Information about Queries Obtained by a Set of Views

Foto Afrati
Nat'l Technical Univ. Athens, Athens, Greece
afrati@softlab.ntua.gr

Manik Chandrachud
NC State University, Raleigh, NC 27695 USA
mnchandr@ncsu.edu

Rada Chirkova
NC State University, Raleigh, NC 27695 USA
chirkova@csc.ncsu.edu

Prasenjit Mitra
The Penn. State Univ., University Park, PA, USA
pmitra@ist.psu.edu

ABSTRACT

Significant research has been done on examining the problems of generating equivalent rewritings (ER) or maximally-contained rewritings (MCR) of queries. At the same time, when ERs and MCRs do not exist, users can still use views to obtain meaningful information on their queries. Emerging applications require a study of the problem of rewriting queries using views in a variety of new scenarios. For example, in web-search applications when MCRs may not exist, users may want to obtain a rewriting that provides all query answers (but may contain false positives). Even when an MCR is not available, security requirements of other applications may require checking whether *any* rewriting exists for a query using a set of views expressed in the fixed query language in which users can pose queries.

In this paper we study *contained* and *containing* rewritings of a query using a set of views; the rewritings give a subset and a superset, respectively, of the query answer. We consider queries and views in the language of conjunctive queries with arithmetic comparisons (CQAC queries), and rewritings in the language of unions of CQAC queries. To the best of our knowledge, no algorithm is known for checking for the existence of MCRs for many interesting cases of CQAC queries and views. In those cases, our results can be used to compute some answers to a query using views — depending on the application — even if some other answers are missing or if false positives are obtained.

We refer to a containing rewriting that contains no false negatives and with the minimal number of false positives (when using a given set of views) as a "minimally containing rewriting" (MiCR). While the running time of our algorithm for finding MiCRs is worst-case exponential in the size of the problem inputs, the algorithm performs well in many practical cases, due to its extensive pruning of the search space. Our experiments show good scalability of our algorithm.

1. INTRODUCTION

Rewriting queries using views and then executing the rewritings to answer the queries is an important technique used in data warehousing, information integration, query optimization, and other applications [8, 20, 22, 26, 36, 35, 12]. Previous research has focused primarily on obtaining equivalent rewritings (ERs) that can be used to derive all answers (see, e.g., [25, 1, 3] and references therein) or maximally contained rewritings (MCRs) that can be used to derive a maximal subset of the set of query answers using given views (see, e.g., [21, 37, 29, 32, 5] and references therein). Various recent applications have put forth the problem of going beyond ERs or MCRs for view-based rewritings. For instance, when equivalent or maximally contained rewritings of a user query in terms of the available views do not exist, then traditional query-processing engines will return no answers at all to the query. At the same time, in many applications, such as querying the web or mass marketing using data warehouses, users prefer to get a superset of the answers to the query, rather than no answers at all. In particular, in mass marketing getting no answers to queries means that users may lose customers. On the other hand, in peer-to-peer networks queries are rewritten as they are routed along a path, and the best option is to use the incomplete information available from neighboring peers to obtain an approximate rewriting. In such cases a seriously restricted subset — in addition to a superset — of answers to the query is also acceptable, as they can act as upper and lower bounds [17] on the actual set of answers. When security issues exist, then we want to know whether users who have access to a set of views can obtain answers to a secure query by using the query language available to them to pose queries on the view schema. If this is not the case then the set of views may be considered secure [15].

The following is a motivating example for types of rewritings that we consider in this paper.

EXAMPLE 1.1. *Consider a user query* `Q` *that asks for phone numbers of people in Raleigh, NC, such that the household income of those people is at least $70K.*

```
Q: SELECT ResidencePhone FROM Person
   WHERE City = 'Raleigh' AND State = 'NC'
   AND HouseholdIncome >= 70000;
```

Suppose we have access only to view `V1`*, which returns phone numbers of people with income greater than $80000. Clearly, this view discloses some information about query* `Q`*, since each answer to* `V1` *is also an answer to* `Q`*. Consider another view,* `V2`*, which lists only the incomes (rather than the names, phone numbers, or other identifying information) of people in Raleigh. If* `Q` *is a sensitive query, view* `V2` *can be considered a "secure" view with respect to* `Q` *because the set of answers to* `V2` *is disjoint from the set of answers to the query. If users can use a more expressive language than CQAC then view* `V2` *may not be secure any more. However,*

it is reasonable to assume that for many users the only access to the information is via a given query language [15].

Now suppose we have access to view `V3`*, which returns the phone numbers of all people in Raleigh, NC. Mass-marketers or analysts looking for terrorism suspects would accept tuples that are not among the answers to the query (false positives), rather than lose some answers or get no answers at all. Hence, they will be satisfied if they obtain the answers to* `V3`*, rather than the answers to* `V1`*, because the answers to* `V3` *contain the answers the users are looking for. (E.g., for mass marketing losing customers is not an option.) However, if the users had available a view* `V4` *that returns phone numbers of people in Raleigh with income over $60,000, then the users would prefer to get the information from* `V4`*, rather than from* `V3`*, to minimize the number of false positives. In this case, view* `V4` *is a minimally-containing rewriting (MiCR) of the query using the view* `V4`*. In another scenario, e.g., in peer-to-peer systems, and in the absence of an MCR the answers obtained by the view* `V4` *(containing rewriting) together with the answers obtained by the view* `V1` *(contained rewriting) give a better idea of the real set of answers to the query than the answers to either* `V4` *or* `V1` *alone.* □

We study *contained rewritings* and *containing rewritings* of a query using a set of views in the presence of arithmetic comparisons, since in many practical applications arithmetic comparisons have to be used. We consider queries and views in the language of CQACs and rewritings in the language of unions of CQACs. In presence of arithmetic comparisons the problems of finding equivalent rewritings and MCRs are recognized to be significantly more complex, and many cases have remained unexplored [37, 5].

We explore the following two classes of the problem of obtaining approximate query answers:

1. For contained rewritings, we study the decision problem of whether non-trivial rewritings exist, and develop an algorithm for finding such rewritings. Clearly, if a maximally contained rewriting can be found then the above questions are answered. However, the approaches in the literature that address the problem of maximally contained rewritings [29, 32] consider primarily CQ queries without ACs, with very little work on more general cases (see, e.g., [5]). Thus the problem of finding MCRs in the presence of arithmetic comparisons remains open in the general case. The complexity of the problem is mainly due to the more complex containment test in the presence of arithmetic comparisons — it is Π_2^P-complete [37, 38].
2. For containing rewritings, we study the decision problem of whether such rewritings exist. Ideally, the set of answers returned by a rewriting should be minimal, that is, should contain as few false positives as possible. We call such a rewriting the *minimally containing rewriting* (MiCR) of a query using views [13]. Intuitively, a MiCR is the analog of MCR, as it finds all the answers to the query that are in all containing rewritings. In this sense, MiCRs give the highest guarantee on the quality of the answers that can be obtained, by minimizing the number of false positives in the answers. We give a sound algorithm that finds MiCRs. The algorithm is complete in the special case where the "homomorphism property"[1] holds between the expansions of the rewritings and the query.

