Answering Queries Using Views

(Extended Abstract)

Alon Y. Levy
AT&T Bell Laboratories
levy@research.att.com

Alberto O. Mendelzon*
University of Toronto
mendel@db.toronto.edu

Yehoshua Sagiv†
Hebrew University, Jerusalem
sagiv@cs.huji.ac.il

Divesh Srivastava
AT&T Bell Laboratories
divesh@research.att.com

Abstract

We consider the problem of computing answers to queries by using materialized views. Aside from its potential in optimizing query evaluation, the problem also arises in applications such as Global Information Systems, Mobile Computing and maintaining physical data independence. We consider the problem of finding a *rewriting* of a query that uses the materialized views, the problem of finding *minimal* rewritings, and finding *complete* rewritings (i.e., rewritings that use only the views). We show that all the possible rewritings can be obtained by considering containment mappings from the views to the query, and that the problems we consider are NP-complete when both the query and the views are conjunctive and don't involve built-in comparison predicates. We show that the problem has two *independent* sources of complexity (the number of possible containment mappings, and the complexity of deciding which literals from the original query can be deleted). We describe a polynomial time algorithm for finding rewritings, and show that under certain conditions, it will find the minimal rewriting. Finally, we analyze the complexity of the problems when the queries and views may be disjunctive and involve built-in comparison predicates.

1 Introduction

We consider the problem of using materialized views to answer queries. Aside from its potential of improving performance of query evaluation [LY85, YL87, KB94, CKPS95], the ability to use views is important in other applications. For example, in applications such as Global Information Systems [LSK95], Mobile Computing [BI94, HSW94], view adaptation [GMR95], maintaining physical data independence [TSI94], the relations mentioned in the query may either not actually be physically stored (e.g., they may be only conceptual relations), or be impossible to consult (e.g. they are stored in a remote server that is temporarily unavailable to a mobile computing device), or be very costly to access.

We consider the complexity of this problem and its variants and describe algorithms for solving them for conjunctive queries involving built-in comparison predicates and for unions of conjunctive queries. Specifically, we consider the problem of finding a rewriting of a query that uses a set of views, the problem of finding a *minimal* such rewriting, and the problem of *completely* solving a query using views, that is, finding rewritings that use nothing but the views and built-in predicates.

The observation underlying our solution of the problem is a general characterization of the usability of views in terms of the problem of query containment. As a consequence, we show that the *all* possible rewritings of a query can be obtained by considering *containment mappings* [CM77] from the bodies of the views to the body of the query. Given this characterization, we show that problems of finding rewritings that mention as few of the database relations as possible are NP-complete for conjunctive queries with no built-in predicates. In fact, we show that these problems have two *independent* sources of complexity. The first comes from the number of possible mappings from the views to the query, and the second source of complexity is determining which literals of the query can be removed when the view literals are added to the query. We describe a polynomial time algorithm for finding literals of the query that can be removed. This algorithm is guaranteed to remove only literals that are necessarily redundant in the rewriting, and we show that under certain conditions (which are likely to cover many practical cases), it is guaranteed to remove the unique maximal set of redundant literals. This algorithm, together with an algorithm for enumerating containment mappings from the views to the query, provides a practical method for finding rewritings of a query. Finally, we show how the presence of built-in predicates in the queries and in the views affect the algorithms and the complexity of the problems.

*Work performed while visiting AT&T Bell Laboratories. Supported by NSERC and ITRC.
†Work supported in part by BSF grant #92-00360.

2 Preliminaries

In our discussion we refer to the relations used in the query as the *database relations*. We consider mostly conjunctive queries, which may in addition contain built-in comparison predicates ($=, \neq, <$ and $\leq$). We briefly describe how our results can be extended to queries that involve unions of conjunctive queries (i.e., Datalog without recursion). We use $V, V_1, \ldots, V_m$ to denote views that are defined on the database relations. Views are also defined by conjunctive queries.

2.1 Rewritings

Given a query Q, our goal is to find an equivalent rewriting Q' of the query that uses one or more of the views:

Definition 2.1: *A conjunctive query Q' is a* rewriting *of Q that uses the views $\mathcal{V} = V_1, \ldots, V_m$ if*

- *Q and Q' are equivalent (i.e., produce the same answer for any given database), and*
- *Q' contains one or more occurrences of literals of $\mathcal{V}$.*

Note that we consider the case in which the rewriting is also a conjunctive query. When queries involve built-in predicates we will see that it may be worthwhile to consider rewritings involving unions.

We say that a rewriting Q' is *locally minimal* if we cannot remove any literals from Q' and still retain equivalence to Q. A rewriting is *globally minimal* if there is no other rewriting with fewer literals.[1]

Example 2.2: Consider the following query Q and view V:

$$
\begin{aligned}
Q: q(X,U) &\ :-\ p(X,Y), p_0(Y,Z), p_1(X,W), p_2(W,U).\\
V: v(A,B) &\ :-\ p(A,C), p_0(C,B), p_1(A,D).
\end{aligned}
$$

The query can be rewritten using V as follows:

$$Q': q(X,U) \ :-\ v(X,Z), p_1(X,W), p_2(W,U).$$

Substituting the view enabled us to remove the first two literals of the query. Note, however, that although the third literal in the query is guaranteed to be satisfied by the view, we could not remove it from the query. This is because the variable D is projected out in the head of V, and therefore, if the literal of p_1 were removed from the query, the join condition between p_1 and p_2 would not be enforced. □

Clearly, we would like to find rewritings that are cheaper to evaluate than the original query. The cost of evaluation depends on many factors that differ from one application to another. In this paper we consider rewritings that reduce the number of literals in the query, and in particular, reduce the number of database relation literals in the rewritten query. In fact, we show that any rewriting of Q that contains a minimal number of literals is isomorphic to a query that contains a subset of the literals of Q and a set of view literals. Although we focus on reducing the number of literals, it should be noted that rewritings can yield optimizations even if we do not remove literals from the query, as illustrated by the following example.

