# Accessing Data Using Views

Foto Afrati
Nat'l Technical University of Athens
Athens, Greece
afrati@ece.ntua.gr

Rada Chirkova
North Carolina State University
Raleigh, NC, U.S.A.
chirkova@csc.ncsu.edu

Prasenjit Mitra
Penn State University
University Park, PA, U.S.A.
pmitra@ist.psu.edu

## Abstract

A popular technique for answering queries using views is to first rewrite the queries using the views and then to evaluate the queries using the data in the views. Existing work has addressed the problem of finding equivalent or maximally-contained rewritings (MCR) of queries using views. At the same time, emerging applications require investigating the problem of rewriting queries using views in a more general setting, and pose new technical challenges. In this paper we consider the problem of finding whether there exist rewritings that (1) are contained in the query, (2) contain the query, or (3) overlap with the query. We investigate complexity issues and present efficient sound and complete algorithms that find minimally containing rewritings (MiCR) and check for the existence of contained rewritings, which is an easier problem than finding an MCR.

## 1 Introduction

In many applications, including data integration [23, 14, 20, 22, 26, 34], security, and web applications (e.g., databases accessible via the Internet [8]), users do not have direct access to the data, the available data are incomplete, or users do not have access rights to the entire database [32]. Consider, for instance, a bank application where customer access to the bank's database may be restricted to the customer's own accounts. Other examples include applications that use approximate query answering [2, 9, 11, 29] and security-aware or privacy-aware applications, see [10] for a survey.

In all the above applications, we can think of data sources as views on a global database schema, which is referenced in user queries. At the same time, specific settings and requirements vary depending on the scenario. For example, in data-integration applications we would like to use the sources (views) to obtain as much query-related information as possible, whereas in data security our objective is to hide sensitive information from certain classes of users. In the former case, our goal is to find all correct answers to the user query; in the latter case, we would like to ensure that the views do not convey any sensitive information.

A popular technique for answering queries using views is to first rewrite the queries using the views and then to evaluate the queries using the data in the views. Data-integration applications are mostly focused on obtaining maximally-contained rewritings [30, 28, 16, 21, 4] (or equivalent rewritings if there are any), whereas in security applications it is important to check whether there are contained rewritings that convey too much information to unauthorized users. In both cases, a designer constructing views or a person performing a security audit of the system needs to determine what information is available from the views.

In this paper we consider queries and views that are expressed as select-project-join SQL (i.e., *conjunctive*) queries with arithmetic predicates. We investigate the problem of rewriting queries using views in a more general setting than just obtaining equivalent or maximally contained rewritings. Specifically, the problem involves finding whether there exist rewritings that (1) are *contained* in the query (that is, users obtain *some* correct answers), (2) *contain* the query (the answers obtained by users include *all* correct answers), or (3) *overlap* with the query (i.e., users obtain some correct and some incorrect answers). Existing work has addressed the problem of finding equivalent or maximally-contained rewritings of queries using views. At the same time, the problem under the more general setting described above has only recently started to attract interest [15]. We investigate the complexity issues and present sound and complete algorithms that address some of these more general problems. Table 1 gives a summary of our results and contribution.

The following example shows some of the issues involved in traditional and new applications of the problem of rewriting queries using views.

*Example* 1. Suppose a census database stores information about ages and incomes of the population in a `Person` relation that has attributes `CensusRecordID, Age,` and `HouseholdIncome`, and stores information about addresses in an `Address` relation with attributes `CensusRecordID, StreetAddress, ResidencePhone, City,` and `State`.

Suppose one or more of the database users would like to obtain phone numbers of retirement-age people (that is, `Age >= 62`) in a given city, such that the household income of those people is at least $70K. This

| Results | Contained | Containing | Overlapping |
|---|---|---|---|
| **Decidability** | CQSI | CQAC | open |
| **Complexity** | CQ: NP | CQAC: NP | open |
| **Algorithms** | Finds nontrivial CR | Finds MiCR | Heuristic |
| **Max/Min** | MCR | MiCR | MOR |
| **Previous Work** | MCR [21] | MiCR [15] | new, this paper |
| **Applications** | Security, Privacy, Peer to Peer | Security, Privacy, Web Applications | Web Applications |

Table 1: Our results, previous work, and applications.

inquiry could be expressed in SQL as follows:

```
Q: SELECT ResidencePhone FROM Person P, Address A
   WHERE P.CensusRecordID = A.CensusRecordID
   AND City = 'Raleigh' AND State = 'NC'
   AND Age >= 62 AND HouseholdIncome >= 70000;
```

Suppose the database administrator has defined three views to anonymize the information in the census database: View `V1` returns the record IDs and age information for all people whose income exceeds \$100K. View `V2` returns the values of record ID, phone number, and city information for all people in North Carolina. Finally, view `V3` gives the phone numbers and city/state information for all people older than 50. As an illustration, the view `V3` could be defined as follows:

```
V3: SELECT ResidencePhone, City, State
    FROM Person P, Address A
    WHERE P.CensusRecordID = A.CensusRecordID
    AND Age > 50;
```

We now discuss three different scenarios for obtaining the answers to the user query `Q`.

*Scenario 1: Contained rewritings.* Suppose answers to views `V1` and `V2` are available. The users can combine the two views in this query `Q1`:

```
Q1: SELECT V2.ResidencePhone FROM V1, V2
    WHERE V1.CensusRecordID = V2.CensusRecordID
    AND V2.City = 'Raleigh' AND Age >= 62;
```

As the view `V1` returns information for just people whose household income is strictly greater than \$70K, by using the two views our users can get only a *subset* of the answer to their query `Q` given by a contained rewriting. Answers to contained rewritings do not have false positives, but may contain *false negatives* — missing answers to the original query on the database. In this example, the answer to `Q1` will not contain phone numbers of any people with household incomes between \$70K and \$100K.

*Scenario 2: Containing rewritings.* If user data access is defined using the view `V3` only, then all the users can get is a *superset* of the answer to the query `Q`. Indeed, `V3` could be used to return the phone numbers of people in Raleigh whose age is greater than 50:

```
Q2: SELECT ResidencePhone FROM V3
    WHERE City = 'Raleigh' AND State = 'NC';
```

The query `Q2` *contains* the user query `Q` — that is, on any database the answer to `Q2` is guaranteed (under the closed-world assumption [1], such as on a company database) to include the answer to `Q`. At the same time, depending on the contents of the relations `Person` and `Address`, the answer to `Q2` may or may not include *false positives*, such as information about people in Raleigh whose age is less than 62.

*Scenario 3: Overlapping rewritings.* If users can access just the view `V3` under the open-world assumption [1, 26] (such as in an information-integration setting), the best they can hope for is to obtain a rewriting that *overlaps* with the query `Q`. For the rewriting, we can use the query `Q2` of Scenario 2. Under the open-world assumption, `Q2` can return both false positives — such as phone numbers of people whose age is less than 62 — and false negatives, by missing information about higher-income people of retirement age.

We now list some technical challenges in determining whether there are rewritings for each scenario in the above example. In *Scenario 1*, one could choose to focus on finding maximally-contained rewritings (MCR) of the query `Q`. At the same time, the problem of finding MCRs has not been solved in the general case of conjunctive queries with arithmetic comparisons. In this paper (see Section 3) we study cases in which we do not know how to find an MCR but we show that we can determine whether there exists some non-empty contained (other than maximally contained) rewriting of the given query.

In *Scenario 2*, we would like to find a *containing* rewriting that gives a minimum number of false positives — a *minimally-containing rewriting* (MiCR) [15]. In this paper (see Section 4) we present efficient sound and complete algorithms for finding MiCRs of queries with arithmetic comparisons using views with arithmetic comparisons.

In this work we provide multiple definitions of overlapping rewritings and discuss their usefulness. The challenge is to come up with a definition such that the results are of practical use. In this case, it is important to find the right balance between reducing false positives and false negatives and quantifying what a good solution is. We study this problem in Section 5.

Algorithms to address the above technical challenges depend upon efficient algorithms for checking query containment. It is known from existing work on query containment [24, 19, 35] that adding arithmetic comparisons to queries and views makes these problems significantly more challenging. In this work we identify special cases of the problem in presence of arithmetic comparisons and provide efficient sound

and complete algorithms.

The remainder of the paper is as follows. In Section 2 we provide the background and formal definitions. Section 3 contains complexity results on contained rewritings and describes an algorithm for finding conjunctive contained rewritings of select-project-join queries with arithmetic comparisons. In Section 4 we present results on finding containing rewritings, and Section 5 contains results on overlapping rewritings. Section 6 describes an implementation and experimental results on finding minimally containing rewritings of queries.

### Related Work

The problem of using views in query answering [25] is relevant in applications in information integration [14, 20, 22, 26, 34], data warehousing [36], web-site design [17], and query optimization [13, 25, 37]. Algorithms for finding rewritings of queries using views include the bucket algorithm [18, 26], the inverse-rule algorithm [7, 16, 31], the MiniCon algorithm [30], and the shared-variable-bucket algorithm [28]; see [21] for a survey. Almost all of the above work focuses on investigating rewritings that are contained in the query and mainly maximally contained or equivalent rewritings [34, 1], as it takes its motivation mostly from information integration and optimization.

Since we consider rewritings that may return some false positives, or false negatives, or both, our work can also be described as approximate answering of queries (see [2, 9, 11, 29] and references therein) on databases using views. Approximate query answering is useful when exact answers to the queries cannot be found, and the user would rather have a good-quality approximate answer returned by the system.

In [15] the authors consider answering queries using views via equivalent, contained, and containing rewritings, in the presence of access patterns, integrity constraints, disjunction, and negation. The paper reports complexity results and presents algorithms for producing rewritings in that general setting. The algorithms depend on the behavior of the chase algorithm and especially hold when the chase terminates and its result is not too large.

Other related work includes the results of [32], where query-rewriting techniques are used for fine-grained access control, and the work by Miklau and Suciu [27], who have performed a formal probabilistic analysis of information disclosure in data exchange under an assumption of independence among the relations and data in a database. Related work in security and privacy includes [10, 33].

## 2 Preliminaries

### 2.1 Queries, Containment, and Views

We consider *conjunctive queries with arithmetic comparisons* (CQAC) (i.e., select-project-join SQL queries) of the form:

$$q(\bar{X}) \ :- \ p_1(\bar{X}_1), \ldots, p_k(\bar{X}_k), C_1, \ldots, C_m.$$

In each *relational* subgoal $p_i(\bar{X}_i)$, predicate $p_i$ represents a *base relation*, and every argument in the subgoal is either a variable or a constant. Each *arithmetic comparison* (AC) subgoal $C_i$ is of the form $X\theta Y$ or $X\theta c$,[1] where the comparison operator $\theta$ is one of $<, \leq, >, \geq$. (We assume that database instances are over densely totally ordered domains.) A variable is called *distinguished* if it appears in the query head. When there are no comparison subgoals in a query, then we refer to it as a conjunctive query (CQ).

