Materializing Views with Minimal Size To Answer Queries

Rada Chirkova
Computer Science Department
North Carolina State University
Campus Box 7535, Raleigh, NC 27695-7535
chirkova@csc.ncsu.edu

Chen Li
Information and Computer Science
University of California, Irvine
Irvine, CA 92697
chenli@ics.uci.edu

ABSTRACT

In this paper we study the following problem. Given a database and a set of queries, we want to find, in advance, a set of views that can compute the answers to the queries, such that the size of the viewset (i.e., the amount of space, in bytes, required to store the viewset) is minimal on the given database. This problem is important for many applications such as distributed databases, data warehousing, and data integration. We explore the decidability and complexity of the problem for workloads of conjunctive queries. We show that results differ significantly depending on whether the workload queries have self-joins. If queries can have self-joins, then a disjunctive viewset can be a better solution than any set of conjunctive views. We show that the problem of finding a minimal-size disjunctive viewset is decidable, and give an upper bound on its complexity. If workload queries cannot have self-joins, there is no need to consider disjunctive viewsets, and we show that the problem is in NP. We describe a very compact search space of conjunctive views, which contains all views in at least one optimal disjunctive viewset. We give a dynamic-programming algorithm for finding minimal-size disjunctive viewsets for queries without self-joins, and discuss heuristics to make the algorithm efficient.

1. INTRODUCTION

In this paper we study the following problem: given a set of queries, how to choose views to compute the answers to the queries, such that the total size of the viewset (i.e., the amount of space, in bytes, required to store the viewset) is minimal. This problem exists in many environments, such as distributed databases [5, 8, 26], data integration [22], and the recent "database-as-a-service" model [19]. For example, mediators in data-integration systems support seamless access to autonomous, heterogeneous information sources [34]. A mediator translates a given user query to a sequence of queries on the sources, and then uses the answers from the sources to compute the final answer to the user query [17]. After receiving many user queries, the mediator can send multiple queries to the same source to receive data. As another example, "database as a service" is a new model for enterprise computing [18], in which companies and organizations choose storing their data on a server over having to maintain local databases. The server provides client users with the power to create, store, modify, and query data on the server. When a client issues a query, the server uses the stored data to compute the answer and sends the results to the user over the network.

These applications share the following characteristics. (1) Both the client and the server are able to do computation. Notice that the client might prefer computing some part of the query answer to receiving excessively large amounts of data from the server, as in the database-as-a-service scenario; in the mediation scenario, the client (mediator) might have to do computation anyway. (2) The computation is data driven; the data resides on the server that is different from the client where a query is issued — either by client's choice, as in the database-as-a-service scenario, or by design, as in the mediation scenario. (3) The server needs to send data to the client over a network. When query results are large, the network could become a bottleneck, and the client may want to minimize the costs of transferring the data over the network.

For instance, consider the following simplified versions of three relation schemas in the TPC-H benchmark [32]:

```
customer(ckey, name, mktsegment)
order(okey, ckey, priority, comment)
lineitem(okey, partkey, quantity)
```

Suppose the relations are stored on a server. A client user issues the query Q in Figure 1. This query is the natural join of the three stored relations, with the selection condition `customer.mktsegment='building'`. The server then computes the answer to Q and sends it back to the client. Notice that the query answer could have a lot of redundancy; for example, an order may have many lineitems, thus the order information will appear many times in the answer.

To reduce the redundancy, we can instead decompose the answer into intermediate results — *views* V_1 and V_2, shown in Figure 1. The views have the same subgoals as the query, but have different output attributes. After receiving the results of the views, the client can compute the answer using the rewriting in Figure 1. One advantage of sending the view results is that their total size could be much smaller than that of the query answer. Another advantage is that if the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODS 2003, June 9-12, 2003, San Diego, CA.
Copyright 2003 ACM 1-58113-670-6/03/06 ...$5.00.

Query Q: $ans(name, okey, priority, comment, partkey, quantity)$:- $customer(ckey, name, \text{'building'})$, $order(okey, ckey, priority, comment), lineitem(okey, partkey, quantity)$.

View V_1: $V_1(name, okey, priority, comment)$:- $customer(ckey, name, \text{'building'})$, $order(okey, ckey, priority, comment), lineitem(okey, partkey, quantity)$.

View V_2: $V_2(okey, partkey, quantity)$:- $customer(ckey, name, \text{'building'})$, $order(okey, ckey, priority, comment), lineitem(okey, partkey, quantity)$.

Rewriting: $ans(name, okey, priority, comment, partkey, quantity)$:- $V_1(name, okey, priority, comment), V_2(okey, partkey, quantity)$.

Figure 1: Using views to answer a query.

client has cached the answers to previously asked queries, which can be used to compute the results of a view, say V_1, then we do not need to send the results of V_1. In this way, we can further reduce the communication costs. Clearly there are other ways to decompose the query answer into views, and which viewset has the smallest total size depends on the database instance.

In general, given a set of queries (a *query workload*) and a fixed database instance, we want to define and precompute offline a set of intermediate results (views), such that these view results can be used to compute the answer to each query in the workload. In addition, we want to choose the views in such a way that their total size is minimal on the given database. In this paper we study this problem for select-project-join queries with equality selections, also known as conjunctive queries [33]. We explore the decidability and complexity of the problem. We show that the results differ significantly depending on whether the queries have self-joins.

After formulating the problem in Section 2, we present the following contributions.

1. In Section 3 we study the decidability and complexity of the problem. We show that if workload queries have self-joins, nontrivial disjunctive views can give rise to smaller viewsets than purely conjunctive views. We establish that the problem of finding a minimal-size viewset in the space of disjunctive views is decidable, and give an upper bound on the complexity of the problem. Further, we show that for arbitrary conjunctive query workloads, to find rewritings of the workload queries in terms of a minimal-size viewset, it is not necessary to consider nontrivial disjunctive rewritings.

2. In Section 4 we study workloads of conjunctive queries without self-joins, and show that disjunctive views cannot provide smaller viewsets than purely conjunctive views. Thus it is enough to consider purely conjunctive views when looking for a minimal-size disjunctive viewset. Moreover, it is enough to explore a very restricted search space of such views, and the problem of finding a minimal-size viewset is in NP.

3. In Section 5 we present a dynamic-programming algorithm for finding minimal-size disjunctive viewsets for conjunctive queries without self-joins, and give heuristics to improve the algorithm.

1.1 Related work

The problem of finding views to materialize to answer queries has traditionally been studied under the name of view selection. Its original motivation comes up in the context of data warehousing. The problem is to decide which views to store in the warehouse to obtain optimal performance [15, 30, 31, 35]; one direction is to materialize views and indexes for data cubes in online analytic processing (OLAP) [3, 16, 21]. Another motivation for view selection is provided by recent versions of several commercial database systems. These systems support incremental updates of materialized views and are able to use materialized views to speed up query evaluation [4, 14, 37]. Choosing an appropriate set of views to materialize in the database is crucial in order to obtain performance benefits from these new features [2].

Traditional work on view selection uses certain critical tacit assumptions. The first assumption is that the only views to be considered to materialize are those that are subexpressions of the given queries, or are given in the input to the problem in some other way. The second assumption is that there is some low upper bound on the number of views in an optimal viewset. These assumptions have been questioned in recent work on database restructuring [11, 12, 13], which considers all possible views that can be invented to optimize a given metric of database performance.

