Minimizing View Sets without Losing Query-Answering Power

Chen Li, Mayank Bawa, and Jeffrey D. Ullman

Computer Science Department, Stanford University, CA 94305, USA
{chenli,bawa,ullman}@db.stanford.edu

Abstract. The problem of answering queries using views has been studied extensively due to its relevance in a wide variety of data-management applications. In these applications, we often need to select a subset of views to maintain due to limited resources. In this paper, we show that traditional query containment is *not* a good basis for deciding whether or not a view should be selected. Instead, we should minimize the view set without losing its *query-answering power.* To formalize this notion, we first introduce the concept of "p-containment." That is, a view set $\mathcal{V}$ is *p-contained* in another view set $\mathcal{W}$, if $\mathcal{W}$ can answer all the queries that can be answered by $\mathcal{V}$. We show that p-containment and the traditional query containment are *not* related. We then discuss how to minimize a view set while retaining its query-answering power. We develop the idea further by considering p-containment of two view sets with respect to a given set of queries, and consider their relationship in terms of maximally-contained rewritings of queries using the views.

1 Introduction

The problem of answering queries using views [2,3,9,13,17,22] has been studied extensively, because of its relevance to a wide variety of data management problems, such as information integration, data warehousing, and query optimization. The problem can be stated as follows: given a query on a database schema and a set of views over the same schema, can we answer the query using only the answers to the views? Recently, Levy compiled a good survey [16] about the different approaches to this problem.

In the context of query optimization, computing a query using previously materialized views can speed up query processing, because part of the computation necessary for the query may have been done while computing the views. In a data warehouse, views can preclude costly access to the base relations and help answer queries quickly. In web-site designs, precomputed views can be used to improve the performance of web-sites [11]. Before choosing an optimal design, we must assure that the chosen views can be used to answer the expected queries at the web-site. A system that caches answers locally at the client can avoid accesses to base relations at the server. Cached result of a query can be thought of as a materialized view, with the query as its view definition. The client could use the cached answers from previous queries to answer future queries.

However, the benefits presented by views are not without costs. Materialized views often compete for limited resources. Thus, it is critical to select views carefully. For instance, in an information-integration system [24], a view may represent a set of web pages at an autonomous source. The mediator [26] in these systems often needs to crawl these web pages periodically to refresh the cached data in its local repository [8]. In such a scenario, the cost manifests itself as the bandwidth needed for such crawls and the efforts in maintaining the cache up-to-date. Correspondingly, in a query-optimization and database-design scenario, the materialized views may have part of the computation necessary for the query. When a user poses a query, we need to decide how to answer the query using the materialized views. By selecting an optimal subset of views to materialize, we can reduce the computation needed to decide how to answer typical queries. In a client-server architecture with client-side caching, storing all answers to past queries may need a large storage space and will add to the maintenance costs. Since the client needs to deal with an evolving set of queries, any of these can be used to answer future queries. Thus, redundant views need not be cached.

The following example shows that views can have redundancy to make such a minimization possible, and that traditional query containment is *not* a good basis for deciding whether a view should be selected or not. Instead, we should consider the *query-answering* power of the views.

Example 1. Suppose we have a client-server system with client-side caching for improving performance, since server data accesses are expensive. The server has the following base relation about books:

$$book(Title, Author, Pub, Price)$$

For example, the tuple $\langle$`databases`, `smith`, `prenhall`, \$`60`$\rangle$ in the relation means that a book titled `databases` has an author `smith`, is published by Prentice Hall (`prenhall`), and has a current price of \$60. Assume that the client has seen the following three queries, the answers of which have been cached locally. The cached data (or views), denoted by the view set $\mathcal{V} = \{V_1, V_2, V_3\}$, are:

V_1: $v_1(T, A, P)$:- $book(T, A, B, P)$
V_2: $v_2(T, A, P)$:- $book(T, A, prenhall, P)$
V_3: $v_3(A_1, A_2)$:- $book(T, A_1, prenhall, P_1), book(T, A_2, prenhall, P_2)$

The view V_1 has title-author-price information about all books in the relation, while the view V_2 includes this information only about books published by Prentice Hall. The view V_3 has coauthor pairs for books published by Prentice Hall. Since the view set has redundancy, we might want to eliminate a view to save costs of its maintenance and storage. At the same time, we want to be assured that such an elimination does not cause increased server accesses in response to future queries.

Clearly, view V_2 is contained in V_1, i.e., V_1 includes all the tuples in V_2, so we might be tempted to select $\{V_1, V_3\}$, and eliminate V_2 as a redundant view.

However, with this selection, we cannot answer the query:

$$Q_1 : q_1(T, P) \text{ :- } book(T, smith, prenhall, P)$$

which asks for titles and prices of books written by `smith` and published by `prenhall`. The reason is that even though V_1 includes title-author-price information about all books in the base relation, the publisher attribute is projected out in the view's head. Thus, using V_1 only, we cannot tell which books are published by `prenhall`. On the other hand, the query Q_1 can be answered trivially using V_2:

$$P_1 : q_1(T, P) \text{ :- } v_2(T, smith, P)$$

In other words, by dropping V_2 we have lost some *power* to answer queries. In addition, note that even though view V_3 is not contained in V_1 and V_2, it can be eliminated from $\mathcal{V}$ without changing the query-answering power of $\mathcal{V}$. The reason is that V_3 can be computed from V_2 as follows:

$$v_3(A_1, A_2) \text{ :- } v_2(T, A_1, P_1), v_2(T, A_2, P_2)$$

To summarize, we should not select $\{V_1, V_3\}$ but $\{V_1, V_2\}$, even though the former includes all the tuples in $\mathcal{V}$, while the latter does not. The rationale is that the latter is as "powerful" as $\mathcal{V}$ while the former is not. *Caution:* One might hypothesize from this example that only projections in view definitions cause such a mismatch, since we do not lose any "data" in the body of the view. We show in Section 3 that this hypothesis is wrong.