We will use the term *semi-interval CQAC* (and its abbreviation SI-CQAC) to refer to conjunctive queries with arithmetic comparisons, where each comparison is either one of $X < c$, $X \leq c$ (*left semi-interval*) or one of $X > c$, $X \geq c$ (*right semi-interval*). We also consider unions of conjunctive queries, with or without comparisons.

*Definition* 1. A query $Q_1$ is *contained* in a query $Q_2$, denoted $Q_1 \sqsubseteq Q_2$, if for all databases $D$, the answer to $Q_1$ on $D$ is a subset of the answer to $Q_2$ on $D$, that is, $Q_1(D) \subseteq Q_2(D)$.

Chandra and Merlin [12] show that a CQ $Q_1$ is contained in another CQ $Q_2$ if and only if there exists a *containment mapping* from $Q_2$ to $Q_1$. The containment mapping maps the head and all the subgoals of $Q_2$ to $Q_1$; it maps each variable to a single variable or constant, and maps each constant to the same constant. The containment test for CQACs is more complicated. There are two containment tests in the literature [24, 19]; we will describe them very briefly, for more details see, e.g., [5]. The first test for CQACs uses the notion of *canonical database*: For each relational subgoal $p_i(\bar{X}_i)$ of a query $Q$, a canonical database for $Q$ has one tuple $t$ in the base relation $P_i$, such that $t$ is a list of "frozen" variables (i.e., assignments of the variables to constants) and constants in $\bar{X}_i$. We define one canonical database for each total ordering of the variables and constants of $Q_1$ that satisfies the ACs of the query. Thus, we have as many canonical databases as we have such orderings. The containment test says that a query $Q_1$ is contained in query $Q_2$ if and only if $Q_2$ computes its head tuple on all canonical databases of $Q_1$.

The second containment test is as follows:

THEOREM 1. *$Q_1 \sqsubseteq Q_2$ if and only if the following logical implication $\phi$ is true:*

$$\phi : \beta'_1 \Rightarrow \mu_1(\beta'_2) \vee \ldots \vee \mu_k(\beta'_2)$$

*where $\mu_i$'s are all containment mappings from $Q'_2$ to $Q'_1$ and $\beta'_i$ is a conjunction of all arithmetic comparisons in $Q'_i$. That is, the comparisons in the normalized query[2] $Q'_1$ logically imply (denoted "$\Rightarrow$") the disjunction of the images of the comparisons of the normalized query $Q'_2$ under all the mappings $\mu_i$.*

[1] We use uppercase letters to denote variables and lowercase letters for constants.

[2] An equivalent *normalized version* of a CQAC query $Q$ does not have constants or repetitions of variable names in relational subgoals and has compensating built-in equality conditions.

When there exists a containment mapping $\mu_i$ such that the right-hand side of $\phi$ is reduced to $\mu_i(\beta_2')$, we say the *homomorphism* property holds. Note that in general, the queries have to be normalized before implication can be checked. Afrati et al. [5] have shown that when the homomorphism property holds, the implication can be checked directly on unnormalized queries. Thus, checking CQAC containment is less complex in that case. In Section 4 we use the homomorphism property to design an algorithm for finding minimally containing rewritings.

### 2.2 Rewriting Queries using Views

We consider the problem of obtaining answers to queries using views where queries are CQ or CQAC and where views, denoted $V, V_1, \ldots, V_m$, are defined on the base relations by CQ or CQAC queries.

We consider the problem under the closed-world assumption [1] (i.e., for a given database, each view instance stores all the tuples satisfying the view definition), as well as under the open-world assumption [1, 26] (i.e., a view instance might store only some of the tuples satisfying the view definition).

Suppose we are given a query $Q$, a database $D$, and a set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$. We seek an answer to $Q$ on $D$ using some *rewriting* $R$ in terms of $\mathcal{V}$ — that is, $R$ is a query defined in terms of the relation names in $\mathcal{V}$. We compute an answer to $R$ on $D$ as follows. The database $D$ determines a database $D_{\mathcal{V}} = \{ans(V_1), \ldots, ans(V_m)\}$, where the relation $ans(V_i)$, $i \in 1, \ldots, m$, is (possibly part of) the answer to the query $V_i$ on $D$.[3] Then the answer to $R$ with respect to $D$ and $\mathcal{V}$ is the result $R(D_{\mathcal{V}})$ of computing an answer to $R$ on the database $D_{\mathcal{V}}$. We consider four types of rewritings $R$ of a query $Q$ using views $\mathcal{V}$:

*Definition* 2.
1. $R$ is a *contained rewriting* of $Q$ using $\mathcal{V}$ if on all databases $D$, $R(D_{\mathcal{V}}) \subseteq Q(D)$.
2. $R$ is a *containing rewriting* of $Q$ using $\mathcal{V}$ if on all databases $D$, $Q(D) \subseteq R(D_{\mathcal{V}})$.
3. $R$ is an *equivalent rewriting* of $Q$ using $\mathcal{V}$ if on all databases $D$, $Q(D) = R(D_{\mathcal{V}})$.
4. $R$ is an *overlapping rewriting* of $Q$ using $\mathcal{V}$ if on all databases $D$, if $Q(D) \bigcap R(D_{\mathcal{V}}) = \phi$ then $Q(D) = \phi$, where $\phi$ is the empty set.

Intuitively, with respect to a given query, contained rewritings cannot contain false positives, containing rewritings cannot have false negatives, equivalent rewritings do not contain either false positives or false negatives, and overlapping rewritings may have both. (See Example 1 in Section 1 for illustrations.) In this paper we will use the term *rewriting* to mean contained, containing, or overlapping rewriting of a given query; we will specify the rewriting type whenever it is not obvious from the context.

[3]Under the open-world assumption, $ans(V_i)$ may be a proper subset of the answer $V_i(D)$ to $V_i$ on $D$.

*Definition* 3. For a CQAC rewriting $R$ in terms of CQAC views $\mathcal{V}$, an *expansion* $R^{exp}$ of $R$ is obtained by replacing each view subgoal in $R$ by the relational and comparison subgoals in the view definition. Existentially quantified variables in the definitions of the views occurring in $R$ are replaced by fresh variables in $R^{exp}$. For rewritings that are unions of CQACs, the expansion is the union of the expansions of the CQACs contained in the rewriting.

The following theorem gives tests for whether a CQAC rewriting $R$ in terms of views $\mathcal{V}$ is contained, containing, or overlapping with respect to a CQAC query $Q$.

THEOREM 2. *Let $Q, V_1, \ldots, V_m$ be CQAC queries in terms of base relations, and let $R$ be a CQAC rewriting of $Q$ in terms of $V_1, \ldots, V_m$. Then:*

1. *$R$ is a contained rewriting of $Q$ if $R^{exp} \sqsubseteq Q$.*
2. *$R$ is a containing rewriting of $Q$ if $Q \sqsubseteq R^{exp}$.*
3. *$R$ is an overlapping rewriting of $Q$ if there exists a query $Q'$ defined on base relations, such that $Q' \sqsubseteq Q$ and $Q' \sqsubseteq R^{exp}$. Furthermore, there exists at least one database $D$, such that $Q'(D)$ is not empty.*

We consider query answering and rewriting in settings where user access to stored data is defined using views; we refer to these settings as *view-based data access*. Given a database schema $\mathcal{S}$, a user query $Q$ defined on the schema $\mathcal{S}$, a set $\mathcal{V}$ of views defined on $\mathcal{S}$ (the views define the users' data access), and a type of rewriting (contained, containing, or overlapping), we determine whether there exists a rewriting $R$, of the given type, of the query $Q$ in terms of the views $\mathcal{V}$.

## 3 Contained Rewritings

The research problem of finding maximally contained rewritings has received a lot of attention, mainly because of its importance in information integration.

*Definition* 4. A contained rewriting of a query $Q$ using a set of views $\mathcal{V}$ with respect to some query language $\mathcal{L}$ is a *maximally contained rewriting* (MCR) if it contains as a query all other contained rewritings of $Q$ using $\mathcal{V}$ in $\mathcal{L}$.

It is known [5] that when queries contain arithmetic comparisons, it is not easy to find an MCR of a query using the views; in some cases and for certain languages such a rewriting does not even exist. For instance, for certain cases of conjunctive queries with semi-interval arithmetic comparisons we cannot find a MCR in the language of unions of CQAC views, but we can find MCR in recursive datalog with ACs [4]. In addition, as shown in [1], an MCR does not exist in certain cases of CQACs and for languages that are polynomially computable.

At the same time, easy subcases are known; for instance, if containment can be checked using a single containment mapping (homomorphism property), then we can constuct an MCR for a CQAC query and CQAC views [4, 5]. The homomorphism property does not hold in the general case when the queries and views are CQACs [24]. In that case, the problem of containment of two queries is harder ($\Pi_2^P$-complete) than in the case where the queries are simple conjunctive queries (NP-complete) [35]. Typically, the problem of finding MCRs of queries using views is in the same or higher complexity class as the problem of checking the containment of queries, for a particular language.

Thus for the general case there is no known algorithm for finding MCRs for CQAC queries and views [4]. In various scenarios, for instance in security applications, we are interested in finding whether a contained rewriting exists, as opposed to obtaining the maximally contained rewriting. To preserve security and privacy of the data, the owner of a database may want to check that answering a set of high-security queries is not possible using just the available views. If a contained rewriting exists, then some answers to the query are available via the views, even though all the answers are not. In those cases where an algorithm to find MCRs exists, this problem is solved: If the algorithm finds a non-empty MCR, then a contained rewriting exists, otherwise it does not.

In this section we study the problem of finding contained rewritings for the case when we do not know how to find an MCR. Because any query that returns the empty set on all databases is contained in all queries, we are interested in finding *nontrivial* contained rewritings — rewritings that have a nonempty answer on at least one database.

**Problem** (*CQAC Contained Rewriting*): Given a conjunctive query with arithmetic comparisons (CQAC) $Q$, a set of CQAC views $\mathcal{V}$, and a language $\mathcal{L}$, is there a nontrivial contained rewriting in $\mathcal{L}$ of $Q$ using $\mathcal{V}$? When both views and query are CQs, then we have the *CQ Contained Rewriting Existence* problem.

### 3.1 Decidabillity and Complexity

Deciding existence of a nontrivial contained rewriting turns out (not surprisingly) to be easier than computing MCR. In this subsection we present decidability and complexity results. At the same time, decidability of the general case remains open. We obtain decidability for two special cases: (1) when all view definitions do not have nondistinguished variables, and (2) when we use only semi-interval comparison subgoals. We show that for CQs the problem is NP-complete.

The following theorem shows that when the query and the views are CQACs and the view definitions have no nondistinguished variables, then the existence of a contained rewriting can be checked.