Other related topics include answering queries using views (e.g., [1, 23, 20]), view-based query answering (e.g., [6, 7]), minimizing viewsets without losing query-answering power [24], and using compression techniques to reduce query-result size [10].

2. PROBLEM FORMULATION

Given a set, or *workload*, $\mathcal{Q}$ of queries on stored relations $R_1, \ldots, R_n$ and a fixed database instance, we want to find and precompute offline a set of intermediate results, defined as views $V_1, \ldots, V_k$ on these relations. These views can be used to compute the answers to all queries in the workload $\mathcal{Q}$. Our goal is to find an *optimal solution* — to choose a set of views $\mathcal{V}$, such that their total size $\Sigma_{V_i \in \mathcal{V}}\ size(V_i)$ is minimal on the given database instance. The size of a view V_i is the amount of space, in bytes, required to store the view V_i. In addition to finding the views, we also find a plan to compute the answer to each query in the workload $\mathcal{Q}$ using the views in $\mathcal{V}$.

We focus on conjunctive queries (i.e., select-project-join queries with equality selections). Each query is of the form

$$ans(\bar{X}) \text{ :- } R_1(\bar{X}_1), \ldots, R_n(\bar{X}_n).$$

Predicate R_i in a subgoal $R_i(\bar{X}_i)$ corresponds to a base (stored) relation, and each argument in the subgoal is either a variable or a constant. We consider views defined on the base relations by safe conjunctive or disjunctive queries. A *disjunctive query* is a union of conjunctive queries. A

query is *safe* if each variable in the query's head appears in the body. A query variable is called *distinguished* if it appears in the query's head. We consider query rewritings that are either conjunctions of views (*conjunctive* query rewritings) or unions of conjunctions of views (*disjunctive* query rewritings), under set semantics.

2.1 Answering queries using views

We briefly review some important concepts of answering queries using views. (See [23] for details.) A query Q_1 is *contained* in a query Q_2, denoted $Q_1 \sqsubseteq Q_2$, if for any database D of the base relations, the answer computed by Q_1 is a subset of the answer by Q_2. The two queries are *equivalent* if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$. A conjunctive query Q_1 is contained in a conjunctive query Q_2 if and only if there is a *containment mapping* from Q_2 to Q_1 [9]. The *expansion* of a query P on a set of views $\mathcal{V}$, denoted P^{exp}, is obtained from P by replacing all views in P by their definitions in terms of the base relations. Given a query Q and a set of views $\mathcal{V}$, a query P is an *equivalent rewriting* of Q using $\mathcal{V}$ if P uses only the views in $\mathcal{V}$ and P^{exp} is equivalent to Q. In the rest of the paper, we use "rewriting" to mean "equivalent rewriting."

EXAMPLE 2.1. *Consider two relations:* `r(Dealer, Make)` *and* `s(Dealer,City)`*. A tuple* `r(d,m)` *means that dealer* `d` *sells a car of make* `m`*. A tuple* `s(d,c)` *means that dealer* `d` *is located in city* `c`*. Consider the following query Q and three views V_1, V_2, and V_3. The query asks for all pairs* `(m,c)`*, such that there is a dealer in city* `c` *selling cars of make* `m`*.*

$$\begin{array}{lll} Q: & ans(M,C) & \text{:- } r(D,M), s(D,C). \\ V_1: & ans(D,M) & \text{:- } r(D,M). \\ V_2: & ans(D,C) & \text{:- } r(D,M), s(D,C). \\ V_3: & ans(D) & \text{:- } r(D,M), s(D,C). \end{array}$$

P is an equivalent rewriting of Q using the three views:

$$P: ans(M,C) \text{ :- } V_3(D), V_1(D,M), V_2(D,C).$$

We can show there are two containment mappings: one from Q to the expansion P^{exp} of P — this mapping is an identity mapping — and another from P^{exp} to Q. □

2.2 Two types of views in rewritings

There are two types of views in a rewriting of a query: (1) containment-target views, and (2) filtering views. They can be distinguished by examining containment mappings from the query to the expansion of the rewriting. Intuitively, in a rewriting, a *containment-target view* "covers" at least one query subgoal. That is, in the computation of the query using the rewriting, the view provides at least one query subgoal. Covering all query subgoals is enough to produce a rewriting of the query. For instance, in Example 2.1, view $V_1(D,M)$ covers the query subgoal $r(D,M)$, whereas view $V_2(D,C)$ covers the query subgoal $s(D,C)$. Thus views V_1 and V_2 are containment-target views for the query.

DEFINITION 2.1. *(***Containment-target view***) A conjunctive view V is a* containment-target view *for a query Q if the following is true. There exists a rewriting P of Q (P uses V), and there is a containment mapping from Q to the expansion P^{exp} of P, such that V provides the image of at least one subgoal of Q under the mapping.* □

A view is a *filtering view* for a query if it is not a containment-target view. Filtering views are not necessary in constructing query rewritings, in the sense that those views do not cover query subgoals. However, there could exist some query plan in which a filtering view removes, or *filters out*, dangling tuples from some join input(s) in the plan, which may reduce the cost of evaluating the query. For instance, view V_3 in Example 2.1 is a filtering view, which removes from the view V_1 all tuples that are dangling with respect to the view V_2. We will show that to solve the problem of minimizing the size of a viewset that gives a rewriting of a query workload, we do not need to consider filtering views.

3. DECIDABILITY AND COMPLEXITY

In this section we study the decidability and complexity of the problem of finding a minimal-size viewset for a workload of conjunctive queries.

3.1 Minimal-size sets of conjunctive views: The problem is decidable

We start by obtaining results on the decidability and complexity of finding a minimal-size *conjunctive* viewset for a set of conjunctive queries, assuming *conjunctive* rewritings.

THEOREM 3.1. *For any finite workload of conjunctive queries and a database instance, it is possible to construct a finite search space of views, which includes all views in all minimal-size conjunctive viewsets for the workload. The number of views in the search space is at most doubly-exponential in the length of the longest query definition in the workload.* □

The idea of the proof is as follows. Suppose a viewset $\mathcal{V}$ is a minimal-size viewset for a workload $\mathcal{Q}$ on a database instance D, and suppose some views in $\mathcal{V}$ have definitions whose length is more than exponential in the length of the longest query definition in the workload $\mathcal{Q}$. Let $size(\mathcal{V})$ be the total size of the views in $\mathcal{V}$. By Theorem 3.1 in [13], using the query workload $\mathcal{Q}$ and the storage limit equal to $size(\mathcal{V})$, we can construct another viewset $\mathcal{W}$ with two properties: (1) each view in $\mathcal{W}$ has a definition whose length is at most exponential in the length of the longest query definition in the workload $\mathcal{Q}$; (2) the viewset $\mathcal{W}$ satisfies the storage limit $size(\mathcal{V})$ on the database D. By construction, the viewset $\mathcal{W}$ is also a minimal-size viewset for the query workload $\mathcal{Q}$ on the database D. Because the length of each view definition in any such viewset $\mathcal{W}$ is at most exponential in the length of the longest query definition in Q, the total number of views in all viewsets $\mathcal{W}$ is at most doubly-exponential in the length of the longest query definition in the workload.

From Theorem 3.1 we obtain a decidability result:

COROLLARY 3.1. *Given a database instance, the problem of finding a minimal-size conjunctive viewset is decidable for finite workloads of conjunctive queries, assuming all rewritings are conjunctive.* □

We also observe that the problem has a triply-exponential upper bound: A naive algorithm will find a minimal-size viewset for a given query workload by exploring all subsets of the at most doubly-exponential search space of views.