Our contributions in detail are as follows:

1. We investigate the existence-of-a-rewriting problem for both contained and containing rewritings. For special cases, we give decidability and complexity results for the existence of nontrivial contained rewritings.[2] Although decidability of the general problem remains open, our results cover a much larger class of queries and views than the class for which decidability is known about MCRs in the literature. For containing rewritings, we prove that the problem of existence of such a rewriting for a given query and set of views is decidable (in the general case), thus also showing that this is an easier problem than the existence-of-a-contained-rewriting problem, where decidability of the general case is still open. We then investigate special (still intractable) lower-complexity cases of the problem.
2. We develop an efficient heuristic algorithm to find a nontrivial contained rewriting whenever there exists one. The algorithm is sound and complete (a) when the query uses only semi-interval arithmetic comparisons (i.e., each comparison compares a variable to a constant), and (b) when the views have no non-distinguished variables in their definition (the case of *full views*). In the general case, the algorithm is still sound and complete whenever it halts, but it is not guaranteed to halt.
3. We develop an algorithm for computing a MiCR. It is efficient, in that we prune the search space significantly by testing views for usability and rejecting early views that are not useful. The algorithm is sound, and is also complete when the homomorphism property holds [6, 23]. (When the homomorphism property does not hold, multiple mappings may be needed to prove the containment.)

Table 1 gives a summary of our results and contributions.

Related Work

The problem of using views in query answering [25] is relevant in applications in information integration [37], data warehousing [21], web-site design [16], and query optimization [12, 25, 39]. Algorithms for finding rewritings of queries using views include the bucket algorithm [18, 26], the inverse-rule algorithm [4, 14, 33], the MiniCon algorithm [31], and the shared-variable-bucket algorithm [29]; see [21] for a survey. Almost all of the above work focuses on investigating MCRs or ERs [37, 1], as it takes its motivation mostly from information integration and query optimization. Query-rewriting algorithms depend upon efficient algorithms for checking query containment. It is known from existing work on query containment [19, 23, 38] that adding arithmetic comparisons to queries and views makes these problems significantly more challenging.

Since we consider rewritings that may return false positives, false negatives, or both, our work has similarities with approximate answering of queries using views (see [2, 7, 10, 30] and references therein). Approximate query answering

[1]Intuitively, the *homomorphism property* is said to hold between a query and its rewriting when a single mapping from the query to the expansion of the rewriting is sufficient to establish the containment of the rewriting in the query.

[2]The *trivial query* that returns no tuples on all databases is contained in all queries.

	Contained Rewritings	Containing Rewritings
Decidability	SI-CQAC or views no non-distinguished variables	CQAC homomorphism property
Complexity	CQ: NP [25]	CQAC homomorphism property: NP
Algorithms	Finds nontrivial CR	Finds MiCR
Previous Work	MCR [21, 5]	MiCR [13]
Applications	Data warehousing, security, privacy	Mass marketing, P2P, information retrieval

Table 1: Our contributions, previous work, and applications. The algorithms are sound and complete for the decidable cases for contained rewritings and when the homomorphism property holds for MiCR.

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.[3] Lee et al [24] have considered non-equivalent query rewritings, applied to the problem of maintaining view definitions using a quantitative estimation of the quality of the relaxed query and enabling a tradeoff between performance and the quality of answers. Rather than focusing on performance, our work considers the problem when no rewritings are possible even when computational or storage resources are not constrained.

The problem of finding containing rewritings of queries using views was introduced by Deutsch et al. [13]. In [13] 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, which render the problem intractable in the general case. The language of rewritings considered in [13] is union of conjunctive queries with negation. Our work focuses on finding rewritings in the language of conjunctive queries with arithmetic comparisons (i.e., without negation). In the absence of negation in the language for rewritings, existence of rewritings is an issue that we address in our paper. Also, in view of the intractability results, we identify special cases where a more efficient algorithm exists for constructing rewritings.

Other related work includes the results of Rizvi et al. [34], where query-rewriting techniques are used for fine-grained access control, and the work of Miklau et al. [28], which contains a formal probabilistic analysis of information disclosure in data exchange under the assumption of independence among the relations and data in a database. Related work in security and privacy includes [27]. Calvanese et al. [9] have discussed query answering, rewriting and losslessness with respect to two-way regular path queries. In our work, we concentrate only on query rewritings.

2. PRELIMINARIES

2.1 Queries, Containment, and Views

We consider *conjunctive queries with arithmetic comparisons* (CQAC for short), i.e., select-project-join SQL queries with equality and comparison selection conditions. Each *arithmetic comparison* (AC) subgoal C_i is of the form $X\theta Y$ or $X\theta c$,[4] where the comparison operator θ 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. In the rest of the paper, for a query Q we denote the conjunction of all relational subgoals in Q as Q_0 and the conjunction of all arithmetic comparisons in Q as β. We will use the term *semi-interval CQAC (SI-CQAC)* to refer to conjunctive queries with arithmetic comparisons, where all comparisons in the query are either one of $X < c$, $X \leq c$ (*left semi-interval*) or one of $X > c$, $X \geq c$ (*right semi-interval*).

DEFINITION 2.1. *(***Query containment***) A query Q_1 is* contained *in a query Q_2, denoted $Q_1 \sqsubseteq Q_2$, if and only 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 [11] have shown 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 test for CQACs is more complicated. There are two ways to test for containment [19, 23]; we will describe them very briefly, for more details see, e.g., [6]. The first test for CQACs uses the notion of a *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 Q_1. The test says that a query Q_1 is contained in query Q_2 if and only if Q_2 computes the same head tuple as Q_1 on all the canonical databases of Q_1.

The second containment test is as follows:

THEOREM 2.1. *$Q_1 \sqsubseteq Q_2$ if and only if the following logical implication ϕ is true:*

$$\phi : \beta_1' \Rightarrow \mu_1(\beta_2') \vee \ldots \vee \mu_k(\beta_2')$$

where μ_i's are all containment mappings from Q_2' to Q_1' and β_i' is a conjunction of all arithmetic comparisons in Q_i'. That is, the comparisons in the normalized query[5] Q_1' logically imply (denoted "$\Rightarrow$") the disjunction of the images of the comparisons of the normalized query Q_2' under each mapping μ_i. □

If there exists a containment mapping μ_i such that the right-hand side of ϕ is reduced to only one $\mu_i(\beta_2')$, we say the *homomorphism property* holds. It has been shown [6] that when the homomorphism property holds, the implication can be checked directly on queries that are not normalized. Checking CQAC containment is less complex in that case, because one just needs to check for the existence of one mapping that satisfies the implication. In Section 4 we outline an algorithm for finding minimally containing rewritings for cases where the homomorphism property holds.

[3] In our work, we do not measure approximations using probabilities or uncertainty, but the answers to the queries are approximate in the sense that the derived answers may contain false positives.

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

[5] 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.

2.2 Rewriting Queries using Views

We consider the problem under the closed-world assumption [1] (where the views are both sound and complete, i.e., for a given database, each view instance stores exactly the tuples satisfying the view definition), as well as under the open-world assumption [1, 26] (where the views are sound, 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 are considering contained rewritings both under the open-world assumption (OWA) and under the closed-world assumption (CWA).

We consider the following types of rewritings R of a query Q using views $\mathcal{V}$:

DEFINITION 2.2. *(***Rewritings***)*

1. *a. (CWA) R is a* contained rewriting *of Q using $\mathcal{V}$ under the CWA if and only if $R(D_\mathcal{V}) \subseteq Q(D)$ for all databases D.*
   b. (OWA) R is a contained rewriting *of Q using $\mathcal{V}$ under the OWA if and only if $R(I_V) \subseteq Q(D)$ for all databases D and view instances I_V such that $I_V \subseteq D_\mathcal{V}$.*
2. *(CWA) R is a* containing rewriting *of Q using $\mathcal{V}$ if and only if $Q(D) \subseteq R(D_\mathcal{V})$ for all D.*
3. *(CWA) R is an* equivalent rewriting *of Q using $\mathcal{V}$ if and only if $Q(D) = R(D_\mathcal{V})$ for all D.* □

Given a query Q and a set of views $\mathcal{V}$, whether a contained or containing rewriting of Q using $\mathcal{V}$ exists depends on the language of the rewriting. We consider the problem of existence of such rewritings in the language of union of CQACs. In the rest of the paper, we will assume that this is the language of the rewritings unless otherwise stated.