Example 2.3: Using the same query as in Example 2.2, suppose we have the following view:

$$v_1(A) \ :-\ p(A,C), p_1(A,D).$$

We can add the view literal to the query to obtain the following rewritten query.

$$
\begin{aligned}
q(X,U) \ :-\ & v_1(X), p(X,Y), p_0(Y,Z), p_1(X,W),\\
& p_2(W,U).
\end{aligned}
$$

The view literal acts as a filter on the values of X that are considered in the query. It restricts the set of values of X to those that appear in the join of p and p_1. □

In some applications we may not have access to any of the database relations. For example, in Global Information Systems [LSK95], the relations used in a query are only *virtual*, and the actual data is all stored in views defined over these relations. Therefore, it is important to consider the problem of whether the query can be rewritten using *only* the views. We call such rewritings *complete rewritings*:

Definition 2.4: *A rewriting Q' of Q, using $\mathcal{V} = V_1, \ldots, V_m$ is a* complete rewriting *if Q' contains only literals of $\mathcal{V}$ and built-in predicates.*

Example 2.5: Suppose that in addition to the query and the view of Example 2.2 we also have the following view:

$$V_2: v_2(A,B) \ :-\ p_1(A,C), p_2(C,B), p_0(D,E).$$

The following is a complete rewriting of Q that uses V and V_2:

$$Q'': q(X,U) \ :-\ v(X,Z), v_2(X,U).$$

It is important to note that this rewriting cannot be achieved in a stepwise fashion by first rewriting Q using V and then trying to incorporate V_2 (or the other way around). This is because the relation p_0, which appears in V_2 does not even appear in Q' which is the intermediate result of using V in Q. Finding the complete rewriting requires that we consider the usages of both views in *parallel*. □

[1] Note that in counting the number of literals in the query, we ignore the literals of the built-in predicates.

2.2 Containment Mappings

In the next section we show that the problem of finding a rewriting is closely related to the query containment problem. *Containment mappings* [CM77] have been used to show containment among conjunctive queries. In this paper we show that they also provide the core of the solution to the problem of finding the possible usages of a view. Formally, a containment mapping from a query Q_1 to a query Q_2 is a mapping from the variables of Q_1 into the variables of Q_2, such that every literal in the body of Q_1 is mapped to a literal in Q_2. (Note that to show that Q_1 contains Q_2, the containment mapping must also map the head of Q_1 to the head of Q_2; however, in this paper we use the term containment mapping to refer only to mappings on the bodies of the queries). In Example 2.2, the correctness of the rewriting can be established by considering the containment mapping $\{A \rightarrow X, B \rightarrow Z, C \rightarrow Y, D \rightarrow W\}$.

When neither Q_1 nor Q_2 contain built-in predicates, finding a containment mapping is a necessary and sufficient condition for deciding that Q_1 contains Q_2, and is an NP-complete problem [CM77]. This remains true also when Q_2 contains built-in predicates. However, when Q_1 contains built-in predicates, finding a containment mapping provides only a sufficient condition, and the containment problem in this case is Π_2^p-complete [vdM92]. In order to generalize our results to queries containing built-in predicates it is useful to note how containment mappings are also used to show containment of such queries. In particular, it follows from [LS93] that if Q_1 contains Q_2, then there exist queries $Q_2^1, \ldots, Q_2^n$ such that:

- $Q_2^1, \ldots, Q_2^n$ differ only in their built-in literals, and
- Q_2 is equivalent to the union of $Q_2^1, \ldots, Q_2^n$, and
- For every i, $1 \leq i \leq n$, there is a containment mapping ϕ_i from Q_1 to Q_2^i, such that $bi(Q_2^i)$ entails $\phi_i(bi(Q_1))$, where $bi(Q)$ is the conjunction of built-in atoms in the query Q.

For example, consider the following queries, where Q_1 contains Q_2:

$$\begin{aligned} Q_1 : q(Y) &\ :- \ e(Y), r(U_1, V_1), U_1 \leq V_1. \\ Q_2 : q(X) &\ :- \ e(X), r(U, V), r(V, U). \end{aligned}$$

The query Q_2 can be represented by the union:

$$\begin{aligned} Q_2^1 : q(X) &\ :- \ e(X), r(U, V), r(V, U), U \leq V. \\ Q_2^2 : q(X) &\ :- \ e(X), r(U, V), r(V, U), U \geq V. \end{aligned}$$

The containment mappings would be

$$\begin{aligned} &\phi_1 : \{Y \rightarrow X, U_1 \rightarrow U, V_1 \rightarrow V\}. \\ &\phi_2 : \{Y \rightarrow X, U_1 \rightarrow V, V_1 \rightarrow U\}. \end{aligned}$$

In the next section we consider the complexity of finding rewritings, minimal rewritings and complete rewritings.

3 Complexity of Finding Rewritings

Previous solutions to the problem of using views to answer queries were based on either finding syntactic or 1-1 mappings from the view to the query. The first observation underlying our solution is that the problem of using views is closely related to the problem of query containment. In fact, the proposition below gives a necessary and sufficient condition for the existence of a rewriting of Q that includes a view V.

Proposition 3.1: *Let Q and V be conjunctive queries with built-in predicates. There is a rewriting of Q using V if and only if $\pi_\emptyset(Q) \subseteq \pi_\emptyset(V)$, i.e., the projection of Q onto the empty set of columns is contained in the projection of V onto the empty set of columns.*

Note that the containment $\pi_\emptyset(Q) \subseteq \pi_\emptyset(V)$ is equivalent to the following statement: If V is empty for a given database, then so is Q.

The importance of this proposition is in the fact that it provides a *complete* characterization of the problem of using views, thereby enabling us to explore the different aspects of the problem.