3.2 Containment-target views are enough

We now show that when looking for a minimal-size viewset (either conjunctive or disjunctive) for a conjunctive query workload, we only need to consider containment-target views, assuming all rewritings are conjunctive. In addition, there is a linear upper bound on the number of views in any such minimal-size viewset.

LEMMA 3.1. *Given a database instance, for any conjunctive query workload $\mathcal{Q}$ and for any minimal-size disjunctive viewset $\mathcal{V}$ for $\mathcal{Q}$ (assuming conjunctive rewritings), each view in $\mathcal{V}$ is a containment-target view for at least one query in the workload $\mathcal{Q}$.* □

Intuitively, in looking for a viewset that is a solution for a query workload, we want to minimize the number of views as a way to minimize the total size of the viewset. In particular, we want to minimize the number of containment-target views. They are the only type of views needed to produce rewritings of each workload query by covering subsets of the query subgoals. We also observe that in a minimal-size viewset, there is no need to cover any query subgoal by more than one containment-target view. From these observations we obtain the following result.

THEOREM 3.2. *Given a database instance, for any conjunctive query workload $\mathcal{Q}$ and for any minimal-size disjunctive viewset $\mathcal{V}$ for $\mathcal{Q}$ (assuming conjunctive rewritings), if the queries in the workload $\mathcal{Q}$ have a total of n subgoals, then the viewset $\mathcal{V}$ has at most n views.* □

It has been shown [23] that for any conjunctive query with n subgoals, if the query has a rewriting using views, then there exists a rewriting with at most n views. At the same time, that result did not provide any optimality guarantees for the views in the rewriting.

Even though Lemma 3.1 says that in searching for a minimal-size viewset, we can restrict our consideration to containment-target views only, the search space of views for a query workload can still be very large, even if we examine conjunctive views only. There are mainly two reasons: (1) there are many ways to choose subsets of the query subgoals; and (2) there are many ways to project out variables in a view definition. The following example illustrates the point.

EXAMPLE 3.1. *For an integer value $k \geq 1$, consider a query workload $\{Q_k\}$, where:*

$$Q_k : ans(X, Y_1, \ldots, Y_k) \text{ :- } p_1(X, Y_1), \ldots, p_k(X, Y_k).$$

We can define at least the following containment-target views for $\{Q_k\}$. Each view is defined as a subset of the subgoals of Q_k, and all variables in each view are distinguished. It is easy to see that by taking conjunctions of some of these views, we can obtain many different rewritings of Q_k. Further, the total number N of these containment-target views is $N = 2^k - 1$. Notice that the space can be larger if we also consider views with nondistinguished variables. □

OBSERVATION 1. *Given a database instance, for the problem of finding a minimal-size viewset for a workload of conjunctive queries, the size of the search space of views can be (at least) exponential in the length of the definitions of the queries.* □

3.3 Disjunctive views can provide better solutions

A query has a *self-join* if the minimized query definition [9] has at least two subgoals with the same relation name. When workload queries have self-joins, we show that it might be better to materialize disjunctive views than conjunctive views.

PROPOSITION 3.1. *There exists a query workload and a database instance, such that a solution with disjunctive views requires strictly less storage space than any solution using conjunctive views only.* □

We give the proof by constructing such an example. Let *flight(source,destination)* be a base table, in which a tuple $flight(s, t)$ means that there is a direct flight from city s to city t. A query workload $\{Q\}$ has a single query Q that asks for all sequences of airports that give one-stop flights:

$$Q : ans(X, Y, Z) \text{ :- } flight(X, Y), flight(Y, Z).$$

We define a disjunctive view $V = V_1 \cup V_2$, where

V_1: $ans(X, Y)$:- $flight(X, Y), flight(Y, Z)$.
V_2: $ans(Y, Z)$:- $flight(X, Y), flight(Y, Z)$.

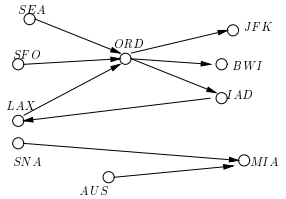

Figure 2: A database of the "flight" relation.

For the database in Figure 2, we show that the disjunctive viewset $\{V\}$ is smaller than any conjunctive viewset. Let $\mathcal{W}$ be any optimal *conjunctive* solution for the workload. By Lemma 3.1, the viewset $\mathcal{W}$ consists of conjunctive containment-target views only. We show that for the given database instance, the solution $\mathcal{W}$ has more data values than the disjunctive solution $\{V\}$.

Let P be an equivalent rewriting of the query Q using the views in $\mathcal{W}$.

$$P : ans(A, B, C) \text{ :- } w_1(\bar{X}_1), \ldots, w_k(\bar{X}_k).$$

By Lemma 3.1, P has no more than two view literals, i.e., $k = 1$ or 2. There are two cases: (1) each view w_i is used exactly once in the rewriting P; (2) some view w_j is used more than once in P. We consider the two cases separately.

Case (1): each view w_i is used exactly once in rewriting P. From Theorem 3 in [12], each view in P can be defined as a subexpression of the query Q. It follows that all views in the rewriting P come from a set S of all conjunctive containment-target views that can be defined as subexpressions of the query Q. The set S for the given Q has just four views. By considering all combinations $\mathcal{W}$ of the views in the set S that produce equivalent rewritings and by computing the sizes of the resulting viewsets, we can show that the disjunctive solution $\{V\}$ has fewer data values than any such viewset $\mathcal{W}$.

Case (2): some conjunctive view w_j can be used more than once in the rewriting P. Because P has at most two subgoals, the only possibility in this case is that P is a self-join of a single conjunctive view:

$$P : ans(A, B, C) \text{ :- } w(\bar{X}_1), w(\bar{X}_2).$$

By Theorem 3.1 in [13], in this case views can have more subgoals than the query Q. Suppose there exists a conjunctive view w that has more subgoals (after minimization) than query Q. Then the minimized definition of w has at least three subgoals of the *flight* relation. In addition, the definition of w cannot have cross-products (otherwise the solution $\{w\}$ would be suboptimal). By considering all possible combinations of nontrivial join predicates on three *flight* subgoals, we verify that the three subgoals in w cannot contain any subset of subgoals of the query Q. In assuming that a view can be used in the rewriting P and have more subgoals than Q, we have arrived at a contradiction. On the other hand, suppose the view w is a subexpression of the query Q (i.e., $w \in S$). Then for any nontrivial equivalent rewriting of the query Q that is a self-join of w, the viewset $\{w\}$, on the given database instance, has more data values than the disjunctive solution $\{V\}$.

In summary, for both cases we have shown that on the given instance of relation *flight*, an optimal conjunctive viewset requires strictly more storage space than the disjunctive viewset $\{V\}$. Therefore, the disjunctive viewset $\{V\}$ is smaller than any conjunctive viewset.

3.4 Disjunctive views and rewritings: The problem is decidable

In this section we show that if we allow disjunctive views in viewsets, then the problem of finding an optimal solution for a workload of conjunctive queries is still decidable, even if we also allow disjunctions in query rewritings. Let a *minimal-size disjunctive viewset* for a workload of conjunctive queries be a minimal-size viewset for the workload in the space of disjunctive views. Any or all of its disjunctive views can be purely conjunctive.