PROPOSITION 2.1. *For queries and views that are conjunctive queries with arithmetic comparisons (CQAC), a union of CQAC rewriting is a contained rewriting under the open-world assumption (OWA) iff it is a contained rewriting under the closed-world assumption (CWA).* □

Thus, hereafter, we refer to contained rewritings under either the CWA or OWA.

We define the expansion of a rewriting as follows:

DEFINITION 2.3. *(***Expansion of rewriting***) 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 all the subgoals in the view definition. Existentially quantified variables in the definitions of the views 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.* □

Evaluation of contained rewritings cannot return false positives, containing rewritings cannot return false negatives, and equivalent rewritings cannot return either false positives or false negatives. We will use the term *rewriting* to mean contained or containing rewriting of a given query; we will specify the rewriting type whenever it is not obvious from the context.

Theorem 2.2 gives tests for whether a CQAC rewriting R in terms of views $\mathcal{V}$ is contained or containing with respect to a CQAC query Q.

THEOREM 2.2. *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. CONTAINED REWRITINGS

When queries contain arithmetic comparisons, it is not easy to find an MCR of a query using views [6]; for certain languages such rewritings do not even exist. For instance, for certain cases of conjunctive queries with semi-interval arithmetic comparisons, we cannot find an MCR in the language of unions of CQAC views, but we can find an MCR in recursive datalog with ACs [5]. In addition, an MCR does not exist in certain cases of CQACs for languages that are polynomially computable [1]. At the same time, easy subcases are known; for instance, if containment can be checked using a single containment mapping (the homomorphism property), then we can construct an MCR for a CQAC query and CQAC views [5, 6]. However, for the general case there is no known algorithm for finding MCRs for CQAC queries and views [5].

In this work we study finding contained rewritings for cases where we do not know how to find an MCR. Because any query that returns an empty answer 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 Existence*): Given a CQAC Q, a set of CQAC views $\mathcal{V}$, and a language $\mathcal{L}$, is there a nontrivial contained rewriting of Q using $\mathcal{V}$ in $\mathcal{L}$? (When the views and query are CQs, then we have the *CQ Contained Rewriting Existence* problem.)

The language of rewritings we consider in this paper (unless otherwise stated) is union of conjunctive queries with arithmetic comparisons.

3.1 Decidabillity and Complexity

Not surprisingly, deciding the existence of a nontrivial contained rewriting turns out to be easier than computing an MCR. In this subsection, we present decidability and complexity results for contained rewritings.

First we look at the CQ Contained Rewriting Existence Problem.

THEOREM 3.1. *Let both the query and views be conjunctive (without comparisons). Then the following hold:*

1. *There is a contained rewriting in the language of CQACs iff there is a contained rewriting in the language of conjunctive queries.*
2. *If, in addition, the views do not have distinguished variables then we can check in PTIME whether there exists a contained rewriting.* □

PROOF. To prove item 1, let R be a CQACs contained rewriting. Let R' be a conjunctive rewriting which results from dropping the comparison subgoals of R. We will prove that R' is a contained rewriting. According to the containment test based on canonical databases, the following holds: If we take the expansion of R and consider the canonical database of the expansion where all variables (which are not explicitly equated after taking the closure of the ACs) are frozen to distinct constants, then there is a homomorphism h from the query subgoals to the expansion subgoals.

Homomorphism h is also a homomorphism from the query subgoals to the expansion of R', hence R' is a contained rewriting according to the containment test for CQs.

To prove item 2, observe that if all predicates of the query appear in the views, then there is a rewriting that consists of all views with head homomorphisms identifying all variables is a contained rewriting. Otherwise there is no rewriting. □

Now we prove NP-completeness for the CQ case (actually membership in NP is known [25]:

THEOREM 3.2. *Let Q be a CQ query, and let $\mathcal{V}$ be a set of CQ views. It is NP-complete to decide whether there exists a nontrivial contained rewriting in the language of union of CQs of Q using $\mathcal{V}$.* □

PROOF. To prove NP hardness, we reduce the CQ containment problem, which is NP complete, to the problem of existence of a contained rewriting in the language of union of CQACs of a CQAC query using CQAC views. Let Q_1, Q_2 be CQs; we ask whether Q_2 is contained in Q_1. We construct a boolean query Q that includes all the subgoals of Q_1. Additionally, for each head variable X in Q_1, we add a subgoal $r_X(X)$ to Q, where r_X is a unique unary predicate distinct from any other predicate appearing in Q_1 or Q_2. We construct a boolean view V from Q_2 that includes all subgoals of Q_2 together with an additional set of subgoals over unary predicates as follows: We consider a mapping h from the head variables of Q_1 to the head variables of Q_2, which maps variable in argument position i in Q_1 to variable in argument position i in Q_2. (If there is no such mapping then Q_2 is not contained in Q_1.) For each distinguished variable Y in Q_2 such that $Y = h(X)$, we add the unary subgoal $r_X(h(X))$ to the view V. It is easy to prove that Q_2 is contained in Q_1 iff there is a contained rewriting of Q using V.

If: Let Q_2 be contained in Q_1. Therefore, there exists a mapping μ from Q_1 to Q_2. The mapping μ maps the subgoals in Q (inherited from Q_1) to the subgoals in V (inherited from Q_2. The mapping μ also maps the i^{th} variable in the head of Q_1 to the i^{th} variable in the head of Q_2; therefore for all X for which h is defined, $h(X) = \mu(X)$. By construction, μ maps the additional unary predicates introduced in the query Q to the unary predicates $p_X(h(X))$ introduced in the view V. Because a containment mapping exists from the query to the body of the view, the view V is a contained rewriting of the query Q in the language of CQAC (a CQ is also a CQAC with no ACs).

Only If: Let there be a rewriting of Q in the language of *CQAC*s. Because V is the only view and V is a boolean view, we can eliminate muliple occurrences of V in the rewriting, keeping only one copy of V. The minimized rewriting is still a contained rewriting (CR) of Q. Furthermore, the CR has no ACs, because the view V does not have any head variables on which the ACs can be defined. All the subgoals of Q_1 are present in Q. Because the additional subgoals in V have unique unary predicates, the subgoals in Q inherited from Q_1 map to subgoals in V inherited from Q_2. Thus, there is a mapping from the body of Q_1 to that of Q_2. By construction, the head of Q_1 also has a mapping to the head of Q_2. Therefore, $Q_1 \sqsubseteq Q_2$.

For membership in NP we use Theorem 3.1: It is known from the literature [25] that for the CQ case, the problem of checking whether there exists a contained rewriting for the case of conjunctive query and views is in NP. □

Now we investigate the CQAC Contained Rewriting Existence Problem. Afrati et al. [5] have proved the following:

THEOREM 3.3. *Let Q be a CQAC query, and let $\mathcal{V}$ be a set of CQAC views that do not have any 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 n subgoals, where n is the number of relational subgoals in Q. Thus there is an exponential algorithm which finds a non-trivial contained rewriting if such a rewriting exists. The respective decision problem is in NP.* □

We prove in the following theorem that the problem in Theorem 3.3 is NP hard.

THEOREM 3.4. *Let Q be a CQAC query, and let $\mathcal{V}$ be a set of full CQAC views, i.e., the views do not have any nondistinguished variables. Then it is NP hard to check whether a contained rewriting exists.* □

PROOF. The reduction uses the 3-colorability problem, where, given a graph G, the question is whether it is 3-colorable. The reduction is as follows: Let e be the predicate that defines an edge. Let query Q be a Boolean query, whose body is the graph G. Let view V be a triangle with all variables exported in the head and with the following inequalities in the body:

$$v(x,y,z) :- e(x,y), e(y,x), e(x,z), e(z.x), e(y,z), e(z,y),$$
$$2 \le x \le 3, 4 \le y \le 5, 6 \le z \le 7.$$

Observe that for view V, any non-empty rewriting will not equate any pair of x, y, z because, e.g., $2 \le x \le 3 \wedge 4 \le x \le 5$ cannot be satisfied by any value of x. The intuition is that the expansion of any non-empty rewriting is such that the graph of the expansion is 3-colorable.