THEOREM 3. *Let $Q$ be a CQAC query, and let $\mathcal{V}$ be a set of CQAC views that do not have nondistinguished variables. If there is a nontrivial contained rewriting of $Q$ using $\mathcal{V}$, then there is a nontrivial contained rewriting that uses at most $M$ subgoals, where $M$ is the number of relational subgoals in $Q$.*

This result follows from the observation that if there exists a contained rewriting of $Q$ using $\mathcal{V}$, then there exists a contained rewriting whose ACs define a total order on the variables and constants of the rewriting.

PROOF. If there is a contained rewriting $R$, then there is also a contained rewriting $R'$ with ACs defining a total order on the variables and constants of the rewriting (any total ordering on the variables and constants of $R$ that satisfies the ACs in $R$ is such a rewriting). Finding $R'$ is possible because all the view variables are distinguished and we can add ACs that impose a total order on all the variables and constants of $R$. The expansion of $R'$ is contained in the query and in fact one mapping proves containment (because of the total order). Hence, we can delete all view subgoals in $R'$ except those view literals of $R'$ whose expansions have subgoals that are images of the query's subgoals under this single containment mapping. Even after the deletion, the same containment mapping holds between the query and the rewriting because none of the views whose expansion contributes target subgoals of the containment mapping have been removed. The query has $M$ subgoals. Thus, the number of view literals remaining $R'$ after removing the extras, is at most $M$. □

The following theorem shows that in the case where the query and views are conjunctive with semi-interval arithmetic comparisons, if there is a contained rewriting then there is one that uses only semi-interval arithmetic comparisons and is of bounded size.

THEOREM 4. *Let $Q$ and $V$ be a query and views which are CQs with semi-interval (SI) arithmetic comparisons. If there is a nontrivial contained rewriting of the query using the views then there is a nontrivial contained rewriting that (1) uses only semi-interval arithmetic comparisons, and (2) uses at most a number of relational subgoals that is exponential only in the maximum arity of the head in a view definition.*

Note that this bound on the number of subgoals is higher in this case than for views without nondistinguished variables, see Theorem 3. The result of Theorem 4 follows from two claims that we show in the proof: (1) There is no need to consider rewritings that use any new constants (other than the constants in the query and views). (2) It is enough to consider contained rewritings that use only SI comparisons.

PROOF. First we show that we do not need to consider rewritings that use new constants (other than the constants in the query and views):

Claim 1: If there is a contained rewriting $R$ that uses other constants than the constants in the query and views, then there is another contained rewriting $R'$ that uses only the constants in the query and views.

To prove the claim, we argue as follows. Let $c$ be a constant used in $R$ and $c$ does not occur in the query or any view. Let $c_1$ be the greatest constant in the query which is less than $c$ and let $c_2$ be the smallest constant in the query which is greater than $c$. When the query contains no constant greater than $c$, set $c_2$ to positive $\infty$. When the query contains no constant less than $c$, set $c_1$ to negative $\infty$. We produce $R'$ from $R$ by replacing each comparison predicate of the form $X < c$ or $X \leq c$ by $X < c_2$ and by replacing each comparison predicate of the form $X > c$ or $X \geq c$ by $X > c_1$. Subgoals that use $\infty$ are just dropped. It is clear that the expansion of $R'$ is contained in the query.

We show that if there exists a contained rewriting then there exists one that uses only SI comparisons.

Claim 2: If there exists a contained rewriting then there exists one which uses only SI comparisons.

Let all constants used in the query and views be $c_1 \leq c_2 \ldots \leq c_m$; we consider all intervals $(-\infty, c_1], (c_1, c_2], \ldots, (c_m, \infty)$. Let $I$ be the set that contains exactly those intervals. To prove this claim we argue as follows. We add (to the contained rewriting) SI predicates that do not contradict the predicates in the expansion of the rewriting such that all variables are placed by these new predicates in one of the above intervals. There is such a set of SI predicates because otherwise the rewriting is either empty or not contained in the query, because there must exist at least one canonical database of the expansion of the rewriting, such that the heads of both the expansion of the rewriting and of the query are computed on this database — if such a database does not exist then the rewriting is either empty (i.e., the head of the rewriting is not computed on any of such databases) or not contained in the query (i.e., the head of the query is not computed on any of such databases). And since a canonical database induces a total order on the frozen variables, we use this total order to add SI subgoals in the rewriting by just copying the interval in which the particular frozen variable belongs.

Thus, we produce rewriting $R'$, which is contained in the query because $R'$ is contained in the original rewriting. We drop the non-SI predicates from the rewriting to obtain a *new* rewriting $R$. We need to prove that $R$ is contained in the query. Consider all canonical databases of the expansion of $R$ on all orderings of the variables and constants. The query applied on each of these databases produces the head tuple in the answers because: A canonical database of the expansion of $R$ is such that there is a canonical database of the expansion of $R'$ in which the only difference is that some variables that are frozen in a particular interval are placed (in this interval) in a different order among them. This however does not affect the computation of the query because the query only contains SI comparisons.

Now, we are ready to prove the theorem. Let $R$ be a contained rewriting. For each subgoal $g$ of $R$, we define its signature to be a vector with two entries for each variable in $g$, one entry for the LSI AC in $g$ that uses this variable and one entry for the RSI AC similarly (one or both of the entries may be empty in case the particular variable is not used in such an AC). Let $d$ be the maximum arity of a view head and $m$ be the number of views and $c$ be the number of constants used in the query and views definitions. There are at most $md!$ distinct subgoals in $R$ up to renaming of variables.

For each subgoal there are at most $c^{2d}$ distinct signatures.

Thus, there are two subgoals which are identical up to variable renaming and have the same signature. Create a new rewriting $R_2$ by collapsing (i.e., equate component-wise the variables) of these two subgoals, say we keep subgoal $g_1$ and delete subgoal $g_2$ and also in the other subgoals, all variables that appeared in $g_2$ are replaced by the corresponding variables of $g_1$.

We claim that $R_2$ is a nontrivial contained rewriting. To prove, observe that any canonical database of the expansion of $R_2$ is a canonical database of the expansion of $R$, hence, the query computes the head tuple of the rewriting on all canonical databases of $R_2$. (It is easy to see that any canonical database of the expansion of $R_2$ uses a total order which does not violate any of the ACs in $R$ either. The identified variables that produce $R_2$ from $R$ are chosen so that they do not contradict any of the ACs in $R$.)

Finally as we need at most $n$ subgoals in the rewriting ($n$ is the number of relational subgoals in the query), we conclude that we can find a contained rewriting with at most $n + c^{2d}md!$ relational subgoals.

□

In the case of CQ queries and views (i.e., without comparison subgoals) the problem is NP-complete:

THEOREM 5. *The problem of finding a nontrivial contained rewriting of conjunctive queries using views that are conjunctive queries is NP-complete.*

PROOF. **Proof Sketch**

1. The problem is in NP. If the answer is "yes" then the certificate is a contained rewriting together with a containment mapping from the query subgoals to the the subgoals of the expansion of the rewriting. If the size of the rewriting is polynomial then the size of the containment mapping is also polynomial and checking that it is a containment mapping can be done in polynomial time. It remains to be proven that the size of the rewriting is polynomial. Consider any contained rewriting $R$ and the containment mapping which proves that it is contained in $Q$. The target subgoals of this mapping are at most equal to the number of subgoals of the query. Obtain a rewriting $R'$ from $R$ by dropping all view subgoals that do not contribute targer subgoals. $R'$ is also a contained rewriting.

2. The problem is NP-hard. By reduction from CQ containement. Given queries $Q_1$ and $Q_2$, we construct $Q_1'$ and $Q_2'$: The heads in both use a new predicate and variable $p(X)$. The body of $Q_1'$ and $Q_2'$ each contain all the subgoals of $Q_1$ and $Q_2$ respectively and a

new predicate $p'(X, t_i)$ where $t_1$ is the tuple of head arguments of $Q_1$ and $t_2$ is the tuple of head arguments of $Q_2$ respectively. Then $Q_2$ is contained in $Q_1$ iff $Q_1'$ has a contained rewriting which uses as view $Q_2'$. To prove, we need to argue that the body of such a rewriting would be exactly one occurrence of $Q_2'$ as a view — and no other view subgoals. This is easily shown by observing that any containment mapping from $Q_1'$ to the expansion of any rewriting will only use target subgoals from one view due to the predicate $p$ which occurs only once in $Q_1'$. □

Note that this result holds in all cases where the homomorphism property holds between the query and the expansion of the rewriting. Afrati et al. [5] have shown examples where the homomorphism property does not hold, identified the conditions under which the homomorphism property holds for CQAC queries, and given an algorithm to find the MCR for the cases where the homomorphism property holds [4].

### 3.2 Algorithm

In the previous subsection, we proved two results: (1) Unlike finding maximally contained rewritings, it is decidable in many special cases whether there exists a contained rewriting given CQAC queries and views. (2) The problem is still NP-hard, even for queries and views without inequality comparisons. In this subsection we present a heuristic that checks whether there exists a nontrivial (that is, not trivially empty) contained rewriting in the case of CQAC query and views. It is known that the containment test (which provides the basis for algorithms for finding rewritings) is more complicated in the CQAC case than in the CQ case. The following Proposition 1, however, says that (1) the complement of the CQAC Contained Rewriting problem can be reduced to the complement of the CQ Contained Rewriting problem (an admittedly much simpler problem), and (2) the search space of candidate contained rewritings can be restricted to rewritings whose ACs define a total order on the variables of the rewriting. Still this does not buy us the homomorphism property, as this example shows:

*Example* 2. Consider a query $Q$ and two views:

$$\begin{array}{ll} q() & :- p(X,4),\ X < 4. \\ v_1(X) & :- p(3,X). \\ v_2(X) & :- p(X,4). \end{array}$$

In this rewriting $R$,

$$r() \quad :- v_1(X),\ v_2(X),\ X \leq 4,$$

the homomorphism property does not hold.

In many cases, even though a nontrivial contained rewriting of a query using views does not exist, we are able to check that by just looking at the query and views without arithmetic comparisons. However, such a test is not straightforward, as the following definitions and example show.

Let $Q = Q_0 + \beta$ be a query and $\mathcal{V}$ be a set of views $V_i = V_{i0} + \beta_i,\ i = 1, \ldots, k$. Consider the query $Q_0$ comprising the relational subgoals of $Q$ and a set of views $\mathcal{V}_0$ with views as in $\mathcal{V}$ but defined using only the relational subgoals in their bodies. The question is whether we can prove the following: If there exists a contained rewriting of $Q$ using $\mathcal{V}$, then there exists a contained rewriting of $Q_0$ using $\mathcal{V}_0$. Unfortunately, this result does not hold; here is a counterexample:

*Example* 3. For a query $Q$ and view $V$,

$$\begin{array}{ll} q() & :- p(X,X), \\ v(X,Z) & :- p(X,Y),\ r(Z),\ X \leq Y,\ Y \leq Z. \end{array}$$

a contained rewriting is:

$$q'() \quad :- v(X,Z),\ Z \leq X.$$

But by removing the ACs from the view we obtain

$$\begin{array}{ll} q & :- p(X,X), \\ v(X,Z) & :- p(X,Y),\ r(Z); \end{array}$$

a rewriting is not possible.