Proposition 3.1 and earlier results on the complexity of containment [CM77, vdM92] entail the following complexity results on the problem of finding a rewriting of Q that uses a set of views $\mathcal{V}$:

Proposition 3.2: *Let Q be a query and $\mathcal{V}$ be a set of views.*

1. *If Q is a conjunctive query with built-in predicates and $\mathcal{V}$ are conjunctive views without built-in predicates, then the problem of determining whether there exists a rewriting of Q that uses $\mathcal{V}$ is NP-complete.*
2. *If both Q and $\mathcal{V}$ are conjunctive and have built-in predicates, then the problem of deciding whether there exists a rewriting of Q that uses $\mathcal{V}$ is Π_2^p-complete.*

Remark: Proposition 3.1 holds for a broader class of queries and rewritings. In particular, suppose $q(\bar{X})$ is any relational calculus query, (or, equivalently, in nonrecursive datalog with negation) as is the view $v(\bar{W})$, and suppose we are considering *conjunctive rewritings*, which are formulas of the form

$$q(\bar{X}) \wedge (\exists \bar{Z}) v(\bar{Y})$$

such that the following equivalence holds:

$$q(\bar{X}) \equiv q(\bar{X}) \wedge (\exists \bar{Z}) v(\bar{Y})$$

Note that $\bar{X}$, $\bar{Y}$ and $\bar{Z}$ are tuples of variables, such that $\bar{Z}$ includes exactly those variables of $\bar{Y}$ that do not appear in $\bar{X}$. Then such a rewriting exists if and only if $\pi_\emptyset(Q) \subseteq \pi_\emptyset(V)$. $\Box$

The proof of Proposition 3.1 constructs a rewriting of Q using V in which the literal of V contains *new* variables that did not occur originally in Q. The following lemma shows that we can always find a rewriting that does not introduce new variables. The lemma also shows that we do not need to consider rewritings that include database-relation literals that do not appear in the original query, i.e., that it is enough to consider rewritings that include view literals and a *subset* of the original literals in the query. These results enable us to significantly prune the search for a minimal rewriting of Q.

Lemma 3.3: *Let Q be a conjunctive query without built-in predicates*

$$q(\bar{X}) \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n)$$

and $\mathcal{V}$ be a set of views without built-in predicates.

1. *If Q' is a locally minimal rewriting of Q using $\mathcal{V}$, then the set of database-relation literals in Q' is isomorphic to a subset of the literals of Q.*

2. *If*

$$q(\bar{X}) \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n), v_1(\bar{Y}_1), \ldots, v_k(\bar{Y}_k).$$

is a rewriting of the query using the views, then there exists a rewriting of the form

$$q(\bar{X}) \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n), v_1(\bar{Y}'_1), \ldots, v_k(\bar{Y}'_k).$$

where $\{\bar{Y}'_1 \cup \ldots \cup \bar{Y}'_k\} \subseteq \{\bar{U}_1 \cup \ldots \cup \bar{U}_n\}$, i.e., a rewriting that does not introduce new variables.

3. *If Q and $\mathcal{V}$ include built-in predicates, then a rewriting as specified in part 2 exists, with the only difference that the rewriting may be a union of conjunctive queries.*

Note that even though in part 2 of the lemma the rewriting includes all the literals of the query, the set of variables will not increase as a result of removing redundant literals. Therefore, the lemma implies that we can find a minimal rewriting that does not introduce new variables.

Proof: To prove the first part of the lemma, let Q' be a locally minimal rewriting of Q using a set of views $\mathcal{V}$. Let Q'' be the expansion of Q' obtained by replacing every occurrence of a view $V \in \mathcal{V}$ by the body of V, with suitable variable renamings. For any conjunctive query R, let $L(R)$ denote the set of literals database-relation literals in the body of R. Since Q'' and Q are equivalent, there are containment mappings ϕ from Q to Q'' and κ from Q'' to Q. Let C be a *core* of Q, that is, a minimal subset of $L(Q)$ such that there exists a containment mapping from $L(Q)$ to C. Let $S = \phi(C)$, the image of C in the body of Q'' under ϕ. Note that $C' = \kappa(S)$ is also a core of Q, since the composition of ϕ and κ is a containment mapping from $L(Q)$ to C'. It follows from uniqueness of the core up to isomorphism ([CM77]) that C and C' are isomorphic. We claim that ϕ is an isomorphism from C to S. By definition of S, every literal in S is in the image of ϕ, hence every variable in S is in the image of ϕ. Now suppose ϕ mapped two variables of C to the same variable in S. Since containment mappings cannot increase the number of variables, C' would have fewer variables than C, a contradiction. So ϕ is a bijection on the variables of C and S. By minimality of C and the existence of κ, S cannot have fewer literals than C, and by definition of S, S has no more literals than C. Hence S and C are isomorphic. To finish the proof we need to show that every database-relation literal in the body of Q' is in S. Suppose there is some database-relation literal in the body of Q' that is not in S; this literal can be removed from Q' while retaining equivalence to Q, contradicting the minimality of the rewriting. So S contains every database-relation literal in the body of Q', and since S is isomorphic to C, the database-relation literals in S are isomorphic to a subset of C.

To prove the second part, suppose that

$$Q' : q(\bar{X}) \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n), v_1(\bar{Y}_1), \ldots, v_k(\bar{Y}_k).$$

is a rewriting of Q. By Proposition 3.1, $\pi_\emptyset(q) \subseteq \pi_\emptyset(v_i)$ $(i = 1, \ldots, k)$. Therefore, there is a containment mapping h_i from the body of the rule defining v_i into the body of the original rule for q. Let $h_i(\bar{Y}_i) = \bar{Y}'_i$ $(i = 1, \ldots, k)$. Consider the query

$$Q'' : q(\bar{X}) \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n), v_1(\bar{Y}'_1), \ldots, v_k(\bar{Y}'_k).$$

It is easy to see that Q' contains Q'' (by using the mappings h_i), and clearly Q'' contains Q. Therefore, Q'' is equivalent to Q, and so it is a rewriting of Q using $\mathcal{V}$. Furthermore, Q'' does not introduce any new variables than those that appeared originally in Q.