THEOREM 3.3. *Given a database instance and a finite workload of conjunctive queries (assuming conjunctive rewritings only), we can construct a finite search space of views that includes all views in all minimal-size disjunctive viewsets for the query. The number of views in the search space is at most triply-exponential in the sum of sizes of the definitions of the workload queries.* □

The main idea of the proof is as follows. For any conjunctive query, to obtain all nontrivial disjunctive views in all minimal-size disjunctive viewsets, it is enough to consider all minimal-size *conjunctive* rewritings of the query. For each such rewriting, it is enough to generate all *disjunctive* views whose conjuncts are subsets of subgoals in the expansion of the rewriting. From Theorem 3.1, we know that the problem of generating all minimal-size conjunctive rewritings of a conjunctive query is decidable. Thus the problem of finding all minimal-size disjunctive viewsets for a conjunctive query is also decidable. We then generalize our observations to finite workloads of conjunctive queries. The upper bound on the complexity of the problem follows from the fact that to find all optimal disjunctive views, it is enough to consider unions of optimal conjunctive views.

We now give the details of the proof. Consider a conjunctive query workload $\mathcal{Q}$ and a disjunctive viewset $\mathcal{V}$ that is a solution for the workload. In this proof, we look at two cases: we first consider all singleton query workloads $\mathcal{Q}$, and then look at all other query workloads $\mathcal{Q}$.

Case 1: Suppose the query workload $\mathcal{Q}$ has just one query q, i.e., $\mathcal{Q} = \{Q\}$, and $\mathcal{V}$ is a (not necessarily optimal) disjunctive solution for $\mathcal{Q}$. Consider a rewriting of the query Q using $\mathcal{V}$, and consider the expansion of the rewriting, which is a union of several conjunctive queries. One of these queries, P, is equivalent to the query Q [28]. We call P the *equivalent conjunct* of Q in terms of $\mathcal{V}$.

Suppose the viewset $\mathcal{V}$ contains a nontrivial disjunctive view V. Let V be a union of conjunctive views $V_1, \ldots, V_l$. Consider an alternative solution $\mathcal{V}'$ that is obtained by replacing, in $\mathcal{V}$, this disjunctive view V by $V_1, \ldots, V_l$. The viewset $\mathcal{V}'$ is a solution for the query Q because of the conjunct P. We say the viewset $\mathcal{V}$ is a *better solution* for the query Q than $\mathcal{V}'$ if $size(\mathcal{V}) < size(\mathcal{V}')$.

$size(\mathcal{V})$ can be less than $size(\mathcal{V}')$ only when the equivalent conjunct P (of Q in terms of $\mathcal{V}$) has a *self-join* of (at least) two *different* conjuncts, V_i and V_j, of the disjunctive view V. (If P has either just one conjunct V_k of the disjunctive view V, or if P has a self-join of just one conjunct V_m of V, we could obtain a solution for the query Q that is at least as good as $\mathcal{V}$, by replacing, in the viewset $\mathcal{V}$, the view V by just one of its conjuncts — V_k or V_m, respectively.) Note that each of V_k and V_m is a purely conjunctive view.

Next, we show that we can reconstruct the disjunctive view V using just the equivalent conjunctive rewriting P of Q, without having access to the disjunctive viewset $\mathcal{V}$. In the first step, consider any viewset $\mathcal{W}$ that consists of purely *conjunctive* views and such that P is an expansion of an equivalent rewriting P' of the query Q using the viewset $\mathcal{W}$. All such viewsets $\mathcal{W}$ can be generated just from P, for the following reason. Each view W in $\mathcal{W}$ is defined from a subset of subgoals of P; the head arguments of W are any subset of the set of all arguments of W that also occur either in other views in the rewriting P' or in the head of P'. Once we have all subgoals of P partitioned into the bodies of views W, a conjunction of the views is an equivalent rewriting of the query Q, and P is the expansion of the rewriting. We conclude that given an expansion P of the query Q, we can generate all purely conjunctive viewsets that are solutions for Q and have the expansion P.

In the second step, we notice that for any conjunctive query Q, we can construct all expansions P of all optimal equivalent conjunctive rewritings of the query. (This result follows immediately from Theorem 3.1.) Notice that for each optimal equivalent conjunctive rewriting of the query, the length of the expansion of the rewriting is at most exponential in the length of the query. (From Lemma 3.1, the rewriting has at most as many views as the query has subgoals; the length of the definition of each view is at most exponential in the length of the query definition.)

To summarize, we have looked at a conjunctive query Q, a nontrivial disjunctive view V in a solution for Q, an equivalent conjunct P in the expansion of the equivalent rewriting of Q, and a set of conjunctive views $V_1, \ldots, V_l$ which are all conjuncts of the view V. We have seen that the view V requires less storage space than $V_1, \ldots, V_l$ only when P has a self-join of at least two different conjuncts, V_i, and V_j, of the view V. Now we are ready to make the third and

last step of the proof. For an arbitrary conjunctive query Q, we can generate all optimal equivalent conjuncts P and all conjunctive views for those conjuncts; there are a finite number of such conjuncts and of such views. It follows that we can generate all conjuncts $V_1, \ldots, V_l$ of all nontrivial disjunctive views V that give rise to the equivalent rewriting P of the query Q. Observe that each such view V can be generated by taking a union of two or more such conjuncts, and that the process of generating all such views V is finite because the number of views $V_1, \ldots, V_l$ is finite. From this observation, we obtain the claim for Case 1 of the theorem.

Case 2. Suppose the query workload $\mathcal{Q}$ has at least two queries: $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$, $n \geq 2$. In addition to finding all disjunctive views that can be used to rewrite individual queries in the workload $\mathcal{Q}$, we now want to account for each nontrivial disjunctive view that can be used to equivalently rewrite *more than one* query in $\mathcal{Q}$. To achieve this goal, all we have to do is to replace, in the reasoning for Case 1 above, P (the equivalent conjunct of the only query Q in the workload in Case 1) by a conjunction $\mathcal{P}$, which we obtain as follows:

- we take an equivalent conjunct P_i of each query Q_i ($i \in \{1, \ldots, n\}$) in the workload $\mathcal{Q}$,
- if necessary, we rename the variables in the conjuncts $P_1, \ldots, P_n$, to avoid using any variable name in more than one conjunct,
- finally, we take a conjunction $\mathcal{P}$ of all these conjuncts: the body of $\mathcal{P}$ is $P_1 \,\&\ldots\&\, P_n$, and the head of $\mathcal{P}$ comprises all head variables of $P_1, \ldots, P_n$.

By applying the reasoning in Case 1 to the individual conjuncts P_i in $\mathcal{P}$, we can easily show that the number of subgoals in $\mathcal{P}$ is at most singly-exponential in the size of the query workload $\mathcal{Q}$.

For any minimal-size disjunctive viewset $\mathcal{V}$ for the workload $\mathcal{Q}$ (assuming conjunctive rewritings only), it is easy to see that $\mathcal{V}$ is also a minimal-size disjunctive viewset for the conjunction $\mathcal{P}$. Using the reasoning in Case 1 above, we can find all disjunctive viewsets $\mathcal{W}$ that can equivalently rewrite the conjunction $\mathcal{P}$; the complexity bounds are the same as in Case 1. All that remains to be done is to find those viewsets among $\mathcal{W}$ that can be used to equivalently rewrite all individual queries in the workload $\mathcal{Q}$. This observation concludes the proof of case 2 and the proof of the theorem.

3.5 Disjunctive rewritings are not needed

We now show that it is not necessary to consider nontrivial disjunctive rewritings of query workloads in terms of minimal-size disjunctive viewsets. Purely conjunctive rewritings are all we need to examine.