In this paper we discuss how to minimize a view set without losing its query-answering power. We first introduce the concept of *p-containment* between two view sets, where "p" stands for query-answering *power* (Sections 2 and 3). A view set $\mathcal{V}$ is *p-contained* in another view set $\mathcal{W}$, or $\mathcal{W}$ is *at least as powerful* as $\mathcal{V}$, if $\mathcal{W}$ can answer all the queries that can be answered using $\mathcal{V}$. Two view sets are called *equipotent* if they have the same power to answer queries. As shown in Example 1, two view sets may have the same tuples, yet have different query-answering power. That is, traditional view containment [6, 23] does not imply p-containment. The example further shows that the reverse direction is also *not* implied. In Section 3.2 we show that given a view set $\mathcal{V}$ on base relations, how to find a minimal subset of $\mathcal{V}$ that is equipotent to $\mathcal{V}$. As one might suspect, a view set can have multiple equipotent minimal subsets.

In some scenarios, users are restricted in the queries they can ask. In such cases, equipotence may be determined *relative* to the expected (possibly infinite) set of queries. In Section 4, we investigate the above questions of equipotence testing given this extra constraint. In particular, we consider infinite query sets defined by finite parameterized queries, and develop algorithms for testing this relative p-containment.

In information-integration systems, we often need to consider not only equivalent rewritings of a query using views, but also maximally-contained rewritings (MCR's). Analogous to p-containment, which requires equivalent rewritings, we introduce the concept of *MCR-containment* that is defined using maximally-contained rewritings (Section 5). Surprisingly, we show that p-containment implies MCR-containment, and vice-versa.

The containments between two finite sets of conjunctive views discussed in this paper are summarized in Table 1. In the full version of the paper [19] we discuss how to generalize the results to other languages, such as conjunctive queries with arithmetic comparisons, unions of conjunctive queries, and datalog.

Containment	**Definition**	**How to test**
v-containment $\mathcal{V} \sqsubseteq_v \mathcal{W}$	For any database, a tuple in a view in $\mathcal{V}$ is in a view in $\mathcal{W}$.	Check if each view in $\mathcal{V}$ is contained in some view in $\mathcal{W}$.
p-containment $\mathcal{V} \preceq_p \mathcal{W}$	If a query is answerable by $\mathcal{V}$, then it is answerable by $\mathcal{W}$.	Check if each view in $\mathcal{V}$ is answerable by $\mathcal{W}$.
relative p-containment $\mathcal{V} \preceq_{\mathcal{Q}} \mathcal{W}$	For each query Q in a given set of queries $\mathcal{Q}$, if Q is answerable by $\mathcal{V}$, then Q is answerable by $\mathcal{W}$.	Test by the definition if $\mathcal{Q}$ is finite. See Section 4.2 for infinite queries defined by parameterized queries.
MCR-containment $\mathcal{V} \preceq_{MCR} \mathcal{W}$	For each query Q, for any maximally-contained rewriting $MCR(Q, \mathcal{V})$ (resp. $MCR(Q, \mathcal{W})$) of Q using $\mathcal{V}$ (resp. $\mathcal{W}$), $MCR(Q, \mathcal{V}) \sqsubseteq MCR(Q, \mathcal{W})$.	Same as testing if $\mathcal{V} \preceq_p \mathcal{W}$, since $\mathcal{V} \preceq_p \mathcal{W} \Leftrightarrow \mathcal{V} \preceq_{MCR} \mathcal{W}$.

Table 1. Containments between two finite sets of conjunctive views: $\mathcal{V}$ and $\mathcal{W}$.

2 Background

In this section, we review some concepts about answering queries using views [17]. Let $r_1, \ldots, r_m$ be m *base relations* in a database. We first consider queries on the database in the following conjunctive form:

$$h(\bar{X}) \text{ :- } g_1(\bar{X}_1), \ldots, g_k(\bar{X}_k)$$

In each subgoal $g_i(\bar{X}_i)$, predicate g_i is a base relation, and every argument in the subgoal is either a variable or a constant. We consider views defined on the base relations by safe conjunctive queries, i.e., every variable in a query's head appears in the body. Note that we take the closed-world assumption [1], since the views are computed from existing database relations. We shall use names beginning with lower-case letters for constants and relations, and names beginning with upper-case letters for variables.