The third part is proved in a similar fashion to the second except for one difference. Proposition 3.1 guarantees that for every v_i, there is a union of conjunctive queries $Q_i^1, \ldots, Q_i^{m_i}$ that is equivalent to Q, and there is a containment mapping h_i^j from v_i to every Q_i^j. The rewriting will be a disjunction of conjunctive queries. In every conjunct we choose one of the h_i^j's for every v_i, and construct the conjunct as in the previous case. $\Box$

The following example shows that the second part of the above lemma does not hold when the view contains built-in predicates.

Example 3.4: Consider the query:

$$Q : q(X, Y, U, W) \quad :- \quad p(X, Y), r(U, W), r(W, U).$$

and the view

$$V : v(A, B, C, D) \quad :- \quad p(A, B), r(C, D), C \leq D.$$

There exists no conjunctive query rewriting of Q that uses V and does not introduce new variables. However, the following is a rewriting of Q:

$$Q' : q(X, Y, U, W) \quad :- \quad v(X, Y, C, D), r(U, W), r(W, U).$$

Furthermore, the disjunctive rewriting that does not introduce new variables is:

$$\begin{aligned} Q' : q(X, Y, U, W) \quad &:- \quad v(X, Y, U, W), r(W, U). \\ Q' : q(X, Y, U, W) \quad &:- \quad v(X, Y, W, U), r(U, W). \end{aligned}$$

□

3.1 Finding Minimal Rewritings

In general, we are interested in using views to answer queries in order to reduce the cost of evaluating the query. In this section we consider the complexity of the problems of reducing the total number of literals in the rewriting, reducing the number of database-relations in the rewriting, and finding rewritings that use only the views. Finally, we show that the problem of finding minimal rewritings has two independent sources of complexity.

The following lemma is the basis for several results. It shows that a minimal rewriting of a query Q, using a set of views $\mathcal{V}$, does not *increase* the number of literals in the query.

Lemma 3.5: *Let Q be a conjunctive query and $\mathcal{V}$ be a set of views, both Q and $\mathcal{V}$ without built-in predicates. If the body of Q has p literals and Q' is a locally minimal and complete rewriting of Q using $\mathcal{V}$, then Q' has at most p literals.*

Note that we can always assume that there are views in $\mathcal{V}$ that are identical to the database-relations, and therefore this lemma entails that any locally minimal rewriting of Q will have at most p literals.

Proof: As before, let Q'' be the expansion of a rewriting Q' of Q, in which the view literals in Q' are replaced by their definitions. Consider the containment mapping ϕ from Q to Q''. Each literal $l_1, \ldots, l_p$ in the body of Q is mapped to the expansion of at most one view literal in the body of Q''. If there are more than p view literals in Q', the expansion of some view literal in the body of Q'' must be disjoint with the image of ϕ; but then this view literal can be removed from Q' while preserving equivalence with Q. Hence there is a rewriting with at most p view literals. □

In the full paper we show that the size of the rewriting is bounded even if the database relations may have functional dependencies, or if the query and views have built-in predicates. The following example shows that the bound of Proposition 3.5 does not hold when the database relations have functional dependencies.

Example 3.6: Consider the query

$$q(X, Y, Z) \quad :- \quad e(X, Y, Z).$$

and the views

$$\begin{aligned} v_1(X, Y) \quad &:- \quad e(X, Y, Z), \\ v_2(X, Z) \quad &:- \quad e(X, Y, Z). \end{aligned}$$

and suppose that in the relation e, the first argument functionally determines the other two. The following is the only complete rewriting of Q using v_1 and v_2:

$$q(X, Y, Z) \quad :- \quad v_1(X, Y), v_2(X, Z). \square$$

In the presence of functional dependencies, the size of a minimal rewriting is at most $p + d$ literals, where d is the sum of the arities of the literals in Q. In the presence of built-in predicates, the size of the rewritten query may be at most exponential in the size of Q.

Using Lemma 3.5, we obtain the following complexity results on finding minimal rewritings.

Theorem 3.7: *Let Q be a conjunctive query without built-in predicates and $\mathcal{V}$ be conjunctive views without built-in predicates.*

1. *The problem of whether there exists a rewriting Q' of Q using $\mathcal{V}$ such that Q' contains less than k literals, where k is less than the number of literals in the body of Q, is NP-complete.*

2. *The problem of whether there exists a rewriting Q' of Q using $\mathcal{V}$ such that Q' contains less than k literals of database relations, where k is less than the number of literals in the body of Q, is NP-complete.*

3. *The problem of whether there exists a complete rewriting of Q using $\mathcal{V}$ is NP-complete.*

4. *If the query Q and views $\mathcal{V}$ have built-in predicates, then Problem 1 is in Σ_3^p.*

Proof: The proof of the first part is as follows. The problem is in NP because, by the Lemmas 3.5 and 3.3, we need only consider rewritings that have no more literals than the query, have a subset of the literals of the query, and do not introduce new variables . We can guess such a rewritten query, verify that it contains less than k literals, and guess containment mappings from the original query to the rewritten one and vice-versa. For the NP-hardness, reduce the problem of existence of a usage to it as follows. Given a query Q and a view V, let V' be the rule whose head is the same as the head of V and whose body is the conjunction of the bodies of Q and V. Now there is a usage of V' in Q with 1 literal in it if and only if there is a usage of V in Q. The other parts of the theorem are proved in a similar fashion. □

Corollary 3.8: *The problem of finding a globally minimal rewriting of a conjunctive query without built-in predicates, using conjunctive views with no built-in predicates is CoNP-Complete. If the queries and views have built-in predicates then the problem is in Π_3^p.*

Using the results of [SY81] for unions of conjunctive queries and of [LS93] for unions of conjunctive queries with built-in predicates, we can generalize the above theorem as follows:

Theorem 3.9: *Let Q and $\mathcal{V}$ be disjunctions of conjunctive queries. When neither Q nor $\mathcal{V}$ have built-in predicates, the problem of whether there exists a complete rewriting of Q using $\mathcal{V}$ is NP-complete.*

The results described up to now suggest a two step algorithm for finding rewritings of a query Q. In the first step, we find some containment mapping from the views to the query and add to the query the appropriate view atoms, resulting in a query Q'. In the second step, we minimize Q' by removing literals from Q that are redundant. These two steps also emphasize the *two* sources of complexity involved in the problem. The first source is the exponential number of possible containment mappings from the views to the query. The second source is determining which literals of Q' are redundant given the mappings from the views to the query. The following theorem shows that these are two *independent* sources of complexity, i.e., that the problem is NP-complete even if there is a single mapping from each view to the query. In the next section we describe a polynomial time algorithm for determining which literals can be removed from the query, and we show that under certain conditions, it is guaranteed to find the unique maximal set of such literals.