THEOREM 3.4. *Given a database instance, let $\mathcal{Q}$ be an arbitrary finite workload of conjunctive queries, and let $\mathcal{V}$ be any minimal-size disjunctive viewset, such that $\mathcal{V}$ gives an equivalent* disjunctive *rewriting of each query in the workload $\mathcal{Q}$. Then for each query in the workload $\mathcal{Q}$, there exists an equivalent* conjunctive *rewriting of the query in terms of the views in $\mathcal{V}$.* □

PROOF. Consider an arbitrary finite workload $\mathcal{Q}$ of conjunctive queries. Let $\mathcal{V}$ be any minimal-size disjunctive viewset, such that $\mathcal{V}$ gives an equivalent *disjunctive* rewriting of each query in the workload $\mathcal{Q}$. For each query Q in $\mathcal{Q}$, consider an equivalent *disjunctive* rewriting P of Q in terms of the views $\mathcal{V}$. The query P is a union of conjunctive queries $P_1, \ldots, P_n$, where:

$$P_1(\bar{X}) \text{ :- } V_{11}(\bar{X}_{11}), \ldots, V_{1m_1}(\bar{X}_{1m_1}).$$

$$P_2(\bar{X}) \text{ :- } V_{21}(\bar{X}_{21}), \ldots, V_{2m_2}(\bar{X}_{2m_2}).$$

$$\ldots$$

$$P_n(\bar{X}) \text{ :- } V_{n1}(\bar{X}_{n1}), \ldots, V_{nm_n}(\bar{X}_{nm_n}).$$

Here, each view V_{ij_k} belongs to the viewset $\mathcal{V}$.

By definition, the conjunctive query Q is equivalent to the union of the expansions of these queries P_i. Each expansion is a union of conjunctive queries, because each view V_{ij_k} may be a disjunctive view. From [28], the query Q is equivalent to one of the expanded queries, denoted E_k, and the remaining expanded queries are contained in Q. Let this expanded query E_k come from the query P_i in $P_1, \ldots, P_n$. We can keep all the views $\mathcal{V}'$ used in P_i. Notice that P_i is an equivalent *conjunctive* rewriting of Q using $\mathcal{V}'$. In addition, $size(\mathcal{V}') \leq size(\mathcal{V})$. □

COROLLARY 3.2. *For any finite workload of conjunctive queries and a database instance, assuming disjunctive rewritings, the problem of finding a minimal-size disjunctive viewset for the workload on the database instance is decidable.* □

4. QUERIES WITHOUT SELF-JOINS: THE PROBLEM IS IN NP

In this section we study workloads of conjunctive queries *without self-joins*. Figure 3 shows the view spaces we consider to find a minimal-size set of disjunctive views. We first show that if the workload queries do not have self-joins, then we need to consider only conjunctive views, because nontrivial disjunctive views do not add any new solutions (Section 4.1). We then further restrict our consideration to subexpression-type views (Section 4.2), and then to full-reducer views (Section 4.3). We show that by considering just these types of views we will always find at least one set of views that is globally optimal, for the given query workload and database instance, in the full search space of nontrivial disjunctive views. We show that the problem of finding an optimal disjunctive viewset for queries without self-joins is in NP, precisely because we can always find a minimal-size viewset in an extremely restricted search space of views.

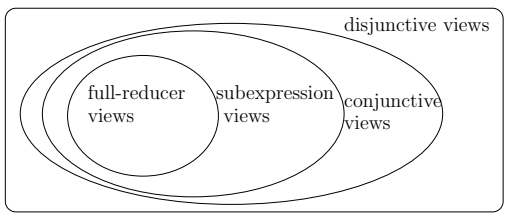

Figure 3: View space.

4.1 Conjunctive views are enough

We first give the following result.

THEOREM 4.1. *Suppose a set $\mathcal{V}$ of disjunctive views is a solution for a given database instance $\mathcal{D}$ and workload $\mathcal{Q}$ of conjunctive queries without self-joins. Then there exists another solution $\mathcal{V}'$ for $\mathcal{D}$ and $\mathcal{Q}$, such that all views in $\mathcal{V}'$ are conjunctive, and $size(\mathcal{V}') \leq size(\mathcal{V})$.* □

PROOF. We first look at singleton query workloads and then extend our observations to arbitrary query workloads.

Case 1 (singleton query workloads only). Let $\mathcal{Q}$ be any singleton query workload, $\mathcal{Q} = \{Q\}$, where the query

$$Q : ans(Z) \text{ :- } R_1(), \ldots, R_n().$$

is conjunctive and does not have self-joins. Let $\mathcal{D}$ be an arbitrary database instance, and let a set $\mathcal{V}$ of disjunctive views be a solution for $\mathcal{Q}$ and $\mathcal{D}$. Then there exists an equivalent rewriting $\mathcal{P}$ of Q using the views in $\mathcal{V}$. Without loss of generality, we assume that all views in $\mathcal{V}$ are used in $\mathcal{P}$. (Those views that are not used in $\mathcal{P}$ can be removed from $\mathcal{V}$ to save on storage space.) Consider each nontrivial disjunctive view

$$V = V_1 \cup V_2 \cup \cdots \cup V_k,$$

$k > 1$, that is used in $\mathcal{P}$; $V_1, \ldots, V_k$ are conjunctive views. The main idea of the proof is that there exists a transformation ν of the view V that produces a new disjunctive view:

$$V' = V_1' \cup V_2' \cup \cdots \cup V_k',$$

where $V_i' = \nu(V_i)$, $i = 1, \ldots, k$, and each V_i' is a conjunctive view. The new disjunctive view V' is used in a new equivalent rewriting $\mathcal{P}'$ of the query Q. We show that for any conjunctive component V_i' of the view V', V_i' can replace the view V in the equivalent rewriting $\mathcal{P}'$. Therefore, we can replace the *disjunctive* view V with a *conjunctive* view V_i', where $|V_i'| \leq |V'| \leq |V|$.

Now we give the details of the proof. Assume there are m occurrences of V in $\mathcal{P}$:

$$\mathcal{P} : ans(\bar{X}) \text{ :- } V(\bar{X}_1), \ldots, V(\bar{X}_m), G$$

where each of $\bar{X}$, $\bar{X}_1$, ..., $\bar{X}_m$ is a list of arguments, and G represents the instances of other views that are not V. Using the disjunctive definitions of all the disjunctive views to replace the instances in $\mathcal{P}$, we get a union of conjunctive queries, which is equivalent to the query Q as expansions. From [28], at least one of these queries — we denote it by P — is equivalent to Q as expansion: $P^{exp} \equiv Q$. Let μ be a containment mapping from P^{exp} to Q. By applying μ on $\mathcal{P}$, we get another equivalent rewriting:

$$\mathcal{P}' : ans(\bar{X}') \text{ :- } V(\bar{X}_1'), \ldots, V(\bar{X}_m'), G'$$

where $\bar{X}' = \mu(\bar{X})$ and $\bar{X}_i' = \mu(\bar{X}_i)$, $i = 1, \ldots, m$. G' represents the subgoals in $\mathcal{P}'$ that are not using the conjunctive components in V. In the definition of V:

$$V = V_1 \cup V_2 \cup \cdots \cup V_k$$

we remove all the V_i's that do not appear in $\mathcal{P}'$. Without loss of generality, $P' = \mu(P)$ can be represented as:

$$P' : ans(\bar{X}') \text{ :- } \hat{V}_1(\bar{X}_1'), \ldots, \hat{V}_m(\bar{X}_m'), G'$$

where each $\hat{V}_i$ is a conjunctive component of the view V.