We claim that there is a contained rewriting of the query Q using the view V iff the graph G is 3-colorable. Observe that Q has no ACs. Thus, if there is a contained rewriting R, then any mapping, say h, from the query to R^{exp} that proves containment can be used to produce a mapping from Q to the body of one view. The reason for that is that R_1^{exp} (which is R^{exp} without the ACs) can be mapped to the body of a single view, i.e., to a triangle. This mapping can be composed with h to map Q on the triangle. This proves that the graph is 3-colorable. For the other direction, if the graph is 3-colorable then the single view is a contained rewriting. □

Finally, consider the case where the query and views are conjunctive with semi-interval arithmetic comparisons (ACs). In this case, if there is a contained rewriting, then there is one that uses only semi-interval ACs and is of bounded size. Thus, the CQAC Contained Rewriting Existence Problem is in EXPTIME in this case.

THEOREM 3.5. *Let query Q and views $\mathcal{V}$ be conjunctive queries with semi-interval ACs. If there is a nontrivial contained rewriting of Q using $\mathcal{V}$, then there is a nontrivial contained rewriting that (1) uses only semi-interval ACs on constants that are contained either in view definitions or in the query, and (2) uses at most a number of relational subgoals that is exponential in the size of the query and views (actually, only the maximum arity of the head of a view definition appears in the exponent). Thus, there is a double exponential time algorithm that finds a non-trivial contained rewriting if such a rewriting exists. The respective decision problem is in NEXPTIME.* □

The following theorem summarizes the complexity results:

THEOREM 3.6. *1. The problem of CQ contained rewriting existence is NP-complete (Theorem 3.2).*

2. If the query and views are CQs and, in addition, views do not have nondistinguished variables, then we can find a contained rewriting in PTIME if one exists (Theorem 3.1).

3. If the views have no nondistinguished variables, then the problem of CQAC contained rewriting existence is NP complete (Theorems 3.3 and 3.4).

4. If the query and views are CQs with semi-interval arithmetic comparisons, then the problem of CQAC contained rewriting existence is in NEXPTIME (Theorem 3.5). □

Thus, even decidability in the general case of CQAC query and views and UCQAC rewritings remains open.

3.2 Reducing the CQAC case to the CQ case

In the remainder of this section, for the case of CQAC query and views we work toward an efficient heuristic algorithm that (1) checks whether there exists a nontrivial contained rewriting, and (2) produces such a rewriting. Before presenting the algorithm, we develop some results that will help us establish the efficiency of the algorithm.

We present Proposition 3.1: (1) It establishes a preliminary test for existence of a rewriting; the test is easier to apply, since it concerns CQ query and views (that is, without arithmetic comparisons). (2) Proposition 3.1 states that if a rewriting exists, then a "simple" rewriting also exists and is easier to find. Intuitively, the preliminary test asserts that we can construct CQ query and views, such that the non-existence of a rewriting of the CQ query using the CQ views implies the non-existence of a rewriting for the original query and views.

Let $Q = Q_0 + \beta$ be a query, and let $\mathcal{V}$ be a set of views $V_i = V_{i0} + \beta_i,\ i = 1, \ldots, k$. Consider 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. Can we 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 3.1. *For a query Q and view V,*

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

a contained rewriting is $r() :- v(X,Z),\ Z \le X$.
But by removing all the ACs, we obtain

$q'() :- p(X,X)$ *and* $v'(X,Z) :- p(X,Y),\ r(Z)$.

where a rewriting of Q' using V' is not possible. □

At the same time, with a slight modification in the definition of the views $\mathcal{V}_0$ we can take advantage of such a remark. To define $\mathcal{V}_0$, we need this definition from [6]:

DEFINITION 3.1. *(***Exportable view variable***) A nondistinguished variable X of a view V is* exportable *if and only if there exists a head homomorphism h (a mapping from the set H of head variables of V to H that maps a head variable in H to either itself or to another variable in 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 3.1 is an exportable variable, because if we equate $X = Z$ then the ACs in the view definition imply $Y = X = Z$.

Now, we define the set of CQ views $\mathcal{V}_0$ and state our result.

PROPOSITION 3.1. *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 set of views $\mathcal{V}_0$ with views as in $\mathcal{V}$ but (1) defined using only their relational subgoals, 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 $\mathcal{V}_0$, and*
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 3.1 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 $\mathcal{V}_0$.

Note that although we restricted the search space to rewritings with a total order on their variables and constants, we have not eliminated the source of some complications. As illustrated in the following example, we cannot take advantage of the homomorphism property that simplifies CQAC query containment and rewriting generation.

EXAMPLE 3.2. *Consider a query Q and its rewriting R using views V_1 and V_2:*

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

There exists no single containment mapping from the expansion $r^{exp}() : -p(3,X), p(X,4), X \le 4$ to the query that will satisfy the conditions of Theorem 2.1. Thus, the homomorphism property does not hold. The intuitive reason is that if $X < 4$, then the subgoal of the query will map to the $p(X,4)$ subgoal of the rewriting, whereas if $X = 4$, then the subgoal of the query will map to the $p(3,X)$ subgoal of the rewriting. □

3.3 Algorithm

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 for contained rewritings of Q_0 using $\mathcal{V}_0$ and halting if none exists, and then (2) considering containment checks in a systematic way, so that it does not repeat unnecessarily checks that can be deduced from the previous iterations. The complexity of the algorithm is doubly exponential. The algorithm is sound and complete for a broad class of queries and views, in fact for all cases stated in Theorem 3.6.

In conjunctive queries, shared variables express a join condition, that is, a constraint that two attributes should have the same value. When checking for containment of two queries, we have to ensure that this constraint is satisfied in the contained query. We define the *shared-variable condition* (SV condition) 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. 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 condition is satisfied. In all its stages, the algorithm uses only candidate views.

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.

The algorithm starts with a contained rewriting of Q_0 using $\mathcal{V}_0$, say with k view subgoals. Then stage 2 of the algorithm checks for containment all rewritings with $k+1$ view subgoals, stage 3 checks all rewritings with $k+2$ view subgoals, and so on. In each stage of the algorithm, the containment test is pruned as follows: 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 R's expansion) that did not pass the containment test. For all candidate views not yet added to R, the algorithm forms rewritings R', where each R' is R with an additional subgoal that comes from one of the candidate views – for each view we try all additional subgoals with all different head homomorphisms. Then the algorithm checks each R' for containment. The correctness of the algorithm is based on the following observation: 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 following is an example that explains how we find the view set $\mathcal{V}_0$ and why the exported variables are critical in the definition of $\mathcal{V}_0$ (and consequently the role they play in the rewriting of the original query and views).

EXAMPLE 3.3. *We refer to Example 3.1. For Q_0 we have $q_0() \ :- \ p(X,X)$. For V_0 we have $v_0(X,Z,Y) \ :- \ p(X,Y), \ r(Z)$, 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:

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

Hence, since there is such a rewriting (of Q_0 using V_0), the algorithm does not halt and continues to find a rewriting of Q using V. The algorithm considers only rewritings with a total order among the variables and constants in the rewriting. In this case, $X = Z$ is a total order among the variables of V that produces a rewriting of Q using V. Hence, the algorithm finds the rewriting

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

THEOREM 3.7. *Let a query and all views be CQAC.*

(1) If the algorithm halts, it produces a nontrivial contained rewriting if one exists.