Theorem 3.10: *The complete rewriting problem is NP-complete for conjunctive queries and views without built-in predicates even when both the query and the views are defined by rules that do not contain repeated predicates in their body.*

Note that when the query and the views are defined by such rules, then each rule is already non-redundant and, moreover, there is at most one mapping from each view into the query and finding those mappings is easy.

Proof: We use a reduction from the problem of exact cover by 3-sets. Given an instance of this problem, we create a predicate p_i for each element i and use a special variable S_j for each set j. For each p_i, we create an atom as follows. If element i is in set j, then the jth argument position of p_i has the variable S_j; if element i is not in set j, then the jth argument position of p_i has a distinct nondistinguished variable. The query is a conjunction of these atoms (i.e., one atom for each p_i). We may assume that the head of the query has no variables, i.e., it is of the form

$$q() \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n).$$

We also create views as follows. For each set j, we create a view v_j. The three subgoals of v_j are the atoms created for the elements that appear in set j. The variables in the head of v_j are all the S_k variables that appear in the body of v_j, except for S_j.

There is exactly one containment mapping from the body of each view into the body of the query. Hence, a minimal rewriting that uses the views will have a subset of the literals in the following query:

$$q() \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n), v_1(\bar{Y}_1), \ldots, v_m(\bar{Y}_m).$$

We have to show that there is a containment mapping that eliminates all the $p_i(\bar{U}_i)$ if and only if there is an exact cover. So, suppose that there is an exact cover. We will map each $p_i(\bar{U}_i)$ to the set that covers it. We have to show that the variables $S_1, \ldots, S_n$ are mapped consistently. So, suppose that two atoms $p_i(\bar{U}_i)$ and $p_j(\bar{U}_j)$ share the variable S_k. There are two cases to be considered. First, suppose that in the exact cover, the elements i and j are covered by the same set l. In this case, both of these atoms are mapped to the same view v_l, and clearly, the two occurrences of S_k in these atoms are mapped to the same variable in v_l. The second case is that elements i and j are covered by different sets, say h and l, respectively. Therefore, set k cannot be in the exact cover and, so, $k \neq h$ and $k \neq l$. It thus follows that S_k is a distinguished variable of both v_h and v_l, and hence, the two occurrences of S_k in $p_i(\bar{U}_i)$ and $p_j(\bar{U}_j)$ are mapped to S_k.

Now consider the other direction; that is, suppose that there is a containment mapping that eliminates all the $p_i(\bar{U}_i)$. Hence each $p_i(\bar{U}_i)$ is mapped to a view v_j, such that set j contains i. Since the variable S_j is not distinguished in v_j, it follows that if one $p_i(\bar{U}_i)$ is mapped to v_j, then so are the other two atoms for the elements of set j. Therefore, this mapping provides an exact cover. □

4 Finding Redundant Literals in the Rewritten Query

In the previous section we have shown that finding rewritings for a query using views can be done in two steps. In the first, we consider containment mappings from the bodies of the views to the body of the query, and add the appropriate view literals to the query. In the second step, we remove literals of the original query that are redundant. We have also shown that in general, both steps provide independent sources of exponential complexity.

In this section we describe a polynomial time algorithm for the second step. In particular, given a set of mappings from the views to the query, the algorithm determines which set of literals from the query can be removed. We show that under certain conditions there is a *unique* maximal set of such literals and that the algorithm is guaranteed to find them. In other cases, the algorithm may find only a subset of the redundant literals, but all the literals it removes are guaranteed to be redundant, and therefore the algorithm is always applicable. Note that in such cases, the rest of the query can still be minimized using known, more computationally expensive techniques. Together with an algorithm for enumerating mappings from the views to the query, our algorithm provides a practical method for finding rewritings. For simplicity, we describe the algorithm for the case of rewritings using a single occurrence of a view, and we begin with the case that does not include built-in predicates.

Formally, suppose our query is of the form

$$Q : q(\bar{X}) \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n). \qquad (1)$$

and we have the following view:

$$V : v(\bar{Z}) \quad :- \quad r_1(\bar{W}_1), \ldots, r_m(\bar{W}_m). \qquad (2)$$

Let h be a containment mapping from the body of v into the body of q, and let the following be the result of adding the view literal to the query:

$$q(\bar{X}) \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n), v(\bar{Y}). \qquad (3)$$

where $\bar{Y} = h(\bar{Z})$. Note that we can restrict ourselves to mappings where the variables of $\bar{Y}$ already appear in the $p_i(\bar{U}_i)$ (by Lemma 3.3). To obtain a minimal rewriting, our goal is to remove as many of the redundant p_i literals as possible.

To determine the set of redundant literals, consider the rule resulting from substituting the definition of Rule (2) instead of the view literal in Rule (3). That is, we rename the variables of Rule (2) as follows. Each variable T that appears in $\bar{Z}$ is renamed to $h(T)$, and each variable of Rule (2) that does not appear in $\bar{Z}$ is renamed to a new variable (that is not already among the $p_i(\bar{U}_i)$). Let the following be the result of this substitution.

$$q(\bar{X}) \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n), r_1(\bar{V}_1), \ldots, r_m(\bar{V}_m). \; (4)$$

Note that the variables of $\bar{Y}$ are the only ones that may appear in both the $p_i(\bar{U}_i)$ and the $r_j(\bar{V}_j)$.