Definition 1. *(query containment and equivalence) A query Q_1 is* contained *in a query Q_2, denoted by $Q_1 \sqsubseteq Q_2$, if for any database D of the base relations, $Q_1(D) \subseteq Q_2(D)$. The two queries are* equivalent *if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$.*

Definition 2. *(expansion of a query using views) The* expansion *of a query P on a set of views $\mathcal{V}$, denoted by P^{exp}, is obtained from P by replacing all the views in P with their corresponding base relations. Existentially quantified variables in a view are replaced by fresh variables in P^{exp}.*

Definition 3. *(rewritings and equivalent rewritings) Given a query Q and a view set $\mathcal{V}$, a query P is a* rewriting *of query Q using $\mathcal{V}$ if P uses* only *the views in $\mathcal{V}$, and $P^{exp} \sqsubseteq Q$. P is an* equivalent rewriting *of Q using $\mathcal{V}$ if P^{exp} and Q are equivalent. We say a query Q is* answerable *by $\mathcal{V}$ if there exists an equivalent rewriting of Q using $\mathcal{V}$.*

In Example 1, P_1 is an equivalent rewriting of the query Q_1 using view V_2, because the expansion of P_1:

$$P_1^{exp} : q_1(T, P) \text{ :- } book(T, smith, prenhall, P)$$

is equivalent to Q_1. Thus, query Q_1 is answerable by V_2, but it is not answerable by $\{V_1, V_3\}$.

In this paper we consider *finite* view sets. Several algorithms have been developed for answering queries using views, such as the bucket algorithm [18, 12], the inverse-rule algorithm [22, 10], and the algorithms in [20, 21]. See [1, 17] for a study of the complexity of answering queries using views. In particular, it has been shown that the problem of rewriting a query using views is $\mathcal{NP}$-complete.

3 Comparing Query-Answering Power of View Sets

In this section we first introduce the concept of *p-containment*, and compare it with traditional query containment. Then we discuss how to minimize a view set without losing its query-answering power with respect to all possible queries.

Definition 4. *(p-containment and equipotence) A view set $\mathcal{V}$ is* p-contained *in another view set $\mathcal{W}$, or "$\mathcal{W}$ is at least as powerful as $\mathcal{V}$," denoted by $\mathcal{V} \preceq_p \mathcal{W}$, if any query answerable by $\mathcal{V}$ is also answerable by $\mathcal{W}$. Two view sets are* equipotent, *denoted by $\mathcal{V} \asymp_p \mathcal{W}$, if $\mathcal{V} \preceq_p \mathcal{W}$, and $\mathcal{W} \preceq_p \mathcal{V}$.*

In Example 1, the two view sets $\{V_1, V_2\}$ and $\{V_1, V_2, V_3\}$ are equipotent, since the latter can answer all the queries that can be answered by the former, and vice-versa. (We will give a formal proof shortly.) However, the two view sets, $\{V_1, V_3\}$ and $\{V_1, V_2, V_3\}$, are not equipotent, since the latter can answer the query Q_1, which cannot be answered by the former. The following lemma suggests an algorithm for testing p-containment.

Lemma 1. *Let $\mathcal{V}$ and $\mathcal{W}$ be two view sets. $\mathcal{V} \preceq_p \mathcal{W}$ iff for every view $V \in \mathcal{V}$, if treated as a query, V is answerable by $\mathcal{W}$.*[1]

[1] Due to space limitations, we do not provide all the proofs of the lemmas and theorems. Some proofs are given in the full version of the paper [19].

The importance of this lemma is that we can test $\mathcal{V} \preceq_p \mathcal{W}$ simply by checking if every view in $\mathcal{V}$ is answerable by $\mathcal{W}$. That is, we can just consider a *finite* set of queries, even though $\mathcal{V} \preceq_p \mathcal{W}$ means that $\mathcal{W}$ can answer *all* the infinite number of queries that can be answered by $\mathcal{V}$. We can use the algorithms in [10,12,18, 22] to do the checking. It is easy to see that the relationship "$\preceq_p$" is reflexive, antisymmetric, and transitive. Using the results of [17] for the complexity of testing whether a query is answerable by a set of views, we have:

Theorem 1. *The problem of whether a view set is p-contained in another view set is $\mathcal{NP}$-hard.*

Example 2. As we saw in Example 1, view V_3 is answerable by view V_2. By Lemma 1, we have $\{V_1, V_2, V_3\} \preceq_p \{V_1, V_2\}$. Clearly the other direction is also true, so $\{V_1, V_2\} \asymp_p \{V_1, V_2, V_3\}$. On the other hand, V_2 cannot be answered using $\{V_1, V_3\}$, which means $\{V_1, V_2, V_3\} \not\preceq_p \{V_1, V_3\}$.

3.1 Comparing p-containment and Traditional Query Containment

We are interested in the relationship between p-containment and the traditional concept of query containment (as in Definition 1). Before making the comparisons, we first generalize the latter to a concept called *v-containment* to cover the cases where the views in a set have different schemas.

Definition 5. *(v-containment and v-equivalence) A view set $\mathcal{V}$ is* v-contained *in another view set $\mathcal{W}$, denoted by $\mathcal{V} \sqsubseteq_v \mathcal{W}$, if the following holds. For any database D of the base relations, if tuple t is in $V(D)$ for a view $V \in \mathcal{V}$, then there is a view $W \in \mathcal{W}$, such that $t \in W(D)$. The two sets are* v-equivalent*, if $\mathcal{V} \sqsubseteq_v \mathcal{W}$, and $\mathcal{W} \sqsubseteq_v \mathcal{V}$.*

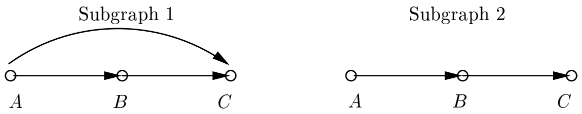
In Example 1, the two view sets $\{V_1, V_2, V_3\}$ and $\{V_1, V_3\}$ are v-equivalent, while their views have different schemas. The example shows that v-containment does not imply p-containment, and vice-versa. One might guess that if we do not allow projections in the view definitions (i.e., all the variables in the body of a view appear in the head), then v-containment could imply p-containment. However, the following example shows that this guess is incorrect.