(2) For conjunctive queries with semi-interval arithmetic comparisons, or when views have no nondistinguished variables, the algorithm produces a nontrivial contained rewriting if one exists. $\square$

In the absence of the homomorphism property, for CQACs there exists no bound on the size of the rewriting. Afrati et al. [5] have shown that the language of CQACs is not sufficient to express an MCR (recursive Datalog may be necessary). Our algorithm shown above considers rewritings of iteratively increasing sizes; the lack of a bound indicates that the algorithm is not guaranteed to halt.

4. CONTAINING REWRITINGS

In this section we discuss *containing rewritings* of queries using views. We consider containing rewritings that are safe, i.e., each variable in the head of a containing rewriting occurs in at least one subgoal in the body of the rewriting. We investigate decidability and complexity of the problem of existence of a safe containing rewriting. We develop a *pruned-MiCR* algorithm that finds minimally containing rewritings in an *efficient* and *scalable* way.

DEFINITION 4.1. *(**Minimally containing rewriting**) A query Q' is a* minimally containing rewriting *of a CQAC query Q using a set of CQAC 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'.* $\square$

Notice that containing rewritings and MiCRs make sense only under the closed-world assumption (CWA), i.e., given any instance I of a viewset V, $I = V(D)$ where D is the database of base relations. When the instance is only guaranteed to be a subset of $V(D)$, we refer to it as the open-world assumption (OWA), under which the evaluation of a rewriting may not produce all the database tuples that satisfy the query. In this case, generating MiCRs is not useful. The reason is, even though the expansion of the rewriting may contain the query, due to the nature of the views, we are not guaranteed that the rewriting will generate all the answers that the query computes on D. We could call such rewritings "overlapping rewritings". (Informally, an overlapping rewriting returns on evaluation a subset of the query answers, and may also return additional tuples — false positives.) We do not address the problem of generating overlapping rewritings in this paper.

A MiCR of a CQAC query using CQAC views may have to be a union of CQACs, as the following example shows. In this example there is no containing rewriting that is a CQAC.

EXAMPLE 4.1. *Consider a query Q and views V_1 and V_2, all defined as follows:*

$q() \ :- \ p(X,Y), \ X \le 10, Y \le 10.$
$v_1() \ :- \ p(X,10), \ X \le 20.$
$v_2() \ :- \ p(X,Y), \ X \le 20, Y < 10.$

The rewriting $R : (q() : -v_1()) \cup (q() : -v_2())$ is a containing rewriting of Q, because X there is less restricted than in the query and q's Y is covered by the union of the views. $\square$

We define the language $\mathbf{C} = \cup_{i=1}^{4} \mathbf{C}_i$, a union of four sub-languages, which is a subclass of CQACs and has good properties with respect to checking query containment. We denote by cLSI (oLSI, respectively) the closed (open, respectively) left-semi-interval arithmetic comparisons, and by cRSI (oRSI, respectively) the closed (open, respectively) right-semi-interval arithmetic comparisons.

DEFINITION 4.2. *(**Language C**) We say that an arithmetic comparison is of SI type 1,2,3, or 4 if it is an cLSI, oLSI, cRSI, or oRSI comparison, respectively.*

A query Q belongs to the class $\mathbf{C}_i$, $i = 1,2,3,4$, if Q is a CQ that uses solely arithmetic comparisons of SI type i.

We define $\mathbf{C} = \cup_{i=1}^{4} \mathbf{C}_i$ and say that a query is in class (or language) $\mathbf{C}$ and of type i *if it belongs to class $\mathbf{C}_i$.* $\square$

Klug [23] has shown that when queries are expressed in language $\mathbf{C}$ and are both of the same type then the homomorphism property holds [23].

We now present a theorem, which says that when queries and views are in one of the $\mathbf{C_i}$'s, then there exists a containing rewriting that is also in one of the four languages.

THEOREM 4.1. *For a query Q and a set of views $\mathcal{V}$, where Q and the views in $\mathcal{V}$ are defined using relational predicates from a set P, and such that both Q and $\mathcal{V}$ are expressed using the same sublanguage in $\mathbf{C}$, the following holds: If there exists a containing rewriting R of Q using $\mathcal{V}$, such that R is a union of CQAC queries, then there exists a query R' (in the sublanguage in $\mathbf{C}$) that is a containing rewriting of Q using $\mathcal{V}$.* □

4.1 Decidability and Complexity

We now examine the complexity of computing containing rewritings.

THEOREM 4.2. *Given a CQAC query and a set of CQAC views, it is decidable whether there exists a contaning rewriting that is a union of CQAC queries. Furthermore, there exists an algorithm for computing a MiCR.* □

The following lemma helps prove Theorem 4.2.

LEMMA 4.1. *Given a CQAC query q and a set of CQAC views V. If there exists a union-of-CQAC containing rewriting R of q using V, then there exists a containing rewriting R', which is contained in R and has at most a doubly exponential number of relational subgoals.* □

We now turn our attention to special cases. We show that finding a containing rewriting of a query using views expressed using one of the four languages in $\mathbf{C_i}$ indicated above is NP complete.

THEOREM 4.3. *Given (1) a query Q whose relational predicates belong to a set of predicates P, (2) a set of views $\mathcal{V}$ such that all views in $\mathcal{V}$ have relational predicates only from P, and given that (3) Q and $\mathcal{V}$ are expressed using the same language $\mathbf{C_i}$ in $\mathbf{C}$, it is NP complete to decide whether there exists a containing rewriting of Q using $\mathcal{V}$.* □

The proof that the problem is in NP uses the result of Lemma 4.2; NP-hardness is by reduction from CQ containment and is a consequence of the following theorem.

THEOREM 4.4. *Let Q be a CQ query, and let $\mathcal{V}$ be a set of CQ views. It is NP hard to decide whether there exists a containing rewriting in the language of union of CQACs of Q using $\mathcal{V}$.* □

For safe containing rewritings, the following lemma holds:

LEMMA 4.2. *If there exists a safe CQAC rewriting R of a CQAC query Q, $Q \sqsubseteq R$, and if the homomorphism property holds between R^{exp} and Q, then there exists a safe 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 Q.* □

PROOF SKETCH. Suppose the rewriting R exists. We try to get a bound on the size of R. There exists a containment mapping μ 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) 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. Note that R' is CQ, rather than CQAC, because the arithmetic predicates from R were dropped. Hence, the right-hand side of the implication being empty, one containment mapping is enough to show containment. □

For all Boolean queries, the query $q :-$, which says that q is always true, is a safe containing CQAC. Note that this is consistent with the theorem above, because the number of variables in the head of q and the number of subgoals in its body is the same (zero).

4.2 The pruned-MiCR Algorithm

In this subsection, we describe a sound and efficient algorithm, the *pruned-MiCR algorithm*, to find the MiCR. We assume that the queries and views are expressed using the same language C_i in C. However, in general, the algorithm is sound and complete when there exists a rewriting of the query in the language of CQACs such that the homomorphism property holds between the query and the expansion of the rewriting.

It turns out that a MiCR is unique up to equivalence as expansions.

THEOREM 4.5. *Under the CWA, and for queries and views in the same language $\mathbf{C_i}$ in $\mathbf{C}$, MiCR is unique up to equivalence as expansions.* □

PROOF. Let R_1 and R_2 be two MiCRs of a query Q expressed in the language of union of CQACs that are not equivalent as expansions. The expansions of R_1 and R_2 do not contain each other; otherwise they would not be minimally containing. For each canonical database D_i of Q, consider the rewriting $r1_i$ in R_1 and $r2_i$ in R_2 that produces a tuple, t, on D_i that is the same as that produced by Q. Consider the mappings from $r1_i$ and $r2_i$ to the constants in D_i, which produce the t. Because the constants in D_i are obtained by freezing the variables in the query, there exists a mapping from $r1_i$ and $r2_i$ to Q. Replace distinguished variables in $r1_i$ and $r2_i$ by the query variables they map to. Construct a rewriting r_i by conjoining the bodies of $r1_i$ and $r2_i$. D_i corresponds to a total order among the query variables and constants. Add all the ACs (including variable-equality constraints) in this total order to r_i wherever possible, i.e., on all query variables that appear in r_i. If the ACs indicate that two distinguished query variables are equal, make them equal in the head of r_i.