Given the mapping h, there is a natural containment mapping from Rule (4) into the original rule for q (i.e., Rule (1)) that is defined as follows. Each literal $p_i(\bar{U}_i)$ is mapped to itself and each literal $r_j(\bar{V}_j)$ is mapped to the same literal of Rule (1) as in the containment mapping h (from Rule (2) to Rule (1)). We denote this containment mapping as ϕ. Note that the containment mapping ϕ maps each variable of $\bar{Y}$ to itself.

Each literal $p_i(\bar{U}_i)$ of Rule (1) is the image (under ϕ) of itself, and maybe a few of the $r_j(\bar{V}_j)$ literals. We say that the literals $r_j(\bar{V}_j)$ that map to $p_i(\bar{U}_i)$ under ϕ are the *associates* of $p_i(\bar{U}_i)$. For the rest of the discussion, we choose arbitrarily one of the associates of $p_i(\bar{U}_i)$ and refer to it as *the* associate of $p_i(\bar{U}_i)$. Note that if h does not map two literals $r_j(\bar{V}_j)$ to the same literal in Rule (1), then each $p_i(\bar{U}_i)$ will have at most one associate.

Before we show how to find the set of redundant literals, we need the following definition:

Definition 4.1 : *A literal* $r_j(\bar{V}_j)$ covers *a literal* $p_i(\bar{U}_i)$ *that has the same predicate if the following two conditions hold:*

- *If* $p_i(\bar{U}_i)$ *has a distinguished variable (i.e., a variable in* $\bar{X}$*) or a constant in some argument position* a*, then* $r_j(\bar{V}_j)$ *also has that variable (or the constant) in argument position* a*.*
- *If argument positions* a_1 *and* a_2 *of* $p_i(\bar{U}_i)$ *are equal, then so are the argument positions* a_1 *and* a_2 *of* $r_j(\bar{V}_j)$*.*

Intuitively, if $r_j(\bar{V}_j)$ is the associate of $p_i(\bar{U}_i)$ and does not cover $p_i(\bar{U}_i)$, then we cannot remove $p_i(\bar{U}_i)$, because $p_i(\bar{U}_i)$ enforces quality constraints that are not enforced by $r_j(\bar{V}_j)$.

The set of *needed* literals $\mathcal{N}$ of the query Q is defined below. The set of redundant literals is the complement of the set of needed literals.

Definition 4.2: *The set* $\mathcal{N}$ *is the minimal subset of literals in* Q *satisfying the following four conditions.*

1. *Literals that have no associate.*
2. *Literals that are not covered by their associates.*
3. *If all the following conditions hold, then* $p_i(\bar{U}_i)$ *is in* $\mathcal{N}$*:*
   - *Literal* $p_i(\bar{U}_i)$ *has the variable* T *in argument position* a_1*.*
   - *The associate of* $p_i(\bar{U}_i)$ *has the variable*[2] H *in argument position* a_1*.*

[2]Note that the associate of $p_i(\bar{U}_i)$ cannot have a constant in argument position a_1 if $p_i(\bar{U}_i)$ has a variable in that argument position.

- *The variable H is not in $\bar{Y}$ (hence, H appears only among the $r_j(\bar{V}_j)$).*
- *The variable T also appears in argument position a_2 of $p_l(\bar{U}_l)$.*
- *The associate of $p_l(\bar{U}_l)$ does not have H in argument position a_2.*

4. Suppose that $p_i(\bar{U}_i)$ is in $\mathcal{N}$ and that variable T appears in $p_i(\bar{U}_i)$. If $p_l(\bar{U}_l)$ has variable T in argument position a and its associate does not have T in argument position a, then $p_l(\bar{U}_l)$ is also in $\mathcal{N}$.

The third condition in the definition adds to $\mathcal{N}$ those literals in Q whose associates do not enforce the same join constraints. The fourth condition iteratively adds to $\mathcal{N}$ literals that are *connected* to a literal in $\mathcal{N}$ via a common variable. It is important to note that the set of needed variables can be found in polynomial time in the size of the query.

Example 4.3: Consider the query and the view of Example 2.2. The result of substituting the view in the query would be the following:

$$\begin{aligned} q(X,U) \quad :- \quad & p(X,Y), p_0(Y,Z), p_1(X,W), p_2(W,U), \\ & p(X,C), p_0(C,Z), p_1(X,D). \end{aligned}$$

The literal $p_2(W,U)$ is needed because it does not have an associate. The literal $p_1(X,W)$ is needed by the fourth condition of the definition, because its associate $p_1(X,D)$ does not contain the variable W (which appears in $p_2(W,U)$). Consequently, these two literals need to be retained to obtain the minimal rewriting. □

Theorem 4.4:

1. The query

$$q(\bar{X}) \quad :- \quad \mathcal{N}, v(\bar{Y}) \qquad (5)$$

is a rewriting of Q using V.

2. Suppose that h does not map two literals $r_j(\bar{V}_j)$ to the same literal in Rule (1), and Rule (1) is minimal. Then the maximal set of redundant $p_i(\bar{U}_i)$ in Rule (4) is unique and is exactly the complement of the set $\mathcal{N}$.

Proof: We will use ψ to denote a containment mapping from the original rule for q (i.e., Rule (1)) into the rewritten rule (i.e., Rule (4)).

Recall that the composition $\phi\psi$ is a containment mapping from Rule (1) to itself. Since Rule (1) is minimal, there is a k, such that $(\phi\psi)^k$ is the identity mapping on Rule (1). Let $\tau = \psi(\phi\psi)^{k-1}$. Note that τ is a containment mapping from Rule (1) into Rule (4), and $\phi\tau$ is the identity mapping on Rule (1).

The containment mapping ϕ (restricted to the image of τ) is the inverse of τ, since $\phi\tau$ is the identity mapping on Rule (1). Therefore, τ maps a literal $p_i(\bar{U}_i)$ of Rule (1) either to the literal $p_i(\bar{U}_i)$ or to the associate of $p_i(\bar{U}_i)$ in Rule (4).