But with a slight modification in the definition of the views we obtain the following result. Before we state the result, we need the following definitions from [5] which we will use in our algorithms here.

The *shared-variable requirement* is as follows: If one occurrence of a nondistinguished query variable $X$ maps to a nondistinguished view variable $Y$, then all occurrences of $X$ must map to $Y$. A corollary of this condition is that all query subgoals that contain $X$ must be covered by the same view.

A nondistinguished variable $X$ of a view $V$ is *exportable* if and only if there exists a head homomorphism $h$ such that in the expansion of $h(V)$, $X$ can be equated to a distinguished variable in $h(V)$ by either using the ACs in the view or by adding ACs on distinguished variables in $h(V)$. For example, variable $Y$ in the view definition in Example 3 is an exportable variable because if we equate $X = Z$ then the ACs in the view definition imply $Y = X = Z$.

PROPOSITION 1. *Let $Q = Q_0 + \beta$ be a query and $\mathcal{V}$ be a set of views $V_i = V_{i0} + \beta_i,\ i = 1, \ldots, k$. Consider a query $Q_0$ comprising the relational subgoals of $Q$ and a set of views $\mathcal{V}_0$ with views as in $\mathcal{V}$ but (1) defined using only the relational subgoals in their bodies, and (2) with additional head variables, those that are exportable. Suppose there exists a contained rewriting of $Q$ using $\mathcal{V}$. Then:*

1. *There exists a contained rewriting of $Q_0$ using $\mathcal{V}_0$.*
2. *There exists a contained rewriting of $Q$ using $\mathcal{V}$ with ACs that define a total order on its variables and with relational subgoals that define a contained rewriting of $Q_0$ using $\mathcal{V}_0$.*

The proof of Proposition 1 is based on containment mappings for $Q_0$ and $\mathcal{V}_0$, which (the mappings) are made possible by using the exportable variables of the views $\mathcal{V}$ in the heads of the views $\mathcal{V}_0$.

PROOF. For the first item, consider a contained rewriting $R$ and any canonical database of its expansion. Clearly there is a mapping from the query relational subgoals to the tuples of this database. This mapping produces a containment mapping from $Q_0$ to the following rewriting $R_0$ of $Q_0$ using $V_0$: Retain in $R_0$ those subgoals of $R$ that provide targets to this mapping. Some fresh variables from the expansion may be exported due to ACs in the query definition and if we do not use these ACs exporting them is not possible. These variables however are already in the view heads in the definitions of $V_0$ hence we may use them as distinguished variables.

The proof of item 2 uses similar techniques that have been used here and the argument for item 1. Note that the contained rewriting of $Q_0$ using $V_0$ may have many redundant subgoals. □

We now outline an algorithm for checking containment of rewritings in queries; the algorithm uses properties of containment tests to prune the search space by (1) testing first for contained rewritings of $Q_0$ using $\mathcal{V}_0$ and halting if none exists, and by (2) considering containment checks in a systematic way, so that it does not repeat unnecessarily checks that can be deduced from the previous iteration.

Due to lack of space, we give here just an intuition for the algorithm. First, the algorithm tests all views for candidacy in a contained rewriting (CR). A view is a *candidate view* if it covers at least one relational subgoal of the query, and the shared-variable (SV) condition is satisfied. Thus, the first stage of the algorithm checks whether there exists a CR of $Q_0$ using $\mathcal{V}_0$. If there exists none, then the algorithm returns "no". Otherwise, it starts with any CR $R_0$ of $Q_0$ using $\mathcal{V}_0$. Then it considers all rewritings with the relational subgoals of $R_0$ and with added ACs that define a total order on the variables, for all total orders.

For each such rewriting it does the following: It checks for containment in the query $Q$ using the canonical-database test. If containment holds, then the algorithm returns "yes", otherwise it keeps a record of the canonical databases that did not pass the test. In the next stage, the algorithm adds to the rewritings considered in the previous stage one candidate view (for all candidate views not yet added) and checks containment in the query. The modified check at this stage is restricted only to those canonical databases — in fact, to their extensions, as a new view is now added — that did not pass the test in the previous stage. (Adding a new view to the rewriting does not break containment on those canonical databases that *did* pass the test in the previous stage. The reason is, if the rewriting with the newly added view has a nonempty answer on one such database, then we still get the frozen heads of the view, of the rewriting, and of the query, and thus the containment test is succesful on those databases. If the rewriting with the newly added view has an empty answer on one of those databases, then the frozen head of the rewriting cannot be computed on the database, therefore the answers to the rewriting are trivially contained in the answers to the query on this database.) Again, for the next stage the algorithm keeps record of the canonical databases that did not pass the test.

In a little more detail: In stage $n$, the algorithm considers all rewritings of stage $n-1$, say $R$ is such a rewriting. $R$ comes with a set of canonical databases (of its expansion) that did not pass the containment test. For all candidate views not yet added in $R$, the algorithm forms rewritings $R'$, where each $R'$ is $R$ with an additional subgoal that comes from one of the candidate views. Then the algorithm checks each $R'$ for containment. Observe that the algorithm is designed to start with a contained rewriting of $Q_0$ using $\mathcal{V}_0$, say with $k$ view subgoals. Then stage 2 is checking for containment all rewritings with $k+1$ view subgoals, stage 3 is checking for containment all rewritings with $k+2$ view subgoals, and so on.

*Example* 4. We refer to Example 3. For $Q_0$ we have

$$q_0() \quad :- p(X, X).$$

For $V_0$ we have

$$v_0(X, Z, Y) \quad :- p(X, Y),\ r(Z).$$

because variable $Y$ is exportable — that is, $Y$ can be equated to a distinguished variable by adding an equality between distinguished variables, in this case by adding the equality $X = Z$. A contained rewriting of $Q_0$ using $V_0$ is:

$$q_0'() \quad :- v_0(X, Z, X).$$

$X = Z$ is a total order among the variables of $V$ that produces a rewriting of $Q$ using $V$. Hence we have found a rewriting of $Q$:

$$q'() \quad :- v(X, Z),\ X = Z.$$

THEOREM 6. *Let query and views be CQAC.*

- *If the algorithm halts then it produces a contained rewriting if there exists one.*
- *In the case of CQSI query and views or when view definitions have no nondistinguished variables, the algorithm produces a nontrivial contained rewriting if there exists one.*

## 4 Containing Rewritings

In this section we discuss *containing rewritings* of queries using views (cf. [15]) and describe an algorithm to find minimally containing rewritings under the closed-world assumption.

In many data-use scenarios, including users browsing the Web or looking for investment products in a database using some criteria, if the system cannot generate an exact (equivalent) rewriting of a user query using the available views, the user would be satisfied with a rewriting that contains all the correct answers to the query with a minimum number of "false positives." We refer to such rewritings as *minimally containing rewritings* (MiCR) of a query using views.

*Definition* 5. A query $Q'$ is a *minimally containing rewriting* of a query $Q$ using a set of views $\mathcal{V}$, if and only if: (1) $Q'$ is a containing rewriting of $Q$, and (2) there exists no containing rewriting $Q''$ of $Q$ using $\mathcal{V}$, such that the expansion of $Q''$ properly contains the expansion of $Q'$.

Under the open-world assumption (OWA), views are incomplete; thus, when a containing rewriting of a query is evaluated using the tuples in views, the answer may not contain all the tuples that satisfy the original query, even though the expansion of the rewriting contains the original query by definition. Therefore, it makes sense to only talk of a MiCR under the closed-world-assumption (CWA). The rest of the discussion of MiCRs in this section assumes CWA.

We now give some containing rewritings of a query using a set of views and show a MiCR.

*Example* 5. Suppose we have a base relation `Person(CensusRecordID,Age,HouseholdIncome, ResidencePhone)`, and are interested in a query

$$q(R) \quad :- p(I, A, H, R),\ A \geq 62,\ H \geq 70K.$$

that asks for phone numbers of retirement-age people whose household income is at least \$70K.

Suppose we have four views on the base relation:

$$\begin{array}{ll} w_1(I, R) & :- p(I, A, H, R),\ A > 50. \\ w_2(I, A, R) & :- p(I, A, H, R). \\ w_3(I) & :- p(I, A, H, R),\ H \geq 70K. \\ w_4(I) & :- p(I, A, H, R),\ H > 60K. \end{array}$$

The following are some rewritings for the query using the views.

$$\begin{array}{ll} r_1(R) & :- w_1(I, R),\ w_2(I, A, R),\ A \geq 62. \\ r_2(R) & :- w_3(I),\ w_1(I, R),\ w_2(I, A, R),\ A \geq 62. \\ r_3(R) & :- w_1(I, R),\ w_4(I). \\ r_4(R) & :- w_3(I),\ w_1(I, R),\ w_4(I). \\ r_5(R) & :- w_3(I),\ w_1(I, R),\ w_2(I, A, R),\ w_4(I),\ A \geq 62. \end{array}$$

We can show that all of these rewritings contain the query $Q$. There are several useful observations on these rewritings. First, we cannot minimize $R_2$ to generate $R_1$, because the first subgoal of $R_2$ gives us a tighter containing rewriting than $R_1$ of the query $Q$. (The same observation holds on $R_4$ and $R_3$.) In this example, none of $R_1$, $R_3$, and $R_4$ is a MiCR rewriting given the views $W_1$ through $W_4$. The reason is, rewritings $R_2$ and $R_5$ also contain the query $Q$ and produce a smaller number of false positives than any other rewriting given here.

In general, a MiCR can be a union of conjunctive queries, as we show in the full version of the paper.

### 4.1 Non-union Condition

The complexity of the problem of finding all containing rewritings of a CQAC query using CQAC views is lower if the following condition holds:

**Non-union condition:** If a containing rewriting of a query exists, it can be expressed as a single conjunctive query and the language of unions of conjunctive views is not necessary. We will refer to this condition as the *non-union condition*.

THEOREM 7. *When two syntactic conditions $C1$ and $C2$ hold, the non-union condition also holds:*

- *C1: The query and all the views contain only left-semi-interval (LSI) (correspondingly only right-semi-interval, RSI) arithmetic comparisons,*
- *C2: If the query contains closed-LSI (-RSI) arithmetic comparisons, then the views do not contain open-LSI (-RSI) arithmetic comparisons.*

These conditions are not necessary but are sufficient conditions for the non-union condition to hold. These conditions were derived from the conditions provided in [5] under which the homomorphism property holds. Thus, not surprisingly, when the non-union conditions hold, the homomorphism property holds between the expansion of a containing rewriting and the queries.