Consider the conjunctive view $\hat{V}_1$ and its corresponding contained rewriting (in $\mathcal{P}'$) that does not use other conjunctive components in V:

$$H : ans(\bar{X}') \text{ :- } \hat{V}_1(\bar{X}_1'), \ldots, \hat{V}_1(\bar{X}_m'), G$$

Let β be a containment mapping from Q to H^{exp}. For any $j \geq 2$, we show that for all the local mappings (of β) from Q to the subgoals/arguments in $\hat{V}_1(\bar{X}_j')^{exp}$, we can "redirect" them to the corresponding subgoals/arguments in the expansion of $\hat{V}_1(\bar{X}_1')$, and thus get another containment mapping β' from Q to H^{exp}, where the images of Q do not come from the expansion of any $\hat{V}_1(\bar{X}_j')^{exp}$, $j \geq 2$.

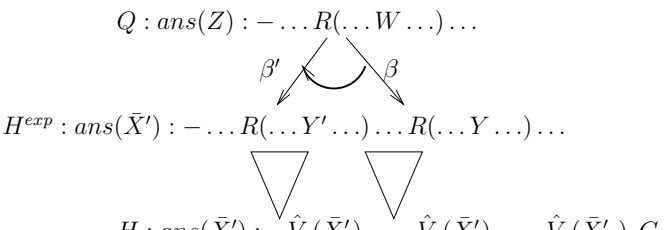

Figure 4: Redirecting the mapping β to β'.

Here are the details on redirecting the mappings. Consider each subgoal $R(\ldots Y \ldots)$ in $\hat{V}_1(\bar{X}_j')^{exp}$, where $j \geq 2$. Let $R(\ldots W \ldots)$ be the corresponding query subgoal in Q. Because Q does not have self-joins, it has only one instance of relation R. Here both Y and W are the l-th argument for relation R. There are two possible cases.

1. Y is a distinguished variable of $\hat{V}_1$. That is, in the definition of $\hat{V}_1$, there is at least one R-subgoal that "exports" its l-th attribute, which appears in the head. Since $R(\ldots W \ldots)$ is the only R-subgoal in Q, the mapping μ guarantees that $Y = W$. In addition, the expansion of $\hat{V}_1(\bar{X}_1')$ has a subgoal $R(\ldots W \ldots)$ because of the mapping μ. We then redirect the mapping from W in Q (to the W in the expansion of $\hat{V}_1(\bar{X}_j')$) to the W in the expansion of $\hat{V}_1(\bar{X}_1')$.[1]

2. Y is a nondistinguished variable of $\hat{V}_1$, i.e., Y is "fresh" in the expansion of $\hat{V}_1(\bar{X}_j')$. Then W must also be a nondistinguished variable of Q. The expansion of $\hat{V}_1(\bar{X}_1')$ also provides a corresponding fresh variable Y' that can be used as the image of W. Notice that in this case, $\hat{V}_1(\bar{X}_1')^{exp}$ and $\hat{V}_1(\bar{X}_j')^{exp}$ provide all instances of Y' and Y, respectively. (The reason is as follows: as observed in [1, 27], if a nondistinguished query variable A is mapped to a nondistinguished variable B in the expansion of a rewriting, then all instances of A should be mapped the corresponding instances of B in the expansion.) So we can redirect the mappings (to the Y instances in the expansion of $\hat{V}_1(\bar{X}_j')$, $j \geq 2$) to the Y' instances in the expansion of $\hat{V}_1(\bar{X}_1')$.

By redirecting all the local mappings from the expansion of $\hat{V}_1(\bar{X}_j')$ to the subgoals in the expansion of $\hat{V}_1(\bar{X}_1')$, we have obtained, from the containment mapping β, another containment mapping β', from Q to the expansion of the following rewriting:

$$H_{new} : ans(\bar{X}') \text{ :- } \hat{V}_1(\bar{X}_1'), G.$$

The conjunctive mapping β' implies that $H_{new}^{exp} \sqsubseteq Q$. Since P' is an equivalent rewriting of Q, we obtain that Q is contained in the expansion of P'. In addition, the expansion

[1]Note that this claim is not correct if the relation R appears more than once in Q; see the flight example in Section 3.3.

of P' is also contained in H^{exp}_{new}, since the subgoals in H_{new} are a subset of those in P'. Thus $Q \sqsubseteq H^{exp}_{new}$. So H_{new} is also an equivalent rewriting of Q. Notice that in H_{new}, of all the conjunctive components of V we use only one component $\hat{V}_1$. Thus we can replace the *disjunctive* view V with a *conjunctive* view $\hat{V}_1$.

By doing this replacement for all the disjunctive views in $\mathcal{V}$, we get a set $\mathcal{V}'$ of purely conjunctive views that can answer the query Q. By construction, $\mathcal{V}'$ does not require more storage space than $\mathcal{V}$.

Case 2 (arbitrary query workloads). Let $\mathcal{Q}$ be a query workload, such that $\mathcal{Q}$ has at least two conjunctive queries, $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$, $n \geq 2$, and such that all queries in $\mathcal{Q}$ are queries without self-joins. Let $\mathcal{V}$ be a disjunctive viewset that is a solution for $\mathcal{Q}$ and for an arbitrary database instance $\mathcal{D}$. If, for any nontrivial disjunctive view V in $\mathcal{V}$, there exists at most one query in $\mathcal{Q}$, such that the rewriting of the query (in terms of $\mathcal{V}$) uses the view V, then this case reduces to Case 1 above. In the remainder of the proof we assume that in the viewset $\mathcal{V}$, there is a nontrivial disjunctive view $V = V_1 \cup \ldots \cup V_m$ (each V_i is purely conjunctive), such that V is used in the the rewritings of at least two queries in the workload $\mathcal{Q}$.

Without loss of generality, let Q_1 and Q_2 be two such queries. Using the reasoning in Case 1 above, we find that there are two conjuncts, V_1 and V_2, of the view V, such that the query Q_1 (resp. Q_2) has a *conjunctive* rewriting that uses just the conjunct V_1 (resp. V_2) of V. (We assume here that V is the only nontrivial disjunctive view used in the rewritings of the queries Q_1 and Q_2. It is easy to generalize the proof to the case of multiple disjunctive views used in the rewritings of multiple queries in the workload $\mathcal{Q}$.) In the remainder of the proof, we show that $V_1 = V_2$. It follows that if a nontrivial disjunctive view V is used to rewrite more than one query in the workload $\mathcal{Q}$, then there exists a conjunct $\tilde{V}$ of the view V, such that $\tilde{V}$ can be used to construct equivalent *conjunctive* rewritings of all those queries. This observation concludes our proof.