Example 3. Let $e(X_1, X_2)$ be a base relation, where a tuple $e(x, y)$ means that there is an edge from vertex x to vertex y in a graph. Consider two view sets:

$$\begin{array}{l} \mathcal{V} = \{V_1\}, \quad V_1\text{: } v_1(A, B, C) \text{ :- } e(A, B), e(B, C), e(A, C) \\ \mathcal{W} = \{W_1\}, W_1\text{: } w_1(A, B, C) \text{ :- } e(A, B), e(B, C) \end{array}$$

As illustrated by Figure 1, view V_1 stores all the subgraphs shown in Figure 1(a), while view W_1 stores all the subgraphs shown in Figure 1(b). The two views do not have projections in their definitions, and $\mathcal{V} \sqsubseteq_v \mathcal{W}$. However, $\mathcal{V} \not\preceq_p \mathcal{W}$, since V_1 cannot be answered using W_1.

The following example shows that p-containment does not imply v-containment, even if the views in the sets have the same schemas.



(a) View V_1 (b) View W_1

Fig. 1. Diagram for the two views in Example 3

Example 4. Let $r(X_1, X_2)$ and $s(Y_1, Y_2)$ be two base relations on which two view sets are defined:

$$\begin{array}{lll} \mathcal{V} = \{V_1\}, & V_1\colon\ v_1(A, C) & \text{:-}\ r(A, B), s(B, C) \\ \mathcal{W} = \{W_1, W_2\}, & W_1\colon w_1(A, B) & \text{:-}\ r(A, B) \\ & W_2\colon w_2(B, C) & \text{:-}\ s(B, C) \end{array}$$

Clearly $\mathcal{V} \not\sqsubseteq_v \mathcal{W}$, but $\mathcal{V} \preceq_p \mathcal{W}$, since there is a rewriting of V_1 using $\mathcal{W}$: $v_1(A, C)$:- $w_1(A, B), w_2(B, C)$.

3.2 Finding an Equipotent Minimal Subset

In many applications, each view is associated with a cost, such as its storage space or the number of web pages that need to be crawled for the view [8]. We often need to find a subset of views that has the same query-answering power.

Definition 6. *(equipotent minimal subsets) A subset M of a view set $\mathcal{V}$ is an* equipotent minimal subset *(EMS for short) of $\mathcal{V}$ if $M \asymp_p \mathcal{V}$, and for any $V \in M : M - \{V\} \not\asymp_p \mathcal{V}$.*

Informally, an equipotent minimal subset of a view set $\mathcal{V}$ is a minimal subset that is as powerful as $\mathcal{V}$. For instance, in Example 1, the view set $\{V_1, V_2\}$ is an EMS of $\{V_1, V_2, V_3\}$. We can compute an EMS of a view set $\mathcal{V}$ using the following Shrinking algorithm.

> Algorithm Shrinking initially sets $M = \mathcal{V}$. For each view $V \in M$, it checks if V is answerable by the views $M - \{V\}$. If so, it removes V from M. It repeats this process until no more views can be removed from M, and returns the resulting M as an EMS of $\mathcal{V}$.

Example 5. This example shows that, as suspected, a view set may have multiple EMS's. Suppose $r(A, B)$ is a base relation, on which the following three views are defined:

$$\begin{array}{ll} V_1\colon v_1(A) & \text{:-}\ r(A, B) \\ V_2\colon v_2(B) & \text{:-}\ r(A, B) \\ V_3\colon v_3(A, B) & \text{:-}\ r(A, X), r(Y, B) \end{array}$$

Let $\mathcal{V} = \{V_1, V_2, V_3\}$. Then $\mathcal{V}$ has two EMS's: $\{V_1, V_2\}$, and $\{V_3\}$, as shown by the following rewritings:

$$\begin{array}{lll} \text{rewrite } V_1 \text{ using } V_3\text{:} & v_1(A) & \text{:- } v_3(A, B) \\ \text{rewrite } V_2 \text{ using } V_3\text{:} & v_2(B) & \text{:- } v_3(A, B) \\ \text{rewrite } V_3 \text{ using } \{V_1, V_2\}\text{:} & v_3(A, B) & \text{:- } v_1(A), v_2(B) \end{array}$$

We often want to find an EMS such that the total cost of selected views is minimum. We believe that the problem of finding an optimal EMS efficiently deserves more investigations.

4 Testing p-containment Relative to a Query Set

Till now, we have considered p-containment between two view sets with respect to a "universal" set of queries, i.e., users can ask *any* query on the base relations. However, in some scenarios, users are restricted in the queries they can ask. In this section, we consider the relationship between two view sets with respect to a given set of queries. In particular, we consider infinite query sets defined by finite parameterized queries.