We will now show that every $p_i(\bar{U}_i)$ in $\mathcal{N}$ must be mapped to itself by τ and, hence, all the $p_i(\bar{U}_i)$ of $\mathcal{N}$ are in the image of ψ. Recall that we already know that τ maps each $p_i(\bar{U}_i)$ either to itself or to its associate. If $p_i(\bar{U}_i)$ satisfies either Condition 1 or 2 (in the definition of $\mathcal{N}$), then clearly $p_i(\bar{U}_i)$ must be mapped to itself.

Suppose that $p_i(\bar{U}_i)$ and $p_l(\bar{U}_l)$ satisfy Condition 3. If $p_i(\bar{U}_i)$ is mapped to its associate, then $p_l(\bar{U}_l)$ must also be mapped to its associate, because variable H appears only among the $r_j(\bar{V}_j)$. But $p_i(\bar{U}_i)$ and $p_l(\bar{U}_l)$ cannot both be mapped to their associates, because $p_i(\bar{U}_i)$ and $p_l(\bar{U}_l)$ have the same variable T in argument positions a_1 and a_2, respectively, while their associates have different variables in these argument positions. Therefore, $p_i(\bar{U}_i)$ must be mapped to itself.

Now suppose that $p_i(\bar{U}_i)$ and $p_l(\bar{U}_l)$ satisfy Condition 4. Since $p_i(\bar{U}_i)$ is in $\mathcal{N}$, we may assume inductively that it must be mapped to itself. Therefore, variable T is mapped to itself and, hence, $p_l(\bar{U}_l)$ must also be mapped to itself. Thus, we have shown that all the literals of $\mathcal{N}$ must be mapped to themselves by τ.

We now define the mapping ψ' from Rule (1) into Rule (4) as follows. If $p_i(\bar{U}_i)$ is in $\mathcal{N}$, then it is mapped to itself; otherwise, it is mapped to its associate. We will show that ψ' is a containment mapping.

Clearly, every $p_i(\bar{U}_i)$ is mapped to a literal that covers it. So, it remains to show that if $p_i(\bar{U}_i)$ and $p_l(\bar{U}_l)$ have the same variable T in argument positions a_1 and a_2, respectively, then their images under ψ' also have the same symbol in these argument positions. There are three cases to be considered in order to prove this claim. In the first case, both $p_i(\bar{U}_i)$ and $p_l(\bar{U}_l)$ are mapped to themselves and the claim is clearly true.

In the second case, $p_i(\bar{U}_i)$ is mapped to itself (because it is in $\mathcal{N}$) while $p_l(\bar{U}_l)$ is mapped to its associate. By Condition 4 in the definition of $\mathcal{N}$, the associate of $p_l(\bar{U}_l)$ must also have variable T in argument position a_2 (or else $p_l(\bar{U}_l)$ would be in $\mathcal{N}$ and, hence, would be mapped to itself). So, the claim is proved also in this case.

In the third case, both $p_i(\bar{U}_i)$ and $p_l(\bar{U}_l)$ are mapped to their associates. Suppose that the associates have distinct variables, C and D, in argument positions a_1 and a_2, respectively. It is impossible that both C and D are in $\bar{Y}$, because ϕ is one-to-one on the variables of $\bar{Y}$ (because ϕ is the identity on $\bar{Y}$). So, one of them, say C, is not in $\bar{Y}$. But in this case, $p_i(\bar{U}_i)$ and $p_l(\bar{U}_l)$ satisfy Condition 3 in the definition of $\mathcal{N}$ and, hence, $p_i(\bar{U}_i)$ is in $\mathcal{N}$ and is mapped by ψ' to itself—a contradiction. Thus, we have shown that ψ' is a containment mapping.

In conclusion, we have shown that $\mathcal{N}$ is in the image of every containment mapping ψ from the original rule for q (i.e., Rule (1)) into the rewritten rule (i.e., Rule (4)). We have also shown that there is a mapping ψ', such that the $p_i(\bar{U}_i)$ in the image of ψ' are exactly those of $\mathcal{N}$. Therefore, the set of $p_i(\bar{U}_i)$ not in $\mathcal{N}$ is the unique maximal set of redundant $p_i(\bar{U}_i)$ in Rule (4). □

It is well known that a containment mapping can be found in polynomial time if each literal has at most two potential destinations; the exact algorithm is based on a reduction to the 2-SAT problem [SY81]. In some sense, this is the case in the minimization algorithm presented in Theorem 4.4, since each $p_i(\bar{U}_i)$ can be mapped either to itself or to its associate. However, the contribution of Theorem 4.4 is twofold. First, it shows that each $p_i(\bar{U}_i)$ has at most two destinations. This fact is not obvious (indeed, when ordinarily using the reduction to 2-SAT, each literal that covers $p_i(\bar{U}_i)$ is considered a potential destination of $p_i(\bar{U}_i)$). The second contribution of Theorem 4.4 is in providing a more direct (and, hence, likely to be more efficient) way of computing the redundant $p_i(\bar{U}_i)$, as compared to the algorithm that uses the reduction to 2-SAT.

Adding Built-in Predicates

When the views may have built-in predicates, we need to repeat a similar process of finding needed literals for several containment mappings, and we can remove only literals that are not deemed needed for *any* of the mapping. Formally, suppose the result of adding the view literal to the query is

$$Q' : q(\bar{X}) \quad :- \quad p_1(\bar{U}_1), \ldots, p_n(\bar{U}_n), v(\bar{Y}). \qquad (6)$$

As before, we can expand the definition of v in Q', obtaining the a conjunction Q'' (as in Rule 4). By Proposition 3.1, there are a set of queries $Q_1, \ldots, Q_m$, that differ only on the built-in predicates, such that:

- Q is equivalent to the union of $Q_1, \ldots, Q_m$, and
- For every i, $1 \leq i \leq m$, there is a containment mapping ϕ_i from the body of Q'' into the body of Q_i, such that $bi(Q_i)$ entails $\phi_i(bi(Q''))$.