THEOREM 8. *For a CQAC query $Q$ and a set of CQAC views $\mathcal{V}$, where both $Q$ and the views in $\mathcal{V}$ are defined using relational predicates from a set $P$, and such that $Q$ and $\mathcal{V}$ satisfy the non-union conditions $C1$ and $C2$ above. If there exists a containing rewriting $R$ of $Q$ using $\mathcal{V}$, such that $R$ is a union of CQAC queries, then (1) there exists a CQAC query $R'$, which is a containing rewriting of $Q$ using $\mathcal{V}$, and (2) the homomorphism property holds between $R'^{exp}$ and $Q$.*

The proof is by construction of $R'$ from $R$.

PROOF. Without loss of generality, we assume that the query $Q$ has left-semi-interval arithmetic predicates. Construct a canonical database $D$ by freezing each variable in $Q$ to a unique constant and by populating $D$ with the resulting tuples of the relational subgoals of $Q$. Specifically, the variables that appear in the arithmetic predicates of $Q$ are frozen as follows. For any AC ($X\ \theta\ c$) of $Q$ where $\theta$ is one of $<$ or $\leq$, freeze $X$ to a value $c - \epsilon_x$, where $\epsilon_x$ does not appear in $Q$ or in any view in $V$ and can be chosen to be arbitrarily small. Because the disjunctive rewriting $R$ contains $Q$, there must be a conjunct $R_1$ in $R$ that produces a tuple on $V(D)$ that is also produced by $Q(D)$. Therefore, there must be a containment mapping $\mu$ from the relational subgoals of $R_1$ to the predicates and constants in $D$ that produces the (frozen) head of $R_1$ on $D$. Because the mapping $\nu$ from the predicates and variables of $Q$ to the tuples and frozen constants in $D$ is bijective, a composition of $\mu$ with $\nu^{-1}$ is also a containment mapping from $R_1$ to the relational subgoals of $Q$.

We now construct from $R_1$ a new purely conjunctive rewriting $R_2$ by dropping any ACs in $R_1$; by construction, $R_1 \sqsubseteq R_2$. Because the query $Q$ and the set of views $V$ satisfy the conditions $C1$ and $C2$, and because all ACs in $R_2^{exp}$ come from the expansion of the views in $R_2$, we conclude that the homomorphism property holds between $R_2^{exp}$ and $Q$. (This conclusion follows from Theorem 4 in [5].)

Thus, to show containment of $Q$ in $R_2$, it remains to prove that $AC(Q) \Rightarrow AC(R_2)$. Due to condition $C1$

and because the query $Q$ and the views $V$ only contain left-semi-interval comparisons, all ACs in $R_2$ are left-semi interval inequality comparisons of the form $X\ \theta\ k$. We constructed the database $D$ by choosing the maximum values for the variables (that is, we chose each $c - \epsilon_x$ to be as large a value as possible). Computing $R_2$ on $D$ produces the (frozen) head of $R_2$. It follows that for each variable $X$ in $R_2$, such that $X$ maps to a variable $Y$ that occurs in an AC: $Y\ \theta\ c$ of $Q$, the range $(k)$ of $X$ is equal to or exceeds the maximum value $c$ for $Y$ in $D$, for otherwise computing $R_2$ on $D$ would not produce the (frozen) head of $R_2$. We conclude that $AC(Q) \Rightarrow AC(R_2)$. Thus, $R_2$ is $R'$ in the statement of the theorem.

(The only exception to $AC(Q) \Rightarrow AC(R_2)$ could be when $Q$ has an AC $Z \leq c$. Because $Z$ is frozen to $c - \delta$, it could be accepted by $R_2$ having an AC $T < c$, where $\mu(T) = Z$. In this case, the implication $(Z \leq c) \Rightarrow (Z < c)$ does not hold. However, this case is precluded by condition $C2$. Therefore, $R_2$ must be having an AC $T \leq c$, where $\mu(T) = Z$. Now, the implication $(Z \leq c) \rightarrow (Z \leq c)$ holds trivially.) □

In the rest of the section we assume that the non-union and the homomorphism property condition hold among containing and contained queries.

### 4.2 Decidability and Complexity

We now examine the complexity of computing containing rewritings and show that finding a containing rewriting of a query using views is NP-complete.

THEOREM 9. *(Rewriting that contains the query) Given a query $Q$ whose relational predicates belong to a set of predicates $P$, given a set of views $\mathcal{V}$ such that all views in $\mathcal{V}$ have relational predicates only from $P$, and given that $Q$ and $\mathcal{V}$ satisfy the non-union conditions $C1$ and $C2$, it is NP-complete to decide whether there exists a containing rewriting $R'$ of $Q$ using $\mathcal{V}$.*

The proof that the problem is in NP uses the result of Lemma 1; NP-hardness is by reduction from CQ containment.

PROOF. 1. The problem is in NP. By Theorem 8, the homomorphism property holds between some containing rewriting $R'$ and the query $Q$. If the answer is "yes" then the certificate is a containing rewriting together with a containment mapping from the expansion of the rewriting to the query. If the size of the rewriting is polynomial then the size of the containment mapping is also polynomial and checking that it is a containment mapping can be done in polynomial time because the homomorphism property holds. It remains to be proven that the size of the rewriting is polynomial. We do so in Lemma 1; see the statement and proof of the Lemma after the end of this proof.

2. The problem is NP-hard: By reduction from CQ containment. Given queries $Q_1$ and $Q_2$, we construct $Q_1'$ and $Q_2'$: The head in both is $p(X)$ such that $p$ and $X$ do not appear in $Q_1$ and $Q_2$. The body of $Q_1'$ and $Q_2'$ each contain all the subgoals of $Q_1$ and $Q_2$ respectively and a new predicate $p'(X, t_i)$ where $t_1$ is the tuple of variables in the head of $Q_1$ and $t_2$ is the tuple of variables in the head of $Q_2$ respectively. Then $Q_2$ is contained in $Q_1$ iff $Q_2'$ has a containing rewriting which uses as view $Q_1'$. To prove, we need to argue that such a rewriting would use only one copy of the view and one copy of the query. This is easily shown by observing that $Q_1'$ is the only view. $Q_1'$ has the variable $X$ in the head. Thus, all subgoals in a contining rewriting must be $Q_1'(X)$ modulo variable renamings of $X$. Any rewriting must have $Q_1'(X)$ in the body in order to be safe because there is no other view and no different head homomorphisms of $Q_1'$ is possible. If the view has conjunctions of $Q_1'$ with itself, it can be rewritten with only one copy of the view. Similarly, if the query has multiple copies of $Q_2'$, it is equivalent to a query with one copy of $Q_2'$. □

The proof of the theorem uses the following lemma. (The proofs of the theorem and the lemma are omitted due to space considerations and are available in [3].)

LEMMA 1. *If there exists a CQAC rewriting $R$ of a CQAC query $Q$, $Q \sqsubseteq R$, and if the homomorphism property holds between $R$ and $Q$, then there exists a CQAC rewriting $R'$ of $Q$, $Q \sqsubseteq R'$, such that the number of views in $R'$ is less than or equal to the arity of the head of the query $Q$.*

PROOF. Suppose the rewriting $R$ exists. We try to get a bound on the size of $R$. There exists a containment mapping $\mu$ from $R^{exp}$ to $Q$ because the homomorphism property holds between $R$ and $Q$. We obtain $R'$ from $R$ as follows: (1) drop all arithmetic predicates from $R$, and (2) drop all subgoals from the body of $R^{exp}$ except any one view subgoal from the body of $R$. Since any subgoal in $R'$ is also in $R$, $\mu$ proves containment of $Q$ in $R'$. Note that $R'$ is CQ and not CQAC because the arithmetic predicates from $R$ were dropped, hence the right-hand side of the implication is empty and so one containment mapping is enough to show containment.

Still, $R'$ may be unsafe. To ensure that the new rewriting is safe, we modify the construction of $R'$ as follows: (1) drop all arithmetic predicates from $R$, and (2) for each distinguished variable $X$ of $R$, choose exactly one view $V$ such that $X$ is in the head of $V$ in (the body of) $R$; drop all other views in the body of $R$. By construction, $R'$ has $n$ views in its body, where $n$ is the number of unique distinguished variables of $R$. We conclude the proof by noting that the number of unique distinguished variables of $R$ cannot exceed the arity of the head of $Q$. □

### 4.3 Algorithm for MiCR under the CWA

We define a *minimal MiCR* such that if a subgoal is deleted from the rewriting, then it is no longer a MiCR. For example, the rewriting $R_5$ in Example 5 is a MiCR of the query given the views, but is not a minimal MiCR, whereas $R_2$ is a minimal MiCR. It turns out that a MiCR is unique up to equivalence as expansions.

THEOREM 10. *Under the CWA and for queries and views satisfying conditions C1 and C2, MiCR is unique up to equivalence as expansions.*

We prove this result by contradiction: We assume that for some query $Q$ there exist two nonequivalent MiCRs, $R_1$ and $R_2$, and then show that there exists a third rewriting that (1) contains the query $Q$, and (2) is properly contained in one of the presumed MiCRs. Thus, by definition, neither of $R_1$ and $R_2$ is a MiCR of the query $Q$.

PROOF. By contradiction.

Let us assume that there are two MiCR's of a query $Q$, $R_1$ and $R_2$, and that $R_1^{exp}$ and $R_2^{exp}$ are not equivalent. Without loss of generality, we assume that the head of $Q$ has no repeated variables or constants. The two rewritings $R_1$ and $R_2$ contain $Q$, therefore, their heads map to the head of $Q$. Since the query has no variables or constants repeated in its head, the queries $R_1$ and $R_2$ must also have no variables or constants repeated in their heads, otherwise, the mapping from the head of the rewriting to the head of the query cannot be constructed.

We create a rewriting $R_3$ whose head is the same as that of one of the rewritings, say $R_1$, and whose body is the conjunction of the bodies of the two rewritings $R_1$ and $R_2$. We identify variables in $R_2^{exp}$ and $R_1^{exp}$ that map to the same query variable. Of these variables, we rename the variables in $R_2$ with the corresponding variable in $R_1$. Clearly, $R_3^{exp}$ is contained in $R_1^{exp}$ because $R_1$ appears in $R_3$. We show that $R_3$ does not contain $R_1$ but contains the query — a contradiction of the hypothesis that $R_1$ is a MiCR.

