic comparisons. Technical report, UC Irvine, 2002.
4. F. Afrati, C. Li, and P. Mitra. On containment of conjunctive queries with arithmetic comparisons. In *EDBT*, pages 459–476, 2004.
5. F. Afrati, C. Li, and J. D. Ullman. Generating efficient plans for queries using views. In *SIGMOD*, pages 319–330, 2001.
6. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.
7. A. Deutsch. *XML Query Reformulation over Mixed and Redundant Storage*. PhD thesis, University of Pennsylvania, 2002.
8. O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997.
9. G. Grahne and A. O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *ICDT*, pages 332–347, 1999.
10. A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint checking with partial information. In *PODS*, pages 45–55, 1994.
11. A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.
12. A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, 35(1):146–160, 1988.
13. P. G. Kolaitis, D. L. Martin, and M. N. Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *PODS*, pages 197–204, 1998.
14. A. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.
15. A. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, pages 251–262, 1996.
16. A. Levy, A. Rajaraman, and J. Ullman. Answering queries using limited external processors. In *PODS*, pages 227–234, 1996.
17. A. Levy and Y. Sagiv. Queries independent of updates. In *VLDB*, pages 171–181, 1993.
18. P. Mitra. An algorithm for answering queries efficiently using views. In *Proceedings of the Australasian Database Conference*, pages 99–106, 2001.
19. L. Popa. *Object/Relational Query Optimization with Chase and Backchase*. PhD thesis, University of Pennsylvania, 2000.
20. R. Pottinger and A. Halevy. A scalable algorithm for answering queries using views. *VLDB Journal*, 10(2-3):182–198, 2001.
21. X. Qian. Query folding. In *ICDE*, pages 48–55, 1996.

22. Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.
23. L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.
24. R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *PODS*, pages 331–345, 1992.
25. X. Zhang and M. Z. Ozsoyoglu. On efficient reasoning with implication constraints. In *DOOD*, pages 236–252, 1993.