Definition 7. *(relative p-containment) Given a (possibly infinite) set of queries $\mathcal{Q}$, a view set $\mathcal{V}$ is p-contained in a view set $\mathcal{W}$ w.r.t. $\mathcal{Q}$, denoted by $\mathcal{V} \preceq_{\mathcal{Q}} \mathcal{W}$, iff for any query $Q \in \mathcal{Q}$ that is answerable by $\mathcal{V}$, Q is also answerable by $\mathcal{W}$. The two view sets are equipotent w.r.t. $\mathcal{Q}$, denoted by $\mathcal{V} \asymp_{\mathcal{Q}} \mathcal{W}$, if $\mathcal{V} \preceq_{\mathcal{Q}} \mathcal{W}$ and $\mathcal{W} \preceq_{\mathcal{Q}} \mathcal{V}$.*

Example 6. Assume we have relations $car(Make, Dealer)$ and $loc(Dealer, City)$ that store information about cars, their dealers, and the cities where the dealers are located. Consider the following two queries and three views:

$$\begin{array}{llll} \text{Queries:} & Q_1\text{:} & q_1(D, C) & \text{:- } car(toyota, D), loc(D, C) \\ & Q_2\text{:} & q_2(D, C) & \text{:- } car(honda, D), loc(D, C) \\ \text{Views:} & W_1\text{:} & w_1(D, C) & \text{:- } car(toyota, D), loc(D, C) \\ & W_2\text{:} & w_2(D, C) & \text{:- } car(honda, D), loc(D, C) \\ & W_3\text{:} & w_3(M, D, C) & \text{:- } car(M, D), loc(D, C) \end{array}$$

Let $\mathcal{Q} = \{Q_1, Q_2\}$, $\mathcal{V} = \{W_1, W_2\}$, and $\mathcal{W} = \{W_3\}$. Then $\mathcal{V}$ and $\mathcal{W}$ are equipotent w.r.t. $\mathcal{Q}$, since Q_1 and Q_2 can be answered by $\mathcal{V}$ as well as $\mathcal{W}$. Note that $\mathcal{V}$ and $\mathcal{W}$ are not equipotent in general.

Given a view set $\mathcal{V}$ and a query set $\mathcal{Q}$, we define an *equipotent minimal subset* (EMS) of $\mathcal{V}$ w.r.t. $\mathcal{Q}$ in the same manner as Definition 6. We can compute an EMS of $\mathcal{V}$ w.r.t. $\mathcal{Q}$ in the same way as in Section 3.2, if we have a method to test relative p-containment. This testing is straightforward when $\mathcal{Q}$ is finite; i.e., by definition, we can check for each query $Q_i \in \mathcal{Q}$ that is answerable by $\mathcal{V}$, whether Q_i is also answerable by $\mathcal{W}$. However, if $\mathcal{Q}$ is infinite, testing relative p-containment becomes more challenging, since we cannot use this enumerate-and-test paradigm for all the queries in $\mathcal{Q}$. In the rest of this section we consider ways to test relative p-containment w.r.t. infinite query sets defined by finite parameterized queries.

4.1 Parameterized Queries

A *parameterized query* is a conjunctive query that contains *placeholders* in the argument positions of its body, in addition to constants and variables. A placeholder is denoted by an argument name beginning with a "$" sign.

Example 7. Consider the following parameterized query Q on the two relations in Example 6:

$$Q : q(D) \text{ :- } car(\$M, D), loc(D, \$C)$$

This query represents all the following queries: a user gives a car make m for the placeholder $\$M$, and a city c for the placeholder $\$C$, and asks for the dealers of the make m in the city c. For example, the following are two instances of Q:

$$\begin{array}{l} I_1\text{: } q(D) \text{ :- } car(toyota, D), loc(D, sf) \\ I_2\text{: } q(D) \text{ :- } car(honda, D), loc(D, sf) \end{array}$$

which respectively ask for dealers of Toyota and Honda in San Francisco (sf).

In general, each *instance* of a parameterized query Q is obtained by assigning a constant from the corresponding domain to each placeholder. If a placeholder appears in different argument positions, then the same constant must be used in these positions. Let $IS(Q)$ (resp. $IS(\mathcal{Q})$) denote the set of all instances of the query Q (resp. a query set $\mathcal{Q}$). We assume that the domains of placeholders are infinite (independent of an instance of the base relations), causing $IS(Q)$ to be infinite. Thus we can represent an infinite set of queries using a finite set of parameterized queries.

Example 8. Consider the following three views:

$$\begin{array}{lll} V_1\text{: } v_1(M, D, C) & \text{:- } car(M, D), loc(D, C) \\ V_2\text{: } v_2(M, D) & \text{:- } car(M, D), loc(D, sf) \\ V_3\text{: } v_3(M) & \text{:- } car(M, D), loc(D, sf) \end{array}$$

Clearly, view V_1 can answer all instances of Q in Example 7, since it includes information for cars and dealers in all cities. View V_2 cannot answer all instances, since it has only the information about dealers in San Francisco. But it can answer instances of the following more restrictive parameterized query, which replaces the placeholder $\$C$ by sf:

$$Q' : q(D) \text{ :- } car(\$M, D), loc(D, sf)$$

That is, the user can only ask for information about dealers in San Francisco. Finally, view V_3 cannot answer any instance of Q, since it does not have the *Dealer* attribute in its head.

Given a finite set of parameterized queries $\mathcal{Q}$ and two view sets $\mathcal{V}$ and $\mathcal{W}$, the example above suggests the following strategy of testing $\mathcal{V} \preceq_{IS(\mathcal{Q})} \mathcal{W}$:

1. Deduce *all* instances of $\mathcal{Q}$ that can be answered by $\mathcal{V}$.

2. Test if $\mathcal{W}$ can answer *all* such instances.

In the next two subsections we show how to perform each of these steps. We show that all answerable instances of a parameterized query for a given view set can be represented by a *finite* set of parameterized queries. We give an algorithm for deducing this set, and an algorithm for the second step. Although our discussion is based on one parameterized query, the results can be easily generalized to a finite set of parameterized queries.