We now show that $V_1 = V_2$. Using the reasoning in Case 1, we can show that the conjunct V_1 contains the self-join $V_2 \bowtie \ldots \bowtie V_2$ in the expansion of the rewriting of the query Q_1 (this rewriting uses the view V). Similarly, we can observe that the conjunct V_2 contains the self-join $V_1 \bowtie \ldots \bowtie V_1$ in the expansion of the rewriting of the query Q_2. Using these self-joins and using again the reasoning in Case 1, we construct two containment mappings: (1) a mapping μ_1 from V_1 to just one occurrence of V_2 in the self-join $V_2 \bowtie \ldots \bowtie V_2$, and (2) a mapping μ_2 from V_2 to just one occurrence of V_1 in the self-join $V_1 \bowtie \ldots \bowtie V_1$. Because both V_1 and V_2 are conjunctive views, we conclude that in the disjunctive view V, the conjuncts V_1 and V_2 are equivalent. This observation concludes the proof of the theorem. □

4.2 Subexpression-type views are enough

We saw in Section 3 that to find a minimal-size conjunctive viewset for a workload of conjunctive queries, it is enough to consider views whose definition is at most exponential in the length of the longest query definition in the workload. Now we show that for workloads of queries without self-joins, we can further reduce the size of the search space of views by considering only those conjunctive views that are defined as subexpressions of the workload queries. By considering such views only, we can still find a conjunctive viewset that is an optimal solution for the given problem input in the space of disjunctive views. We begin by showing the following result.

THEOREM 4.2. *Given a database instance, for any conjunctive query Q without self-joins, there is a minimal-size conjunctive viewset $\mathcal{U}$, such that each view in $\mathcal{U}$ is a subexpression of Q, with possibly attributes projected.* □

We prove the theorem by using a modification of the proof of Theorem 3 in [12]. Suppose a viewset $\mathcal{V}$ is some (not necessarily minimal-size) conjunctive viewset, such that the query Q has a rewriting using $\mathcal{V}$. From the viewset $\mathcal{V}$ we construct another viewset, $\mathcal{W}$, such that $\mathcal{W}$ also provides a rewriting of the query Q and has the following properties. The amount of space required to store $\mathcal{W}$ does not exceed the space required to store $\mathcal{V}$, on *any* database, and each view in $\mathcal{W}$ is defined as a subexpression of the query Q, with possibly attributes projected.

Theorems 4.1 and 4.2 and the results in Section 3 imply that to find a minimal-size disjunctive viewset for a workload of conjunctive queries without self-joins, it is enough to consider conjunctive containment-target views whose body is a subexpression of at least one query in the workload. As a result, we have reduced the size of the search space of views that includes all views in all minimal-size disjunctive viewsets, from triply-exponential to singly-exponential in the size of the query workload.

COROLLARY 4.1. *For any database instance and any single conjunctive query without self-joins, we can construct a finite search space of views that includes all views in at least one minimal-size disjunctive viewset for the query, such that the number of views in the search space is at most exponential in the size of the query definition.* □

4.3 Full-reducer views are enough

Now we further reduce the size of the search space of views for a single conjunctive query without self-joins, by considering only full-reducer views. A full-reducer view is a view whose body is the query body [36]. Consider Example 3.1 again. In that example, the body of each view can be replaced by the full body of the query Q_k. After the replacement, the number of tuples in each view cannot increase. More precisely, none of the views will have any dangling tuples after the replacement. At the same time, for each such view, there exists a database where the view is part of some minimal-size viewset for the query Q_k. We formalize these observations in the following result.

LEMMA 4.1. *Given a database instance, consider any conjunctive query Q without self-joins; let $\mathcal{V}$ be any solution for Q. For each view $V \in \mathcal{V}$ that is not a full-reducer view, we can construct from the view V a new view W, by replacing the body of V with the body of Q. Then the resulting viewset $\mathcal{V}'$ is also a solution for Q, and $size(\mathcal{V}') \leq size(\mathcal{V})$.* □

We prove the lemma as follows. For any conjunctive query Q without self-joins and for its conjunctive solution $\mathcal{V}$, consider any containment-target view $V \in \mathcal{V}$, such that V is defined as a proper subexpression of the query. From each such view V, we can construct a full-reducer containment-target view W, by simply adding to the definition of V the missing subgoals of the query Q. We can show that W is contained in

V. Therefore, on any database we have $size(W) \leq size(V)$. Now consider an equivalent rewriting P of Q in terms of the viewset $\mathcal{V}$. If, in the rewriting P, we replace V by W, the resulting rewriting P' will still be equivalent to Q, for the following reason. Consider an expansion P^{exp} of P.

1. Consider a containment mapping μ from Q to P^{exp}. In constructing W from V, all we do is add subgoals to V. Thus μ will also be a containment mapping from Q to the expansion of P'.

2. Consider a containment mapping ν from P^{exp} to Q. Because the body of the view W is the body of the query Q, the containment mapping ν can be extended to map the expansion of P' to query Q.

We conclude that for any conjunctive query Q without self-joins, by replacing, in a solution $\mathcal{V}$ for a query Q, any view V by a corresponding full-reducer view W, we obtain another solution $\mathcal{V}'$, such that $size(\mathcal{V}') \leq size(\mathcal{V})$.

COROLLARY 4.2. *Given a database instance, for any conjunctive query without self-joins, there exists a minimal-size disjunctive viewset $\mathcal{V}$, such that each view in $\mathcal{V}$ is a conjunctive full-reducer containment-target view.* □

We now consider the problem of finding minimal-size views for a query workload that may have more than one query. We first obtain a minimal-size set of full-reducer views for each individual workload query. Because each view is a containment-target view for the query, the number of views in each rewriting does not exceed the number of query subgoals. We can then merge some of the views across the viewsets, to obtain a (possibly smaller-size) viewset for the entire query workload. Each merged view can replace, in the query rewritings, any of the views it was obtained from. Because we merge views without self-joins by removing view subgoals, each merged view has at most as many subgoals as the longest query in the workload. It follows that for a rewriting of the query using any such viewset, the expansion of the rewriting is polynomial in the size of the query.

4.4 Workloads of queries without self-joins: The problem is in NP

Notice that, from the update to Example 3.1 that we suggested in Section 4.3, it follows that the search space of conjunctive full-reducer containment-target views can still be exponential in the length of the query definition. At the same time, we have the following powerful result.

THEOREM 4.3. *Given a database instance, for any finite workload of conjunctive queries without self-joins, the problem of finding a minimal-size disjunctive viewset is in NP.* □

Here is an intuition for the proof. Consider a query Q with n subgoals, a database $\mathcal{D}$, and an integer K. To check whether a conjunctive viewset $\mathcal{V}$ is a solution for the query Q, such that storing the views in the database $\mathcal{D}$ will require at most K bytes, we can do two things. First, to see whether the viewset $\mathcal{V}$ gives an equivalent rewriting of query Q, we need to check a witness that provides (1) a rewriting of the query in terms of the views, and (2) the containment mappings between the query and the rewriting. Second, to check whether storing the views in the database $\mathcal{D}$ will require at most K bytes, it is enough to add up the sizes of the views in $\mathcal{V}$ on the database $\mathcal{D}$. From the results in Sections 3 and 4 it follows that the sizes of the structures we need to examine (and, therefore, the time required to examine them) are polynomial in the size of the query Q.

5. A DYNAMIC-PROGRAMMING ALGORITHM

In this section we study how to find a set of conjunctive views that is a minimal-size viewset, in the space of disjunctive views, for a single conjunctive query without self-joins. As we saw in Section 4, it is enough to consider conjunctive full-reducer containment-target views for the query. We describe a system-R-like dynamic-programming algorithm [29] for finding a minimal-size viewset by searching in the space of these views.

One way to find a disjunctive solution for a non-singleton workload of conjunctive queries without self-joins is as follows. In the first step, we use the dynamic-programming algorithm described in this section to obtain an optimal viewset for each workload query separately. We then use a view-merging algorithm (see Section 4.3) to obtain a solution for the entire workload.