Since the body of $R_3$ contains subgoals from $R_1$ and $R_2$ both of which contained the query, it is easy to show that $R_3$ contains the query. $R_3$ contains the query $Q$ because:

- all subgoals in $R_3$ are from $R_1$ or $R_2$ and have predicates that appear in the query $Q$,
- $R_1$ appears unaltered in $R_3$ and $R_1$ contained $Q$, thus a partial mapping $\mu_1$ can be constructed from the head and the subgoals of $R_3^{exp}$ obtained from $R_1$ to $Q$, such that $AC(Q) \Rightarrow \mu_1(AC(R_1^{exp}))$
- the $R_2$-subgoals map the variables appearing in both $R_1$ and $R_2$ to the same query variables because by construction, they were equated,
- the variables occurring in $R_2$-subgoals but not in $R_1$ were retained from $R_2$ which had a mapping, say $\mu_2$, to $Q$ such that $AC(Q) \Rightarrow \mu_2(AC(R_2^{exp}))$.

Therefore, a mapping $\mu$ exists from $R_3$ to $Q$ and $R_3$ contains $Q$, where $\mu(X) = \mu_1(X)$ if $X$ appears in $R_3$ and $R_1$, and $\mu(X) = \mu_2(X)$ if $X$ appears in $R_3$ and $R_2$. Since the arithmetic comparisons in $R_1^{exp}$ were obtained from $R_1^{exp}$ and $R_2^{exp}$, using the implications above, it is easy to see that $AC(Q) \Rightarrow \mu(AC(R_2^e xp))$

Now, we just need to show that $R_3$ is properly contained in $R_1$; First,we show that $R_3^{exp}$ does not contain $R_1^{exp}$.

By our hypothesis, $R_2^{exp}$ does not contain $R_1^{exp}$. Therefore, there exists no containment mapping $\mu$ from $R_2^{exp}$ to $R_1^{exp}$ such that $AC(R_1^{exp} \Rightarrow \mu(AC(R_2^{exp}))$. If there is no containment mapping from $R_2^{exp}$ to $R_1^{exp}$, then one of the following cases must hold:

1. There exists a subgoal, $s$, in $R_2^{exp}$ with a different predicate than those in $R_1^{exp}$. Clearly, $s$ is in $R_3^{exp}$ and cannot be mapped to a subgoal in $R_1^{exp}$.
2. There exists a variable $X$ in $R_2^{exp}$ whose two occurrences was "mapped" to different variables or constants in $R_1^{exp}$. This variable (if renamed, the corresponding renamed variable) is in $R_3^{exp}$ and will prevent a containment mapping from $R_3^{exp}$ to $R_1^{exp}$.
3. There exists a constant in $R_2^{exp}$ that maps to a different constant or a variable in $R_1^{exp}$. The same constant prevents a containment mapping from $R_3^{exp}$ to $R_1^{exp}$.

If there is no containment implication, there must be an arithmetic comparison, $A_c$ in $\mu(AC(R_2^{exp}))$ that is not implied by $AC(R_1^{exp})$. Since all subgoals from $R_2$ are in $R_3$, $A_c$ creates a subgoal in $\mu(AC(R_3^{exp}))$ that is not implied by $AC(R_1^{exp}$. Therefore, $R_3$ does not contain $R_1$.

From (1) $R_3$ does not contain $R_1$ and (2) by construction, $R_3$ is contained in $R_1$, we conclude that $R_3$ is properly contained in $R_1$ because

□

We now describe an algorithm for finding conjunctive minimally containing rewritings of left-semi-interval, LSI (or right-semi-interval, RSI) queries and LSI (or RSI) views satisfying the non-union conditions $C1$ and $C2$. This restriction is sufficient because our algorithm only works when the homomorphism property holds [5]. It is also shown in [5] that in these cases the queries do not need to be normalized. When the homomorphism property does not hold, the number of containment mappings that are required to prove containment is more than one. In addition, the containing rewriting might not exist in the language of conjunctive queries; thus, a union of CQs might be necessary. To design an efficient and practical algorithm, we concentrate on the restricted cases mentioned above.

The algorithm first finds all views whose bodies contain some query subgoals, and then constructs buckets for query subgoals and the views. The conjunctive rewriting returned by the algorithm has one view subgoal from each bucket. Notice that each bucket in the algorithm represents a pair of subgoals, a query subgoal and a subgoal in the expansion of the view covering it. The details of the algorithm are found in the pseudocode in the Appendix and are clarified in the examples in the remainder of this section.

*Example* 6. Consider a query:

$q() \ :- \ b(A, A, A).$

Let the views be as follows:

$v_1() \ :- \ b(X, Y, Z).$
$v_2() \ :- \ b(X, X, Y).$
$v_3() \ :- \ b(X, Y, Y).$
$v_4() \ :- \ b(X, Y, X).$

The algorithm creates three buckets corresponding to the view subgoals $b(X, X, Y)$, $b(X, Y, Y)$, and $b(X, Y, X)$ that cover $b(A, A, A)$ but do not contain each other. Choosing the views from these buckets, we get an MiCR by conjoining them:

$r() \ :- \ v_2, \ v_3, \ v_4.$

The following example shows why we must consider multiple mappings from the view to the query.

*Example* 7. We use a relation `person` ($p$) with attributes `Name, Address, ZipCode,` and `Age`, and a relation `parent` ($r$) whose first attribute is the parent and the second attribute is the child.

Given a query:

$$q(C) : -p(P, A, Z, E), p(C, A_1, Z_1, E_1), r(P, C), E_1 < 21.$$

that returns the names of children whose age is less than 21. Suppose our access to the stored relations is via views $V_0$ through $V_2$:

$v_0() \quad : - \ p(N_0, A, Z, E).$
$v_1(N, E) \quad : - \ p(N, A, Z, E).$
$v_2(P, C) \quad : - \ r(P, C).$

First, the algorithm creates a bucket for the query subgoal $p(P, A, Z, E)$ and view subgoal $p(N_0, A, Z, E)$, and inserts $V_0$ in that bucket. Consider the mapping of $V_1$ to the first subgoal of $Q$. $V_1$ cannot be inserted in the bucket that contains $V_0$, because the bucket can only contain views whose subgoals are simple variable renamings of the subgoal $p(N_0, A, Z, E)$. The subgoal in the expansion of $V_1(P, E)$ is $p(N, A, Z, E)$ where $N$ is a distinguished variable, whereas the variable $N_0$ is nondistinguished. Therefore, the algorithm creates a new bucket for the query subgoal $p(P, A, Z, E)$ and the view subgoal $p(N, A, Z, E)$. In our algorithm, if the corresponding variable is a shared variable, we prefer to keep the bucket corresponding to the subgoal with the distinguished variable. This is another of the subtleties of our algorithm worth noting. Therefore, the bucket corresponding to the view subgoal $p(N_0, A, Z, E)$ is deleted and the new bucket corresponding to $p(P, A, Z, E)$ is retained.

Since there exists no bucket corresponding to the second query subgoal, a bucket is created corresponding to the query subgoal $p(C, A_1, Z_1, E_1)$ and the subgoal in the expansion of $V_1(C, E_1), E_1 < 21$, namely $p(C, A, Z, E_1)$. We insert the view $V_1(C, E_1), E_1 < 21$ into this bucket. Next, the algorithm considers the view $V_2(P, C)$. There is a mapping from the view subgoal to the query subgoal, and thus the view $V_2(P, C)$ is put into the bucket corresponding to the query subgoal $r(P, C)$ and view subgoal $r(P, C)$.

In the last step of the algorithm all three buckets contribute one subgoal each and their conjunction gives us the rewriting

$$R: \ Q(P) \ :- \ v_1(P, E), \ v_1(C, E_1), \ E_1 < 21, \ v_2(P, C).$$

Now, to show the effect of a view containing multiple query subgoals, we add a view

$$v_3(P, C) \ :- \ p(C, A, Z, E), \ r(P, C), \ E < 22.$$

As explained in the full version of the paper, the minimal MiCR in that case is

$$R: \ Q(P) \ :- \ v_1(P, E), \ v_3(P, C).$$

Note that this MiCR is minimal in the number of view subgoals used and is therefore selected by the set-cover subroutine at the last step of the algorithm.

Even though the algorithm tries to find one MiCR and we are operating under the closed-world assumption, in general we have to check all the view subgoals to find the tightest cover. For example, if some view $V_1$ covers a query subgoal $p(X, X, X)$ using a view subgoal whose expansion provides $p(X, Y, X)$, then the algorithm must check whether another view, say $V_2$, covers the query subgoal more tightly — that is, using a view subgoal whose expansion provides $p(X, X, X)$. If such a view $V_2$ exists, we can eliminate the bucket corresponding to the query subgoal $p(X, X, X)$ and view subgoal $p(X, Y, X)$ that has the view $V_1$. Once we have found the tightest possible cover for $p(X, X, X)$, the algorithm can be altered not to look for any other view subgoals, *provided* we do not seek the minimal MiCR. (If we seek the *minimal* MiCR, the algorithm must look at all views, because the last view might be covering all the subgoals in the query and might thus render all other views redundant.)

THEOREM 11. *If a query $Q$ and views $\mathcal{V}$ satisfy the non-union conditions C1 and C2, Algorithm A finds the MiCR of $Q$ using $\mathcal{V}$.*

PROOF. The algorithm finds a containing rewriting: (Sketch) Containing Rewriting: Each view is put in a bucket only if there exists a homomorphism $h$ and a containment mapping $\mu$ from $h(v_i)$ to a query $Q$ such that $AC(Q) \Rightarrow \mu_i(AC(h(v_i)^{exp}, ac))$. By construction, the head variables in the view are replaced by the query variables they map to. Thus, the containment mapping $\mu_i$ maps distinguished view variables to itself and non-distinguished view variables to a unique query variable or constant. A rewriting $R$ is obtained by conjoining a view from each bucket. A containment mapping $\mu$ constructed as $\mu(X) = \mu_i(X)$ if $X$ is a non-distinguished variable in view $v_i$ and $\mu(X) = X$ for all distinguished view variables. $\mu$ is a containment mapping because all $\mu_i$'s maps a distinguished view variable to itself and the non-distinguished view variables are not shared across views. Furthermore,

because $AC(Q) \Rightarrow \mu_i(AC(h(v_i)^{exp}, ac))$ for each view $v_i$, and because all AC's in $R^{exp}$ are obtained from the views appearing in it, $AC(Q) \Rightarrow \mu_i(AC(R^{exp}))$.

The containing rewriting found is minimal: To show that if there exists a containing rewriting $R'$ of $Q$ using $V$ such that the homomorphism property holds between $R'^{exp}$ and $Q$, then the algorithm A generates a containing rewriting $R$ such that $R'^{exp}$ contains $R^{exp}$.