4.2 Complete Answerability of a Parameterized Query

We first consider the problem of testing whether all instances of a parameterized query Q can be answered by a view set $\mathcal{V}$. If so, we say that Q is *completely answerable* by $\mathcal{V}$.

Definition 8. *(canonical instance) A* canonical instance *of a parameterized query Q (given a view set $\mathcal{V}$) is an instance of Q, in which each placeholder is replaced by a new distinct constant that does not appear in Q and $\mathcal{V}$.*

Lemma 2. *A parameterized query Q is completely answerable by a view set $\mathcal{V}$ if and only if $\mathcal{V}$ can answer a canonical instance of Q (given $\mathcal{V}$).*

The lemma suggests an algorithm TestComp for testing whether all instances of a parameterized query Q can be answered by a view set $\mathcal{V}$.

> Algorithm TestComp first constructs a canonical instance Q_c of Q (given $\mathcal{V}$). Then it tests if Q_c can be answered using $\mathcal{V}$ by calling an algorithm of answering queries using views, such as those in [10, 12, 18, 22]. It outputs "yes" if $\mathcal{V}$ can answer Q_c; otherwise, it outputs "no."

Example 9. Consider the parameterized query Q in Example 8. To test whether view V_1 can answer all instances of Q, we use two new distinct constants m_0 and c_0 to replace the two placeholders $\$M$ and $\$C$, and obtain the following canonical instance:

$$Q_c : q(D) \text{ :- } car(m_0, D), loc(D, c_0)$$

Clearly Q_c can be answered by view V_1, because of the following equivalent rewriting of Q_c:

$$P_c : q(D) \text{ :- } v_1(m_0, D, c_0)$$

By Lemma 2, view V_1 can answer all instances of Q. In addition, since V_2 cannot answer Q_c (which is also a canonical instance of Q given V_2), it cannot answer *some* instances of Q. The same argument holds for V_3.

4.3 Partial Answerability of a Parameterized Query

As shown by view V_2 and query Q in Example 8, even if a view set cannot answer all instances of a parameterized query, it can still answer some instances. In general, we want to know what instances can be answered by the view set, and whether these instances can also be represented as a set of more "restrictive" parameterized queries. A parameterized query Q_1 is more *restrictive* than a parameterized query Q if every instance of Q_1 is also an instance of Q. For example, query $q(D)$:- $car(\$M, D), loc(D, sf)$ is more restrictive than query $q(D)$:- $car(\$M, D), loc(D, \$C)$, since the former requires the second argument of the loc subgoal to be sf, while the latter allows any constant for placeholder $\$C$. For another example, query $q(M, C)$:- $car(M, \$D_1), loc(\$D_1, C)$ is more restrictive than query $q(M, C)$:- $car(M, \$D_1), loc(\$D_2, C)$, since the former has one placeholder in two argument positions, while the latter allows two different constants to be assigned to its two placeholders.

All the parameterized queries that are more restrictive than Q can be generated by adding the following two types of restrictions:

1. Type I: Some placeholders must be assigned the same constant. Formally, let $\{\$A_1, \ldots, \$A_k\}$ be *some* placeholders in Q. We can put a restriction $\$A_1 = \cdots = \$A_k$ on the query Q. That is, we can replace all these k placeholders with any of them.
2. Type II: For a placeholder $\$A_i$ in Q and a constant c in Q or $\mathcal{V}$, we put a restriction $\$A_i = c$ on Q. That is, the user can only assign constant c to this placeholder in an instance.

Consider all the possible (finite) combinations of these two types of restrictions. For example, suppose Q has two placeholders, $\{\$A_1, \$A_2\}$, and Q and $\mathcal{V}$ have one constant c. Then we consider the following restriction combinations: $\{\}$, $\{\$A_1 = \$A_2\}$, $\{\$A_1 = c\}$, $\{\$A_2 = c\}$, and $\{\$A_1 = \$A_2 = c\}$. Note that we allow a combination to have restrictions of only one type. In addition, each restriction combination is consistent, in the sense that it does not have a restriction $\$A_1 = \$A_2$ and two restrictions $\$A_1 = c_1$ and $\$A_2 = c_2$, while c_1 and c_2 are two different constants in Q and $\mathcal{V}$. For each restriction combination RC_i, let $Q(RC_i)$ be the parameterized query that is derived by adding the restrictions in RC_i to Q. Clearly $Q(RC_i)$ is a parameterized query that is more restrictive than Q. Let $\Phi(Q, \mathcal{V})$ denote all these more restrictive parameterized queries.

Suppose I is an instance of Q that can be answered by $\mathcal{V}$. We can show that there exists a parameterized query $Q_i \in \Phi(Q, \mathcal{V})$, such that I is a canonical instance of Q_i. By Lemma 2, Q_i is completely answerable by $\mathcal{V}$. Therefore, we have proved the following theorem:

Theorem 2. *All instances of a parameterized query Q that are answerable by a view set $\mathcal{V}$ can be generated by a* finite *set of parameterized queries that are more restrictive than Q, such that all these parameterized queries are completely answerable by $\mathcal{V}$.*

We propose the following algorithm GenPartial. Given a parameterized query Q and a view set $\mathcal{V}$, the algorithm generates all the parameterized queries that are more restrictive than Q, such that they are completely answerable by $\mathcal{V}$, and they define *all* the instances of Q that are answerable by $\mathcal{V}$.