It is easy to show that this constructed rewriting is contained in both R_1 and R_2 — thus rendering R_1 and R_2 not minimal. This is a contradiction that concludes the proof. □

In the remainder of this section we explain the pruned-MiCR algorithm (see Appendix A), which is guaranteed to find a minimal MiCR when queries and views are expressed using the same language in $\mathbf{C}$. We define a *minimal MiCR* such that if a subgoal is deleted from the rewriting, then it is no longer a MiCR. The algorithm first finds all views whose bodies contain some query subgoals, and then constructs buckets for query subgoals and the views. A rewriting is constructed by conjoining one view subgoal from each

bucket. Each bucket in the algorithm represents a pair of subgoals: a query subgoal and a subgoal in the expansion of the view covering it. We clarify some points of the pruned-MiCR algorithm in the examples in the remainder of this section.

Unlike algorithms for determining MCRs, in the pruned-MiCR algorithm a non-distinguished view variable is allowed to map to a distinguished query variable. In order to determine the minimally containing rewriting, we need to keep the views that cover the queries the most "tightly". We formalize the definition of "tightness" with respect to covering in the following definition, and illustrate the intuition behind the definition in Example 4.2:

DEFINITION 4.3. *(***Dominates***) Let there be a mapping μ_1 from a view subgoal $g(\bar{X})$ to a query subgoal $g(\bar{Z})$, and a mapping μ_2 from another view subgoal $g(\bar{Y})$ to $g(\bar{Z})$. $(g(\bar{X}), ac1)$ is said to* dominate *$(g(\bar{Y}), ac2)$, where $ac1, ac2$ are sets of arithmetic comparisons, iff there is a mapping μ from $\bar{X}$ to $\bar{Y}$ but not vice versa,[6] which respects the following conditions (here, X_i is the i^{th} variable in $\bar{X}$):*

- *if $X_i = X_j$, then $\mu(X_i) = \mu(X_j)$;*
- *if μ_1 maps a non-distinguished variable X_i to a distinguished variable Z_i, then Y_i is also non-distinguished;*
- *$ac2 \Rightarrow \mu(ac1)$; and*
- *if $X_i = c$, where c is a constant, then $Y_i = c$.* □

If none of the conditions above are satisfied, then the first (subgoal, ac) combination is said to *dominate* the second one. The "dominates" relation can be extended to two sets, SS1 and SS2, of view subgoals covering the same set of query subgoals, by checking that each subgoal $(g1, ac1)$ in SS1 dominates the corresponding subgoal $(g2, ac2)$ in SS2. Two view subgoals are said to correspond if they cover the same query subgoal in their respective mappings.

The second condition states that a distinguished variable that maps to a distinguished query variable dominates a non-distinguished variable that maps to the same query variable. When comparing distinguished (non-shared) variables and non-distinguished variables, either is not dominated by the other, and we have to choose based on the other variables. In Example 4.2 we illustrate the details of bucket construction and dominance checking, techniques that provide the early pruning for the pruned-MiCR algorithm. Note that in our algorithm, a bucket corresponds to a (view-subgoal, query-subgoal) pair instead of corresponding to only query subgoals as in the traditional algorithms for determining contained rewritings. We keep the view subgoal that covers a query subgoal, so that we know how tightly the view subgoal covers the query subgoal, and use it to check for dominance, as shown below.

EXAMPLE 4.2. *Let:*

$Q(A)$	$:- p(A,B,B),\ q(A)$
$v1()$	$:- p(X1,Y1,Z1)$
$v2(X2)$	$:- p(X2,Y2,Z2)$
$v3(X)$	$:- p(X,Y,Y)$
$v4(X4,Y4)$	$:- p(X4,Y4,Y4)$
$v5(Y)$	$:- q(Y)$

The algorithm first considers $v1$ and inserts the view subgoal $v1()$ into the bucket $(p(A,B,B), p(X1,Y1,Z1))$ corresponding to the first query subgoal and the view subgoal pair.

[6]For minimization purposes, we keep all the "domination equivalent subgoals".

Next it considers $v2$. The view subgoal $p(X2,Y2,Z2)$ has a mapping to $p(A,B,B)$. The subgoal $p(X2,Y2,Z2)$ dominates $p(X1,Y1,Z1)$ primarily because $X2$ is a distinguished variable mapping to a distinguished query variable but $X1$ is a non-distinguished variable. A bucket for $(p(A,B,B), p(X2,Y2,Z2))$ is created, and $v2(A)$ is inserted into the bucket.

Next, $v3$ is considered, and a bucket is created for the pair $(p(X,Y,Y), p(A,B,B))$ corresponding to the subgoal in $v3$ and the first subgoal in Q. The algorithm compares the new bucket with the existing buckets (i.e., the bucket with $v2(A)$). The subgoal $p(X,Y,Y)$ dominates $p(X2,Y2,Z2)$ by virtue of being more contained. Thus, we delete the existing bucket, create a bucket for $(p(A,B,B), p(X,Y,Y))$, and insert $v3(A)$ into the bucket.

Next $v4$ is considered. $Y4$ is distinguished but does not map to a shared or distinguished variable. Therefore, it does not dominate $p(X,Y,Y)$. Thus, in the existing bucket we insert $v4$ and now have $v3(A), v4(A,B)$.

Finally, $v5$ goes to the bucket corresponding to the second query subgoal, i.e., for the (query-subgoal, view-subgoal) pair $(q(A), q(Y))$.

Thus, we have two rewritings:

```
Q(A)  :-  v3(A),v5(A)
Q(A)  :-  v4(A,B), v5(A)
```

either of which could be chosen as the "minimal" MiCR. □

In order to be a MiCR, a rewriting must satisfy the following properties: (a) it has to cover as many query subgoals as possible (it is ok not to cover some query subgoals as long as they are not "coverable" by any existing view), (b) a query subgoal has to be covered in a dominating way (i.e., in a less relaxed way, explained below), and (c) homomorphism h from a view subgoal to the query has to be "legal", i.e., there is a homomorphism from the view to the query, such that h is its sub-homomorphism.

Even though the algorithm tries to find one MiCR, 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 (dominates) — 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 an unexamined view might be covering all the subgoals in the query. A view covering all query subgoals in the tightest possible manner renders all other views redundant, because it is, by itself, a MiCR.

THEOREM 4.6. *If a query Q and views $\mathcal{V}$ are expressed using the language $\mathbf{C}$, then the algorithm in Appendix A finds a MiCR of Q using $\mathcal{V}$.* □

4.3 Scalability of the Pruned-MiCR Algorithm

Deutsch et al. [13] have remarked on a way to produce MiCRs using a chase-based method. However, the rewritings

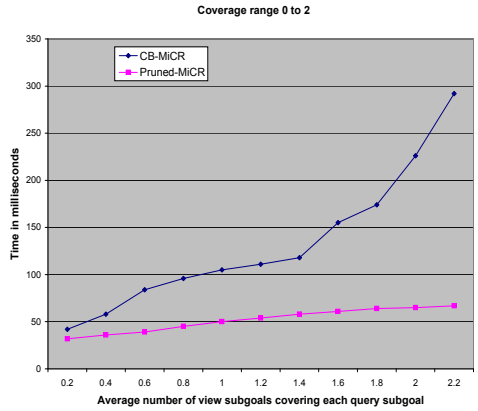

Figure 1: Scalability of the pruned-MiCR algorithm.