For any rewriting $R$ constructed by the algorithm, the homomorphism property holds between $R^{exp}$ and $Q$. This fact is proved in the soundness proof above. Since the homomorphism property holds between $R'^{exp}$ and $Q$, there also exists a containment mapping $\mu$ from $R'exp$ to $Q$ such that $AC(Q) \Rightarrow AC(R'^{exp})$. For each view subgoal, $v_i$ in the body of $R'$, the algorithm considered all possible containment mappings from the body of $v_i$ to the body of $Q$. Therefore, the algorithm also considered each $\mu(v_i)$. Either each $v_i$ is put in one or more buckets corresponding to query subgoals, say $G$, it covers or is rejected. The algorithm must have generated $R$ in the second stage of the algorithm, because all buckets are guaranteed not to be empty. Note that all subgoals from $R'exp$ must map to query subgoals. For each query subgoal $g$, if $R'exp$ covers it with $g_i$ from the view $v_i$, $R^{exp}$ also covers it with a subgoal $g_j$ from $v_j$ because in the second stage of the algorithm we choose a view from each bucket to construct $R$. By construction, $g_j$ is contained in $g_i$ (the algorithm checks for and accepts the view whose expansion contains $g_i$ or the view was rejected because there was another view with a subgoal $g'$ such that $g$ properly contained $g'$). This containment implies a partial mapping from $v_i$ to $v_j$. The composition of these partial mappings is also a containment mapping because two occurrences of the same variable in $R'^{exp}$ map to the same query variable and by construction $R^{exp}$ has the same query variables. Similarly, it can be shown that $AC(R'^{exp}) \Rightarrow AC(R^{exp})$ because (1) the added ACs in $R$ are the same as those in the query, (2) if the ACs in $R'$ came from the views, the algorithm considered them and kept either those views or views with more containing ACs into the buckets. □

# 5 Approximate Query Answering and Rewriting

## 5.1 Approximate Query Answering

In the following example, there is neither a contained nor a containing rewriting of the query, however, we can still obtain some information about the answers to the query by accessing only the views.

*Example* 8. Consider a query $Q$ and two views:

$$\begin{array}{ll} q(Cust) & :- pennStCredUnion(Cust, Bal),\ Bal > 0. \\ v_1(Cust) & :- stanfCredUnion(Cust, Bal),\ Bal > 0. \\ v_2(Cust) & :- pennStCredUnion(Cust, Bal),\ Bal \geq 0, \\ & \quad age(Cust, 32). \end{array}$$

View $v_1$ does not contribute to the query but view $v_2$ can be used to obtain some answers for customers of age 32. The set of answers however is not a subset or a superset of the set of answers to the query because customers of age not equal to 32 are missing and customers with balance equal to 0 are added.

It might not be possible to obtain either a contained rewriting or a containing rewriting of a given query using given views. At the same time, the query may be answered approximately using the views.

We now provide algorithms to answer queries approximately using views.

*Algorithm 1:* An brute-force algorithm that generates the maximum number of true positives can be obtained as follows. From each view obtain all the data values and obtain the cross produce of all the values to generate tuples. Unfortunately, such a solution will contain a large number of false positives and the size of the result is exponential in the number of values, which may be quite large.

*Modified Algorithm 1:* This algorithm can be modified and made more efficient by the following adaptation. First, collect all data values belonging to a particular domain. This can be done by identifying view attributes that are of the same type using some heuristic. Next, for each variable in the query head, we identify its domain. Finally, we use the data values collected from the views to construct all possible tuples while restricting the value of a tuple attribute to values collected from view attributes with the same domain. This algorithm provides all possible tuples derivable from the views but does not guarantee that there will be any overlap between the rewriting and the query.

*Algorithm 2:* In the Modified Algorithm 1 each head variable is assigned values that are not guarnteed to occur in the same predicates as the head variables occurs in the body of the query. Therefore, in this algorithm we construct result tuples in the same manner as the modified algorithm 1 with the following adaptation. Let a head variable, $X$, appear in a set of relations $R$ in the body of the query. Let $Y$ be a distinguished view variable that appears in a set of relations $R'$. $X$ can be only can be bound to values of $Y$ such that there exists a relation $p$ in $R$ and $R'$ and $X$ and $Y$ appear in the same argument position in $p$.

In scenarios such as web applications, where it is often desirable to obtain a solution with fewer false positives, it is worthwhile to obtain an overlapping rewriting of the query and then evaluate it to obtain the answers to the query on demand.

## 5.2 Overlapping Rewritings

An overlapping rewriting is an approximate rewriting that provides more information about the content of the query. In our defintion in Section 2 an overlapping rewriting is defined such that it guarantees to produce an non-empty answer whenever there is a non-empty answer to the query.

We define below the concept of a maximally-overlapping rewriting that conforms to our intuition of a "best" possible solution with maximum overlap.

*Definition* 6. A *maximally-overlapping rewriting* (MOR) of a query $Q$ using views $V$ is an overlapping rewriting of a query that maximizes $(|Q(D) \cap R(V(D))|)/|R(D)|$ over all databases.

We refer to $|Q(D) \cap Q'(D)|$ as the size of the overlap between two queries $Q$ and $Q'$. The above definition seeks to minimize the ratio of the overlap (the number of tuples that are common between the results of the query and the rewriting) with the size of the rewriting. If we can estimate the size of the results of the query and the size of the results of the rewriting, then that information can be used to obtain a good rewriting (maximizing the overlap while minimizing the number of false positives). However, in a number of scenarios such statistics is not available. In these cases, we use the definition of maximally-overlapping rewriting provided in Section 2.

In several practical scenarios, including views of databases on the Internet, the views satisfy only the open-world assumption. The views are incomplete, that is, they are not guaranteed to contain all tuples that satisfy the definition of the views. In this scenario, a user wants a rewriting that contains all (or as many as possible) correct answers and as few false positives as possible. That is, the first criterion to optimize is to find a rewriting that produces as many answers to the query as possible and then for rewritings that produce the same answers to the query to produce as few false answers to the query as possible. We now give an algorithm that produces a rewriting that produces a *maximally overlapping rewriting* of a given query using views.

Like algorithm 2 above, this rewriting first identifies all possible views that can contribute values that can be bound to the head of the query and then takes the cartesian product of these views.

A closer look at the algorithm will illustrate that a large number of tuples are generated by this rewriting upon evaluation. The rewriting may contain a large amount of redundant values. Under the closed-world assumption, we can eliminate several redundant (contained) rewritings from the MOR. The development of this algorithm is left as future work.

## 6 Experiments

In this section we describe an implementation of the MiCR algorithm, see Section 4, and summarize our experimental results. The goal of the experiments was to study how the runtime of the algorithm depends on the number of given views and on the structure of the queries and views. We varied the structure of query and view definitions by changing the number of relational subgoals, the number of ACs, the shape of the queries/views (e.g., chain or star-shaped queries [6]), and the number of self-joins in the queries and views. Our preliminary results suggest good scalability of the algorithm in terms of both the number of views and of the structure of queries and views.

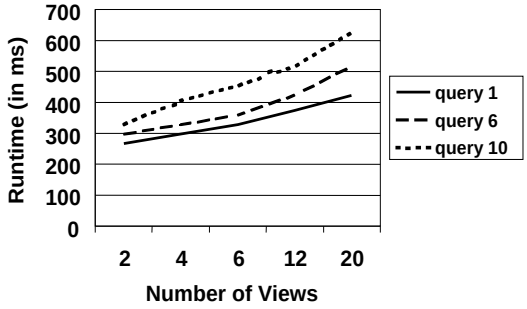


Figure 1: Scalability results.

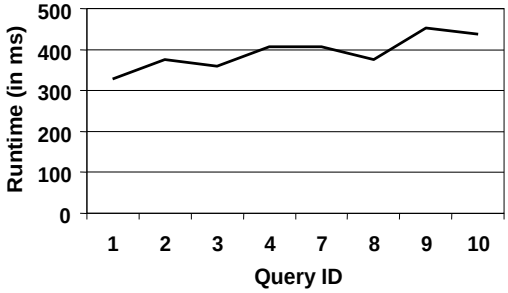


Figure 2: Dependence on query sizes.

Due to space constraints, we give here just two plots, in Figures 1 and 2, and report the total runtimes of the implementation, in milliseconds, in both Figures. (The full set of queries and views is given in Appendix B.) All experiments were run on a machine with a 1.69GHz Intel P4 processor, 1GB RAM, and an internal 40GB hard disk, running Windows XP Professional 2002. In the experiments reported in the plots we used queries and views that each have between one and five relational subgoals and between zero and five arithmetic-comparison subgoals. Figure 1 represents the results of the experiments where we studied the dependence of the runtime on the algorithm on the number of given views. We have also done experiments with the same three queries and a viewset with 100 views. In that experiment, the runtime of the section of the code that maps views into queries and allocating views to buckets was comparable to the runtime of the same section for the experiments reported in Figure 1. At the same time, the runtime of the minimum-set-cover subroutine of MiCR can become unacceptable when the buckets for query subgoals are populated with large numbers of views. Choosing appropriate approximation algorithms for minimum-set cover in MiCR is a direction of our current work.

In Figure 2 we report the results of our study of how the runtime of the algorithm depends on the structure of the queries. In the experiments, we used a single set of six views. We can see that the runtime of the algorithm for queries $Q_9$ and $Q_{10}$, which have the most complex definitions in these tests, is less than twice the runtime of the algorithm for query $Q_1$, which has the simplest definition.

**Algorithm 1:** MOR Algorithm

**Input** : CQ Q, Set of CQ Views V
**Output**: minimal OCR of Q using views V

```
begin
    for each view v in V: do
        for each subgoal g in v: do
            for each subgoal g' in Q with the same predicate as g do
                N ← Number of arguments in g
                for i = 1 to N: do
                    Y ← argument(g', i)
                    if Y is distinguished or exportable in Q then
                        X ← argument(g, i)
                        if X is distinguished in v then
                            Replace X with Y in v
                            Insert the modified v in bucket(Y)
                        else
                            if X is exportable in v then
                                Export X by equating it to a distinguished view variable Z using the least
                                restrictive head homomorphism on v
                                Replace Z with Y in v
                                Insert the modified v in bucket(Y)
    Construct all possible rewritings with the same head as Q and the body of the rewriting is the
    conjunction of views such that the body contains a view from each bucket
    Add the ACs on the distinguished query variables in Q to the rewriting
    Output the union of all the rewritings
end
```

## Acknowledgments

We are grateful to Dimitri Rakitine and Shalu Gupta for the help with the implementation and experiments.

## References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proceedings of PODS*, pages 254–263, 1998.

[2] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *Proceedings of SIGMOD*, pages 574–576, 1999.

[3] F. Afrati, R. Chirkova, and P. Mitra. Accessing data using views. Technical Report TR-2005-14, Computer Science, NC State University, 2005. Available from `http://www4.ncsu.edu/~rychirko/Papers/techReport022805.pdf`.

[4] F. Afrati, C. Li, and P. Mitra. Answering queries using views with arithmetic comparisons. In *Proceedings of PODS*, 2002.