> Algorithm GenPartial first generates all the restriction combinations, and creates a parameterized query for each combination. Then it calls the algorithm TestComp to check if this parameterized query is completely answerable by $\mathcal{V}$. It outputs all the parameterized queries that are completely answerable by $\mathcal{V}$.

4.4 Testing p-containment Relative to Finite Parameterized Queries

Now we give an algorithm for testing p-containment relative to parameterized queries. Let $\mathcal{Q}$ be a query set with only one parameterized query Q. Let $\mathcal{V}$ and $\mathcal{W}$ be two view sets. The algorithm tests $\mathcal{V} \preceq_{IS(\mathcal{Q})} \mathcal{W}$ as follows. First call the algorithm GenPartial to find all the more restrictive parameterized queries of Q that are completely answerable by $\mathcal{V}$. For each of them, call the algorithm TestComp to check if it is also completely answerable by $\mathcal{W}$. By Theorem 2, $\mathcal{V} \preceq_{IS(\mathcal{Q})} \mathcal{W}$ iff all these parameterized queries that are completely answerable by $\mathcal{V}$ are also completely answerable by $\mathcal{W}$. The algorithm can be easily generalized to the case where $\mathcal{Q}$ is a finite set of parameterized queries.

5 MCR-containment

So far we have considered query-answering power of views with respect to equivalent rewritings of queries. In information-integration systems, we often need to consider maximally-contained rewritings. In this section, we introduce the concept of *MCR-containment*, which describes the relative power of two view sets in terms of their maximally-contained rewritings of queries. Surprisingly, MCR-containment is essentially the same as p-containment.

Definition 9. *(maximally-contained rewritings) A* maximally-contained rewriting *P of a query Q using a view set $\mathcal{V}$ satisfies the following conditions: (1) P is a finite union of conjunctive queries using only the views in $\mathcal{V}$; (2) For any database, the answer computed by P is a subset of the answer to Q; and (3) No other unions of conjunctive queries that satisfy the two conditions above can properly contain P.*

Intuitively, a maximally-contained rewriting (henceforth "MCR" for short) is a plan that uses only views in $\mathcal{V}$ and computes the maximal answer to query Q. If Q has two MCR's, by definition, they must be equivalent as queries. If Q is answerable by $\mathcal{V}$, then an equivalent rewriting of Q using $\mathcal{V}$ is also an MCR of Q.

Example 10. Consider the following query Q and view V on the two relations in Example 6:

$$Q\colon q(M,D,C) \text{ :- } car(M,D), loc(D,C)$$
$$V\colon v(M,D,sf) \text{ :- } car(M,D), loc(D,sf)$$

Suppose we have the access to view V only. Then $q(M,D,sf)$:- $v(M,D,sf)$ is an MCR of the query Q using the view V. That is, we can give the user only the information about car dealers in San Francisco as an answer to the query, but not anything more.

Definition 10. *(MCR-containment) A view set $\mathcal{V}$ is* MCR-contained *in another view set $\mathcal{W}$, denoted by $\mathcal{V} \preceq_{MCR} \mathcal{W}$, if for any query Q, we have $MCR(Q,\mathcal{V}) \sqsubseteq MCR(Q,\mathcal{W})$, where $MCR(Q,\mathcal{V})$ and $MCR(Q,\mathcal{W})$ are MCR's of Q using $\mathcal{V}$ and $\mathcal{W}$, respectively.*[2] *The two sets are* MCR-equipotent, *denoted by $\mathcal{V} \asymp_{MCR} \mathcal{W}$, if $\mathcal{V} \preceq_{MCR} \mathcal{W}$, and $\mathcal{W} \preceq_{MCR} \mathcal{V}$.*

Surprisingly, MCR-containment is essentially the same as p-containment.

Theorem 3. *For two view sets $\mathcal{V}$ and $\mathcal{W}$, $\mathcal{V} \preceq_p \mathcal{W}$ if and only if $\mathcal{V} \preceq_{MCR} \mathcal{W}$.*

Proof. "If": Suppose $\mathcal{V} \preceq_{MCR} \mathcal{W}$. Consider each view $V \in \mathcal{V}$. Clearly V itself is an MCR of the query V using $\mathcal{V}$, since it is an equivalent rewriting of V. Let $MCR(V,\mathcal{W})$ be an MCR of V using $\mathcal{W}$. Since $\mathcal{V} \preceq_{MCR} \mathcal{W}$, we have $V \sqsubseteq MCR(V,\mathcal{W})$. On the other hand, by the definition of MCR's, $MCR(V,\mathcal{W}) \sqsubseteq V$. Thus $MCR(V,\mathcal{W})$ and V are equivalent, and $MCR(V,\mathcal{W})$ is an equivalent rewriting of V using $\mathcal{W}$. By Lemma 1, $\mathcal{V} \preceq_p \mathcal{W}$.

"Only if": Suppose $\mathcal{V} \preceq_p \mathcal{W}$. By Lemma 1, every view has an equivalent rewriting using $\mathcal{W}$. For any query Q, let $MCR(Q,\mathcal{V})$ and $MCR(Q,\mathcal{W})$ be MCR's of Q using $\mathcal{V}$ and $\mathcal{W}$, respectively. We replace each view in $MCR(Q,\mathcal{V})$ with its corresponding rewriting using $\mathcal{W}$, and obtain a new rewriting MCR' of query Q using $\mathcal{W}$, which is equivalent to $MCR(Q,\mathcal{V})$. By the definition of MCR's, we have $MCR' \sqsubseteq MCR(Q,\mathcal{W})$. Thus $MCR(Q,\mathcal{V}) \sqsubseteq MCR(Q,\mathcal{W})$, and $\mathcal{V} \preceq_{MCR} \mathcal{W}$.