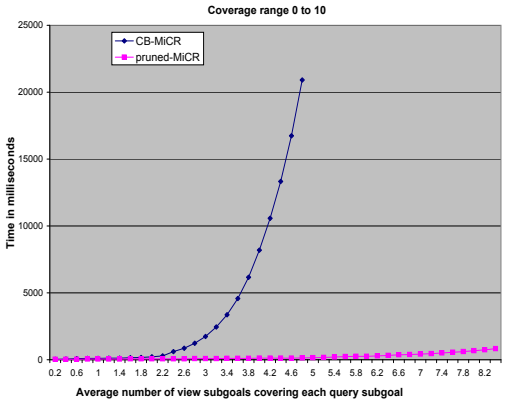

Figure 2: Speedup offered by the pruned-MiCR algorithm over the CB-MiCR approach.

produced using the method are not minimal. As an alternative to the pruned-MiCR algorithm, we consider the Chase-Backchase MiCR algorithm (CB-MiCR). The CB-MiCR uses chase to produce a MiCR and then it uses backchase to produce a minimal MiCR.

In Section 5 we report our experimental results on comparing the scalability of pruned-MiCR and CB-MiCR. The scalablity of pruned-MiCR stems from early pruning of dominated views. Consider Example 4.2. The CB-MiCR algorithm would first generate the rewriting
$Q(A) : -v1(), v2(A), v3(A), v4(A, B), v5(A)$
using the chase method, before performing a costly minimization (using backchase) that removes redundant subgoals. In contrast, the pruned-MiCR algorithm prunes dominated views $v1$ and $v2$ early and produces only those candidate rewritings that do not include them. Furthermore, as an additional optimization, pruned-MiCR does not consider the candidate rewriting $v3(A), v4(A, B), v5(A)$. The reason is, $v3$ and $v4$ cover the same query subgoal with similar "tightness" (i.e., they do not dominate each other) and therefore, only one of them needs to appear in the rewriting. As remarked earlier, we keep both in the same bucket only for minimization purposes. As shown in Section 5, by pruning early the pruned-MiCR algorithm outperforms the CB-MiCR algorithm significantly.

5. EXPERIMENTAL RESULTS

We have performed experiments to compare the execution time of the pruned-MiCR algorithm with that of the CB-MiCR algorithm. We measured the scalability of the two algorithms in the number of views.

Similarly to the pruned-MiCR algorithm, the CB-MiCR algorithm finds all possible $\mu_i(h(V))$'s from views in $\mathcal{V}$ to the query Q.[7] After this stage, the pruned-MiCR algorithm uses its novel strategy to distribute the view subgoals into buckets, and then constructs a minimal MiCR by ensuring that each bucket is represented in the rewriting. It tries all subsets of the set of possible $\mu_i(h(V))$'s; candidate rewritings are formed by taking the conjunction of the subgoals in any such subset. The CB-MiCR algorithm considers the same candidate rewritings. However, in the absence of the MiCR-buckets it is forced to perform an expensive containment test for each candidate, to check if the expansion of the candidate is contained in the expansion of the full MiCR (i.e., in the conjunction of all $\mu_i(h(V))$'s). Our experiments demonstrate that the pruned-MiCR algorithm speeds up rewriting generation, since it eliminates containment checks by doing an early pruning in the process of generating a minimal MiCR.

Both algorithms were implemented in Java and compiled to executables. All experiments were run on a 2 GHz Pentium M processor running Windows XP Professional with 1 GB RAM and a 60GB hard drive. The run-times were averaged over twelve executions, after discarding the minimum and maximum readings.

We studied the effect of increasing the number of views for chain queries. Figure 1 shows the results for a chain query with ten subgoals. It shows that the execution time of the CB-MiCR algorithm increases rapidly with an increase in coverage (i.e., the average number of view subgoals covering each query subgoal). Note that if each query subgoal is covered by at most one view, the pruned-MiCR's early pruning is not used because there are no views that dominate each other. However, when there are multiple views covering a query subgoal, the advantages of the early pruning are substantial. The execution speedup resulting from the use of the pruned-MiCR algorithm is evident even at the low coverage values of up to 2. Our experiments with other query shapes such as star queries, cycles and complete queries show that it is the increase in the coverage that causes a rapid deterioration in the performance of the CB-MiCR algorithm. Consequently, the choice of the queries and views may adversely affect its performance. For example, with star queries and views, the CB-MiCR algorithm had a run-time of around 3000ms for a coverage of just 1.5, while the pruned-MiCR algorithm took less than 100ms.

Figure 2 shows that the pruned-MiCR algorithm executes efficiently even for high coverage values. This is in sharp contrast to the CB-MiCR algorithm, which takes more than 20000ms even for coverages below 5. The increase in the run-time of the pruned-MiCR algorithm at higher coverage values is marginal and stems from the increased time required for forming the MiCR-buckets. In practice, constructing rewritings in the pruned-MiCR algorithm takes time that is about linear in the number of buckets, which

[7]This is equivalent to chasing Q with forward constraints obtained from the views in $\mathcal{V}$. Each homomorphism on a view head from $\mathcal{V}$ that gets added to the chase result of Q is analogous to some $\mu_i(h(V))$ obtained by the MiCR algorithm.

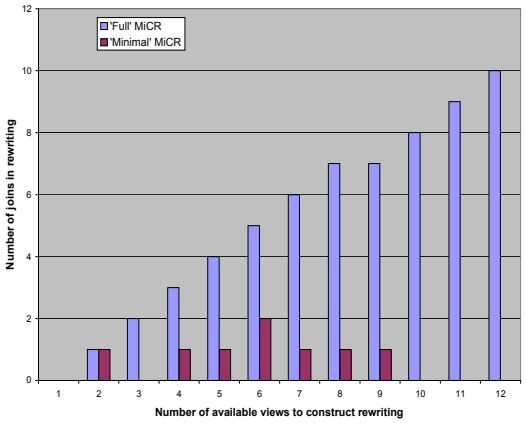

Figure 3: Comparison of the number of joins in the full and minimal MiCR rewritings.

Number of available views	Query/Views	Number of joins		Coverage
		Full MiCR	Minimal MiCR	
	Q() :- p(A, B), r(B, C), s(C, D), t(D, E)			
1	v1() :- p(A, B)	0	0	0.25
2	v2() :- r(B, C)	1	1	0.50
3	v3() :- p(A, B), r(B, C)	2	0	1.00
4	v4() :- s(C, D)	3	1	1.25
5	v5() :- r(B, C), s(C, D)	4	1	1.75
6	v6() :- t(D, E)	5	2	2.00
7	v7() :- s(C, D), t(D, E)	6	1	2.50
8	v8() :- r(B, C), s(C, D), t(D, E)	7	1	3.25
9	v9() :- u(L, M)	7	1	3.25
10	v10() :- p(A, B), r(B, C), s(C, D), t(D, E)	8	0	4.25
11	v11() :- s(C, D), u(L, M)	9	0	4.50
12	v12() :- r(B, C), t(D, E)	10	0	5.00

Figure 4: Example query and views.

does not slow down significantly the overall algorithm execution. The CB-MiCR algorithm spends significant time (potentially exponential in the number of subgoals in the rewriting) on testing each candidate rewriting for containment in the full MiCR. In our experiments, we observed that typically even a single containment test took more time than the entire bucket-forming procedure over all views in the input. The scalability of the pruned-MiCR algorithm makes it useful in finding minimal MiCRs for practical cases involving a large number of applicable views.