[5] F. Afrati, C. Li, and P. Mitra. On containment of conjunctive queries with arithmetic comparisons. In *Proceedings of EDBT*, 2004.

[6] F. Afrati, C. Li, and J. Ullman. Generating efficient plans for queries using views. In *Proceedings of ACM SIGMOD*, 2001.

[7] F. N. Afrati, M. Gergatsoulis, and T. G. Kavalieros. Answering queries using materialized views with disjunctions. In *Proceedings of ICDT*, pages 435–452, 1999.

[8] Amazon.com. `http://www.amazon.com`.

[9] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *Proceedings of SIGMOD*, pages 539–550, 2003.

[10] E. Bertino, E. Ferrari, and A. Squicciarini. Trust negotiations: Concepts, systems, and languages. *IEEE Computing in Science and Engineering*, 6(4), 2004.

[11] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *Proceedings of VLDB*, pages 111–122, 2000.

[12] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 77–90, 1977.

[13] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of ICDE*, pages 190–200, 1995.

[14] 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 *IPSJ*, pages 7–18, 1994.

[15] A. Deutsch, B. Ludaescher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. In *Proceedings of ICDT*, 2005.

[16] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proceedings of PODS*, pages 109–116, 1997.

[17] D. Florescu, A. Y. Levy, D. Suciu, and K. Yagoub. Optimization of run-time management of data intensive websites. In *Proceedings of VLDB*, pages 627–638, 1999.

[18] G. Grahne and A. O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proceedings of ICDT*, pages 332–347, 1999.

[19] A. Gupta, Y. Sagiv, J. Ullman, and J. Widom. Constraint checking with partial information. In *Proceedings of PODS*, pages 45–55, 1994.

[20] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proceedings of VLDB*, pages 276–285, 1997.

[21] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(3), 2001.

[22] Z. G. Ives, D. Florescu, M. Friedman, A. Y. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *Proceedings of ACM SIGMOD*, pages 299–310, 1999.

[23] R. J. B. Jr., W. Bohrer, R. S. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. H. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Semantic integration of information in open and dynamic environments (experience paper). In *Proceedings of ACM SIGMOD*, pages 195–206, 1997.

[24] A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, 35(1):146–160, 1988.

[25] A. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of PODS*, pages 95–104, 1995.

[26] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of VLDB*, pages 251–262, 1996.

[27] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *Proceedings of SIGMOD*, pages 575–586, 2004.

[28] P. Mitra. An algorithm for answering queries efficiently using views. In *Proceedings of the Australasian Database Conference*, 2001.

[29] V. Poosala, V. Ganti, and Y. E. Ioannidis. Approximate query answering using histograms. *IEEE Data Engineering Bulletin*, 22(4):5–14, 1999.

[30] R. Pottinger and A. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB Journal*, 2001.

[31] X. Qian. Query folding. In *Proceedings of ICDE*, pages 48–55, 1996.

[32] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of SIGMOD*, pages 551–562, 2004.

[33] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information. In *Proceedings of PODS*, page 188, 1998.

[34] J. D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

[35] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *Proceedings of PODS*, pages 331–345, 1992.

[36] J. Widom. Research problems in data warehousing. In *CIKM*, pages 25–30, 1995.

[37] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata. Answering complex sql queries using automatic summary tables. In *SIGMOD Conference*, pages 105–116, 2000.

## A Appendix

The pseudocode of the algorithm for MiCR is given below.

## B Queries and Views in the Experiments

The ten queries are as follows:

$q_1() :- p(X,Y), X < Y.$
$q_2() :- p(X,Y), s(Y,Z), X < Y, Z \leq 3.$
$q_3() :- p(X,Y), s(Y,Z), t(Z,W), X < Y.$
$q_4() :- p(X,Y), s(Y,Z), t(Z,W), X < Y,$

```
Input  : CQAC Q, Set of CQAC Views V
Output : minimal MiCR of Q using views V
begin
    for each v, v' ∈ V do
        if v and v' are equivalent then
            Keep only one of v,v'
    for each view v in V do
        for each containment mapping μ_i from the core subgoals in the body of v to Q do
            Construct h(v) by replacing each distinguished variable in v with μ_i(V)
            ac ← null
            for each ac_i ∈ AC(Q) do
                if all variables in ac_i appear in h(v) then
                    ac ← ac ∧ ac_i
            if AC(Q) ⇒ μ_i(AC(h(v), ac)) then
                for each subgoal g in Q that h(v) covers do
                    Let μ_i map g_i in exp(h(v)) to g
                    create_bucket ← true
                    for each Bucket(g_j) covering g do
                        if g_j properly contains g_i then
                            { g_j is redundant }
                            delete Bucket(g_j)
                        else
                            if g_i properly contains g_j then
                                { g_i is redundant, do not need to do anything }
                                create_bucket ← false
                                break
                            else
                                if g_i and g_j are equivalent then
                                    { g_i and g_j are equivalent, no need to create a new bucket }
                                    create_bucket ← false
                                    Insert h(v), ac into Bucket(g_i)
                    if create_bucket == true then
                        create Bucket(g_i)
                        insert h(v), ac into Bucket(g_i)
    { Now, we have a set of buckets, and a set of view subgoals that cover each bucket. }
    Run a minimum set cover algorithm to select a set of view subgoals such that all buckets are covered.
    { Constructing one rewriting is enough because it is an MiCR. }
    Construct a rewriting by taking the conjunction of the selected views and their associated arithmetic
    predicates.
end
```

$Z \leq 3, W < 5.$

$q_5() :- p(X,Y), p(Y,Z), Z \leq 3.$

$q_6() :- p(X,Y), s(Y,Z), s(Z,W), X < Y.$

$q_7() :- p(X,Y), s(Y,Z), t(Z,W), r(Y,A), X < Y, Z \leq 3.$

$q_8() :- p(X,Y), s(Y,Z), s(Z,W), r(Y,A), X < Y, Z \leq 3.$

$q_9() :- p(X,Y), s(Y,Z), s(Z,W), p(Y,A), p(Y,B), X < Y, Z \leq 3.$

$q_{10}() :- p(X,Y), s(Y,Z), X < Y, Z \leq 3, Y < Z, Y < 8.$

The twenty views are as follows:

$v_1(X,Y) :- p(X,Y), X \leq Y.$

$v_2(X) :- p(X,Y).$

$v_3(Y) :- p(X,Y), X \leq Y.$

$v_4(X,Y,Z) :- p(X,Y), s(Y,Z), X \geq Y.$

$v_5(X,Z) :- p(X,Y), s(Y,Z), X \leq Y, Y \leq Z.$

$v_6(X,Y,W) :- p(X,Y), s(Y,Z), t(Z,W), Z \leq 2.$

$v_7(X,Y,W) :- p(X,Y), s(Y,Z), t(Z,W), Z \leq 8.$

$v_8(Y,Z) :- s(Y,Z), t(Z,W), Z \leq 4.$

$v_9(X,Z) :- p(X,Y), p(Y,Z), X \leq Y, Y \leq Z.$

$v_{10}(X,Z) :- p(X,Y), p(Y,Z), X \leq Y.$

$v_{11}(X,Y) :- p(X,Y), r(Y,A).$

$v_{12}(Y,Z) :- p(X,Y), s(Y,Z), X \leq Y, Z \leq 5.$

$v_{13}(Y,Z) :- p(Y,A), p(Y,B), s(Y,Z).$

$v_{14}(X,Y) :- p(X,Y), s(Y,Z), s(Z,W), Z \leq 4.$

$v_{15}(X) :- p(X,Y), p(Y,A), X \leq Y.$

$v_{16}(X,Y) :- p(X,Y), p(Y,A), X \leq Y.$

$v_{17}() :- p(X,Y), s(Y,Z), Z \leq 2, Y \leq Z.$

$v_{18}(Y) :- s(Y,Z), s(Z,W), r(Y,A), Z \leq 3.$

$v_{19}() :- p(X,Y), s(Y,Z), s(Z,W), p(Y,A), p(Y,B), X \leq Y, Z \leq 6.$

$v_{20}() :- p(X,Y), s(Y,Z), X \leq Y, Z \leq 3, Y \leq Z, X \leq Z, Y \leq 9.$

Note 1: Views $V_4$, $V_6$, $V_9$, $V_{17}$ should not match any of the queries $Q_1$ through $Q_{10}$. Note 2: View $V_{20}$ should match query $Q_{10}$.

Experiments:

A. $Q_1$ with $\{V_1, V_2, V_5, V_7, V_{10}, V_{12}\}$
B. $Q_2$ with $\{V_1, V_2, V_5, V_7, V_{10}, V_{12}\}$
C. $Q_4$ with $\{V_1, V_2, V_5, V_7, V_{10}, V_{12}\}$
D. $Q_7$ with $\{V_1, V_2, V_5, V_7, V_{10}, V_{12}\}$
E. $Q_9$ with $\{V_1, V_2, V_5, V_7, V_{10}, V_{12}\}$
F. $Q_1$ with $\{V_1, V_5\}$
G. $Q_6$ with $\{V_1, V_5\}$
H. $Q_{10}$ with $\{V_1, V_5\}$
I. $Q_1$ with $\{V_1, V_5, V_2, V_7\}$
J. $Q_6$ with $\{V_1, V_5, V_2, V_7\}$
K. $Q_{10}$ with $\{V_1, V_5, V_2, V_7\}$
L. $Q_1$ with $\{V_1, V_5, V_2, V_7, V_3, V_8\}$
M. $Q_6$ with $\{V_1, V_5, V_2, V_7, V_3, V_8\}$
N. $Q_{10}$ with $\{V_1, V_5, V_2, V_7, V_3, V_8\}$
P. $Q_1$ with all the 20 views
R. $Q_6$ with all the 20 views
S. $Q_{10}$ with all the 20 views
T. $Q_1$ with 100 views
U. $Q_6$ with 100 views
V. $Q_{10}$ with 100 views
W. $Q_1$ with $\{V_1, V_5, V_2, V_7, V_3, V_8, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{18}\}$
X. $Q_6$ with $\{V_1, V_5, V_2, V_7, V_3, V_8, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{18}\}$
Y. $Q_{10}$ with $\{V_1, V_5, V_2, V_7, V_3, V_8, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{18}\}$
Z. $Q_3$ with $\{V_1, V_2, V_5, V_7, V_{10}, V_{12}\}$
AA. $Q_8$ with $\{V_1, V_2, V_5, V_7, V_{10}, V_{12}\}$
AB. $Q_{10}$ with $\{V_1, V_2, V_5, V_7, V_{10}, V_{12}\}$

Note 3: "100 views" means five copies of each of the views $V_1$ through $V_{20}$, renamed for head uniqueness. Note 4: Figure 1 in Section 6 shows the results of experiments F through S and W through Y. Note 5: Figure 2 in Section 6 shows the results of experiments A through E and Z, AA, AB.