6 Related Work

There has been a lot of work on the problem of selection of views to materialize in a data warehouse. In [5, 14, 15, 25], a data warehouse is modeled as a repository of integrated information available for querying and analysis. A subset of queries are materialized to improve responsiveness, and base relations are accessed for the rest of the queries. Base relations can change over time, and the query results can be huge, resulting in costs for maintenance and space. The study in this setting has, therefore, emphasized on modeling the view-selection problem as cost-benefit analysis. For a given set of queries, various sets of sub-queries are considered for materialization. Redundant views in a set that increase costs are deduced using query containment, and an optimal subset is chosen.

Such a model is feasible when all queries can be answered in the worst case by accessing base relations, and not by views alone. This assumption is incorporated

[2] We extend the query-containment notation "$\sqsubseteq$" in Definition 1 to unions of conjunctive queries in the obvious way.

in the model by replicating base relations at the warehouse. Thus, the base relations themselves are considered to be normalized, independent, and minimal. However, when real-time access to base relations is prohibitive, such an approach can lead to wrong conclusions, as was seen in Example 1. In such a scenario, it is essential to ensure the *computability* of queries using *only* maintained views.

Our work is directed towards scenarios where the following assumptions hold. (1) Real-time access to base relations is prohibitive, or possibly denied, and (2) cached views are expensive to maintain over time, because of the high costs of propagating changes from base relations to views. Therefore, while minimizing a view set, it is important to retain its *query-answering* power. We believe the power of answering queries and the benefit/costs of a view set are orthogonal issues, and their interplay would make an interesting work in its own right.

The term "query-answering" has been used in [4] to mean deducing tuples that satisfy a query, given the view definitions and their extensions. In our framework, this term stands for the ability of a set to answer queries. Another related work is [13] that studies information content of views. It develops a concept *subsumption* between two sets of queries, which is used to characterize their capabilities of distinguishing two instances of a database.

Recently, [7] has proposed solutions to the following problem: given a set of queries on base relations, which views do we need to materialize in order to decrease the query answering time? The authors show that even for conjunctive queries and views only, there can be an infinite number of views that can answer the same query. At the same time, the authors show that the problem is decidable: for conjunctive queries and views, it is enough to consider a finite space of views where all views are superior, in terms of storage space and query answering time, to any other views that could answer the given queries. The problem specification in that paper is different from ours: they start with a set of given queries and no views. In our framework, we assume that a set of views are given, and queries can be arbitrary. We would like to deduce a minimal subset of views that can answer *all* queries answerable by the original set.

Currently we are working on some open problems in our framework, including ways to find an optimal EMS of a view set efficiently, to find a v-equivalent minimal subset of a view set efficiently, and to find cases where v-containment can imply p-containment, and vice-versa.

Acknowledgments: We thank Arvind Arasu for his valuable comments.

References

1. S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *PODS*, pages 254–263, 1998.
2. F. N. Afrati, M. Gergatsoulis, and T. G. Kavalieros. Answering queries using materialized views with disjunctions. In *ICDT*, pages 435–452, 1999.
3. D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Query answering using views for data integration over the Web. *WebDB*, pages 73–78, 1999.
4. D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. What is view-based query rewriting. In *KRDB*, 2000.

5. S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *Proc. of VLDB*, 1991.
6. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. *STOC*, pages 77–90, 1977.
7. R. Chirkova and M. R. Genesereth. Linearly bounded reformulations of conjunctive databases. *DOOD*, 2000.
8. J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. *SIGMOD*, 2000.
9. O. M. Duschka. Query planning and optimization in information integration. *Ph.D. Thesis, Computer Science Dept., Stanford Univ.*, 1997.
10. O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997.
11. D. Florescu, A. Y. Levy, D. Suciu, and K. Yagoub. Optimization of run-time management of data intensive web-sites. In *Proc. of VLDB*, pages 627–638, 1999.
12. G. Grahne and A. O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *ICDT*, pages 332–347, 1999.
13. S. Grumbach and L. Tininini. On the content of materialized aggregate views. In *PODS*, pages 47–57, 2000.
14. J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge. The Stanford Data Warehousing Project. In *IEEE Data Eng. Bulletin, Special Issue on Materialized Views and Data Warehousing*, 1995.
15. W. H. Inmon and C. Kelley. *Rdb/VMS: Developing the Data Warehouse.* QED Publishing Group, Boston, Massachussetts, 1993.
16. A. Levy. Answering queries using views: A survey. *Technical report, Computer Science Dept., Washington Univ.*, 2000.
17. A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.
18. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, 1996.
19. C. Li, M. Bawa, and J. D. Ullman. Minimizing view sets without losing query-answering power (extended version). *Technical report, Computer Science Dept., Stanford Univ.,* `http://dbpubs.stanford.edu:8090/pub/2001-1`, 2000.
20. P. Mitra. An algorithm for answering queries efficiently using views. In *Technical Report, Stanford University*, 1999.
21. R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. In *Proc. of VLDB*, 2000.
22. X. Qian. Query folding. In *ICDE*, pages 48–55, 1996.
23. Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.
24. J. D. Ullman. Information integration using logical views. In *ICDT*, pages 19–40, 1997.
25. J. Widom. Research problems in data warehousing. In *Proc. of the Intl. Conf. on Information and Knowledge Management*, 1995.
26. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.