Any new view that is made available to generate a MiCR for the query, may generate 0 or more $\mu_i(h(V))$'s. Every time that a $\mu_i(h(V))$ is made available: (i) the number of joins in the MiCR increases by 1, to attain a new value of say n, and (ii) the number of joins in the minimal MiCR increases by 1, decreases by any amount, or remains the same, to take on some value between 0 and n. In the worst case, the maximum value that n can take is one less than the sum, over all query subgoals, of the number of $\mu_i(h(V))$'s covering that subgoal. The number of joins, both in the full and minimal MiCR, depends upon the number of views in the input. In general, a plot of the number of joins versus the number of available views may take any shape subject to conditions (i) and (ii) above, and normally the number of joins in a minimal MiCR is significantly lower than the number of joins in the full MiCR. In particular, once all query subgoals have been covered, the number of joins in a minimal MiCR can only decrease or remain the same, whereas the number of joins in the full MiCR go on increasing with every new $\mu_i(h(V))$. Figure 3 illustrates these ideas for the simple example of Figure 4.

In summary, we have shown that (i) the pruned-MiCR algorithm outperforms the CB-MiCR algorithm significantly due to its early pruning, and that (ii) the pruned-MiCR algorithm scales gracefully with an increase in the number of views and is able to generate rewritings within a reasonable time even for a very large number of views.

6. CONCLUSIONS

In this paper we explored the problems of existence of contained and containing rewritings in the language of union of CQACs for queries and views that are CQACs. We outlined complexity results and proposed a sound algorithm to determine the existence of a contained rewriting of a query using views. The algorithm is also complete when the query and views are expressed using conjunctive queries with semi-interval arithmetic comparisons and when we use views that do not have nondistinguished variables. We also examined the problem of generating minimally containing rewritings using views. We showed that the decision problem is NP-hard in general and is NP-complete when the query and views are expressed using a class of conjunctive queries with semi-interval arithmetic predicates. We introduced a pruned-MiCR algorithm that extends a chase-based algorithm, using backchase to derive minimal MiCRs. The algorithm is sound and complete for queries and views expressed in a subclass of conjunctive queries with semi-interval arithmetic comparisons, as well as for other cases where the homomorphism property (see Footnote 1) holds between the expansion of the rewriting and the query. By pruning dominated views early, the pruned-MiCR algorithm outperforms the CB-MiCR algorithm (that has to perform an exponential minimization step) significantly in practice, and scales well for reasonably sized queries. For both contained and containing rewritings, similar problems remain open for a more general class of queries and views than the classes we considered here.

7. ADDITIONAL AUTHORS

8. REFERENCES

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

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

[3] F. Afrati, R. Chirkova, M. Gergatsoulis, and V. Pavlaki. Finding equivalent rewritings in the presence of arithmetic comparisons. In *EDBT*, pages 941–960, 2006.

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

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

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

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

[8] R. Bayardo, W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Semantic integration of information in open and dynamic environments. In *SIGMOD*, pages 195–206, 1997.

[9] D. Calvanese, G. Giacomo, M. Lenzerini, and M. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. In *ICDT*, pages 321–326, 2005.
[10] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *VLDB*, pages 111–122, 2000.
[11] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. *ACM STOC*, pages 77–90, 1977.
[12] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *ICDE*, pages 190–200, 1995.
[13] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. In *ICDT*, pages 352–367, 2005.
[14] O. Duschka and M. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997.
[15] W. Fan, C. Chan, and M. Garofalakis. Secure XML querying with security views. In *SIGMOD*, pages 587–598, 2004.
[16] D. Florescu, A. Levy, D. Suciu, and K. Yagoub. Optimization of run-time management of data intensive web-sites. In *VLDB*, pages 627–638, 1999.
[17] A. Fuxman, E. Fazli, and R. Miller. Conquer: Efficient management of inconsistent databases. In *SIGMOD*, pages 155–166, 2005.
[18] G. Grahne and A. Mendelzon. Tableau techniques for querying information sources through global schemas. In *ICDT*, pages 332–347, 1999.
[19] A. Gupta, Y. Sagiv, J. Ullman, and J. Widom. Constraint checking with partial information. In *PODS*, pages 45–55, 1994.
[20] L. Haas, D. Kossmann, E. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *VLDB*, pages 276–285, 1997.
[21] A. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(3):270–294, 2001.
[22] Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld. An adaptive query execution engine for data integration. In *SIGMOD*, pages 299–310, 1999.
[23] A. Klug. On conjunctive queries containing inequalities. *JACM*, 35(1):146–160, 1988.
[24] A. Lee, A. Koeller, A. Nica, and E. Rundensteiner. Non-equivalent query rewritings. In *International Database Conference, Hong Kong*, July 1999.
[25] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.
[26] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, pages 251–262, 1996.
[27] G. Miklau. *Confidentiality and Integrity in Data Exchange*. PhD thesis, University of Washington, 2005.
[28] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, pages 575–586, 2004.
[29] P. Mitra. An algorithm for answering queries efficiently using views. In *Proceedings of the Australasian Database Conference*, 2001.
[30] V. Poosala, V. Ganti, and Y. Ioannidis. Approximate query answering using histograms. *IEEE Data Engineering Bulletin*, 22(4):5–14, 1999.
[31] R. Pottinger and A. Halevy. MiniCon: A scalable algorithm for answering queries using views. *VLDB Journal*, 2001.
[32] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. In *VLDB*, 2000.
[33] X. Qian. Query folding. In *ICDE*, pages 48–55, 1996.
[34] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD*, pages 551–562, 2004.
[35] D. Theodoratos and T. Sellis. Data warehouse configuration. In *VLDB*, 1997.
[36] J. Ullman. Information integration using logical views. In *ICDT*, pages 19–40, 1997.
[37] J. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.
[38] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *PODS*, pages 331–345, 1992.
[39] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata. Answering complex SQL queries using automatic summary tables. In *SIGMOD*, 2000.

APPENDIX

A. PSEUDOCODE FOR MICR

The pseudocode of the algorithm for MiCR is given in this optional appendix.

```
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
            ac_view ← AC(h(v))
            for each ac_i ∈ AC(Q) do
                if all variables in ac_i appear in h(v) then
                    ac_view ← ac_view ∧ ac_i
            if AC(Q) ⇒ μ_i(ac_view)) then
                for each subgoal g in Q that h(v) covers do
                    Let μ_i map g_i in exp(h(v)) to g
                    { SS is the set of query subgoals covered by the view for which a bucket is being constructed. }
                    SS ← g
                    for each non-distinguished view variable that μ_i maps to a non-distinguished, shared query variable Y
                    do
                        Add all subgoals in Q that Y appears in and that appears in the range of μ_i(h(V)) to SS
                    { project(ac, g) outputs only those arithmetic comparisons in ac whose variables appear in g }
                    Let ac_i = project(ac_view, SS)
                    create_bucket ← true
                    for each Bucket(S_j, ac_j) covering SS do
                        if (S_j, ac_j) dominates (S_i, ac_i) then
                            { (S_j, ac_j) is redundant }
                            delete Bucket(S_j, ac_j)
                        else
                            if (S_i, ac_i) dominates (S_j, ac_j) then
                                { (S_i, ac_i) is redundant, do not need to do anything }
                                create_bucket ← false
                                break
                            else
                                if (S_i, ac_i) and (S_j, ac_j) are equivalent then
                                    { (S_i, ac_i) and (S_j, ac_j) are equivalent, no need to create a new bucket }
                                    create_bucket ← false
                                    Insert h(v), ac_i into Bucket(S_i, ac_i)
                    if create_bucket == true then
                        create Bucket(SS, ac_i) covering g
                        insert h(v), ac into Bucket(SS, ac_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 a MiCR. }
    Construct a rewriting by taking a conjunction of the selected views and their associated arithmetic predicates.
end
```