For each one of the ϕ_i mappings we compute the set of needed literals $\mathcal{N}_i$, and we define

$$\mathcal{N} \;=\; \mathcal{N}_1 \cup \ldots \cup \mathcal{N}_m.$$

Only the literals in $\mathcal{N}$ from Q remain in the rewritten query.

Example 4.5: Consider the query from Example 3.4:

$$Q : q(X, Y, U, W) \quad :- \quad p(X, Y), r(U, W), r(W, U).$$

and the view

$$V : v(A, B, C, D) \quad :- \quad p(A, B), r(C, D), C \leq D.$$

The result of substituting the view in the query would be:

$$\begin{aligned} Q' : q(X, Y, U, W) \quad :- \quad & p(X, Y), r(U, W), r(W, U), \\ & v(X, Y, C, D). \end{aligned} \qquad (7)$$

The query Q can be written as the union

$$\begin{aligned} Q_1 : q(X, Y, U, W) \quad :- \quad & p(X, Y), r(U, W), r(W, U), \\ & U \leq W. \\ Q_2 : q(X, Y, U, W) \quad :- \quad & p(X, Y), r(U, W), r(W, U), \\ & U \geq W. \end{aligned}$$

and the mappings from the expansion of (7) to Q_1 and Q_2 are the identity on X, Y, U and W, and

$$\phi_1 : \{C \rightarrow U, D \rightarrow W\}.$$
$$\phi_2 : \{C \rightarrow W, D \rightarrow U\}.$$

For the mapping ϕ_1, we will deem only the literal $r(W, U)$ as needed, because it does not have an associate, and for ϕ_2, $r(U, W)$ will be deemed needed. Therefore, since the only literal that is not needed for either of the mappings is $p(X, Y)$, it can be removed, resulting in the following rewriting:

$$q(X, Y, U, W) \quad :- \quad r(U, W), r(W, U), v(X, Y, C, D).$$

□

5 Related Work

Several authors have considered the problem of implementing a query processor that uses the results of materialized views (e.g., [YL87, Sel88, SJGP90, CR94, TSI94, CKPS95]), but the formal aspects of finding the equivalent (and minimal) rewritings have received little attention.

Yang and Larson [YL87] considered the problem of finding rewritings for select-project-join queries and views. In their analysis they considered what amounts to one-to-one mappings from the views to query, and do not search the entire space of rewritings (and therefore may not always find all the possible rewritings of the query).

Chaudhuri et al. [CKPS95] considered the problem of finding rewritings for select-project-join queries and views, such that the rewritten query preserves the *bag* semantics. They show that in this case all the usages of views are obtained by 1-1 mappings from the views to the query, and therefore their algorithm would not find all the usages in the case where the relations are sets. Chaudhuri et al. [CKPS95] also considered the question of how to extend a query processor to chose between the different rewritings, a question that was not addressed in this paper. Dar et al. [DJLS95] recently extended the work in [CKPS95] to consider queries that involve aggregation. Finally, Rajaraman et al. [RSU95] built on our results and considered the problem of finding rewritings when the views may only be queried using specific binding patterns.

References

[BI94] Daniel Barbará and Tomasz Imieliński. Sleepers and workaholics: Caching strategies in mobile environments. In *Proceedings of SIGMOD-94*, pages 1–12, 1994.

[CKPS95] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *Proceedings of International Conference on Data Engineering,* 1995.

[CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[CR94] C. M. Chen and N. Roussopoulos. The implementation and performance evaluation of the ADMS query optimizer: Integrating query result caching and matching. In *Proceedings of the International Conference on Extending Database Technology*, 1994.

[CV92] Surajit Chaudhuri and Moshe Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *The Proceedings of the PODS–92*, pages 55–66, 1992.

[DJLS95] Shaul Dar, H.V. Jagadish, Alon Y. Levy and Divesh Srivastava. Answering SQL queries with aggregation using views. *AT&T Technical Memorandum*, 1995.

[GMR95] Ashish Gupta, Inderpal Singh Mumick, and Kenneth A. Ross. Adapting Materialized Views after Redefinitions. In *Proceedings of SIGMOD-95*, 1995.

[HSW94] Yixiu Huang, Prasad Sistla, and Ouri Wolfson. Data replication for mobile computers. In *Proceedings of SIGMOD-94*, pages 13–24, 1994.

[KB94] Arthur M. Keller and Julie Basu. A predicate-based caching scheme for client-server database architectures. In *Proceedings of PDIS-94*, 1994.

[LSK95] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 1995. Special Issue on Networked Information Discovery and Retrieval (to appear).

[LS93] Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *Proceedings of the 19th VLDB Conference, Dublin, Ireland*, pages 171–181, 1993.

[LY85] P. A. Larson and H.Z. Yang. Computing queries from derived relations. In *Proceedings of the 11th International VLDB Conference*, pages 259–269, 1985.

[RSU95] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering Queries Using Templates with Binding Patterns. In *Proceedings of the ACM Symposium on Principles of Database Systems*, San Jose, CA, May 1995.

[SJGP90] M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos. On rules, procedures, caching and views in database systems. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1990.

[SY81] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. In *J. ACM 27:4 pp. 633-655*, 1981.

[Sel88] Timos Sellis. Intelligent caching and indexing techniques for relational database systems. *Information Systems*, pages 175–185, 1988.

[Shm87] Oded Shmueli. Decidability and expressiveness aspects of logic queries. In *Proceedings of the Sixth Symposium on Principles of Database Systems (PODS)*, pages 237–249, San Diego, CA, March 1987.

[TSI94] Odysseas G. Tsatalos, Marvin H. Solomon, and Yannis E. Ioannidis. The GMAP: A versatile tool for physical data independence. In *Proceedings of VLDB-94*, pages 367–378, 1994.

[YL87] H. Z. Yang and P. A. Larson. Query transformation for PSJ-queries. In *Proceedings of the 13th International VLDB Conference*, pages 245–254, 1987.

[SY81] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. In *J. ACM 27:4 pp. 633-655*, 1981.

[vdM92] Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *The Proceedings of PODS–92*, pages 331–345, 1992.