5.1 The algorithm

In the remainder of this section, we consider a single conjunctive query Q without self-joins. We assume that Q is a join of n subgoals $R_1, \ldots, R_n$, where all R_i's are different:

$$Q : ans(\bar{X}) \text{ :- } R_1(\bar{X}_1), \ldots, R_n(\bar{X}_n).$$

The algorithm considers the search space of full-reducer views for the query Q, where all views are of the form:

$$W_i : ans(\bar{Y}_i) \text{ :- } R_1(\bar{X}_1), \ldots, R_n(\bar{X}_n).$$

That is, each view W_i is defined as the set of *all* subgoals of the query Q. Some view variables could be nondistinguished. To ensure that each W_i is a containment-target view for the query, we need to make sure that W_i covers a nonempty subset of the query's subgoals. A view W_i covers a subset $\{R_{i_1}, \ldots, R_{i_{k_i}}\}$ of subgoals of the query Q if the set of distinguished variables of the view W_i includes all variables of the set $\{R_{i_1}, \ldots, R_{i_{k_i}}\}$ that also occur "outside" the set. That is, these variables occur either in the query's head, or in the subgoals of Q that are not among $\{R_{i_1}, \ldots, R_{i_{k_i}}\}$. It is easy to see that it is enough to consider just those views W_i whose distinguished variables are exactly the set of variables that occur outside the set $\{R_{i_1}, \ldots, R_{i_{k_i}}\}$.

Formally, the algorithm finds an optimal solution for a conjunctive query Q without self-joins by searching in the space $\mathcal{W}$ of all views W_i described above. It constructs a table with an entry for each nonempty subset of the set $\mathcal{R} = \{R_1, \ldots, R_n\}$. For each subset $\mathcal{R}' = \{R_{i_1}, \ldots, R_{i_{k_i}}\}$ of $\mathcal{R}$, $k_i \leq n$, the algorithm constructs a table entry with the following information:

1. The total size S of the optimal (i.e., minimum-size) set of views in $\mathcal{W}$ that allows us to compute the join of the subgoals $\mathcal{R}'$. To compute S, we take the minimum of the following:

   (a) The size of the minimum-size full-reducer containment-target view that covers the set $\mathcal{R}'$; the view

is a join of all relations in $\mathcal{R}'$, with just the necessary variables of $\mathcal{R}'$ projected out (a variable is necessary for the set $\mathcal{R}'$ if it is used in future joins or is in the head of the query Q), and

(b) For each partition of $\mathcal{R}'$ into two subsets $\mathcal{R}'_1$ and $\mathcal{R}'_2$, the sum of the total sizes of the optimal viewsets that cover each of $\mathcal{R}'_1$ and $\mathcal{R}'_2$.

2. The expression to compute the join of the subgoals $\mathcal{R}'$ of Q, using the optimal set of views.

The construction of the table is by induction on the subset size. The table entry for the full set $\mathcal{R} = \{R_1, \ldots, R_n\}$ contains a viewset that allows us to compute the query $Q = \pi_{\bar{X}}(R_1 \bowtie \cdots \bowtie R_n)$. For instance, suppose the query

$$Q(X, Y, Z) \text{ :- } R_1(X, T), R_2(Y, T), R_3(Z, T).$$

has an optimal viewset $\{ W_1, W_2 \}$:

$$\begin{array}{ll} W_1(X, Y, T) & \text{:- } R_1(X, T), R_2(Y, T), R_3(Z, T). \\ W_2(Z, T) & \text{:- } R_1(X, T), R_2(Y, T), R_3(Z, T). \end{array}$$

The view W_1 covers the first two subgoals of the query, whereas the view W_2 covers the third. Thus in the table entry for the set $\mathcal{R} = \{R_1, R_2, R_3\}$, the expression to compute the query Q will be $W_1 \bowtie W_2$.

PROPOSITION 5.1. *For a conjunctive query without self-joins, the algorithm described above always returns a minimal-size viewset for the query in the space of disjunctive views. The algorithm constructs a table whose size is exponential in the number of the query subgoals.* □

PROOF. We prove the claim by induction. Let n be the number of subgoals of the query: $Q = \pi_{\bar{X}}(R_1 \bowtie \ldots \bowtie R_n)$.

Basis 1. $n = 1$: any single relation R_i, with just the necessary variables projected out, always constitutes an optimal viewset for covering the subgoal R_i of the query Q.

Basis 2. $n = 2$: for any $i \leq n$ and $j \leq n$, to find a minimum-size viewset that covers the subgoals R_i and R_j in the query Q, it is enough to compare the the sizes of the following views:

- A minimum-size full-reducer containment-target view that covers both subgoals. This view is computed by taking a join of all relations in the definition of the query Q, and then projecting out just the variables of R_i and R_j that are used outside the two subgoals.
- The two views that cover the two subgoals separately.

Induction. Suppose the induction assumption is true for all sets of subgoals of the query Q, such that the number of subgoals (relations) in each set is between 1 and $k-1$. Consider a set $\mathcal{R}'$ of subgoals of the query Q, such that $\mathcal{R}'$ has k relations. For each way of partitioning the set $\mathcal{R}'$ into two nonempty subsets, the algorithm computes the sum of sizes of two optimal viewsets, where each viewset covers one partition. (Each partition is a set of subgoals of the query that has between 1 and $k-1$ relations.) Then the algorithm takes the minimum among these sums and compares the result to the size of the view that covers all k relations in the set $\mathcal{R}'$. Notice that the algorithm considers all ways of covering the k relations in the set $\mathcal{R}'$. Therefore, it returns an optimal viewset that covers the k relations. By induction, the reasoning is true for the case $k = n$.

To finalize the proof, we observe that in a globally optimal viewset for the query Q, each subgoal of the query Q is covered by exactly one view. It is easy to see that the algorithm considers all such viewsets. □

The algorithm can be modified to consider only those subsets of query subgoals that do not have cross-products. This modification does not sacrifice completeness of the algorithm. For some queries (e.g., chain queries), the modified algorithm has to construct only a polynomial number of table entries and still produces an optimal solution. Meanwhile, to find an optimal solution for query Q_k in Example 3.1, the modified algorithm has to construct all $2^k - 1$ table entries.

5.2 Reducing the size of the search space

In the space searched by the algorithm, there can be an exponential number of solutions. In this section we propose heuristics for reducing the size of the search space. The main idea is to explore the strategy of reducing the arity (the number of attributes) of each view in a solution. Intuitively, the strategy allows us to keep the views small by reducing the number of their attributes. (Recall that this work has been done under the set-semantics assumption, where a nontrivial projection on a table can have fewer tuples than the table itself. At the same time, the strategy also makes sense under the bag-semantics assumption, because it strives to reduce the total number of attributes in a solution and therefore to reduce the disk space that is required to store the solution.) Here are three heuristics to reduce the arity of the views in a solution.

1. Consider only those subsets of query subgoals that have no more than a certain number of subgoals (e.g., up to 3 subgoals).
2. Consider only those subsets of query subgoals that have up to a certain total number of attributes (e.g., no more than 10 attributes).
3. Partition the subgoals of the query Q into several subsets of subgoals, and then apply the dynamic-programming algorithm separately to each subset. By taking the union of the resulting partial viewsets, we obtain a solution for the entire query Q.

In [25] we describe more techniques for reducing the size of the search space of views and for computing efficiently the view relations in a minimal-size viewset. In that paper we also discuss how to efficiently compute the answer to a query using the views.

6. CONCLUSIONS

In this paper we studied the problem of finding viewsets to answer a workload of conjunctive queries, such that the total size of the views is minimal. We gave decidability and complexity results for workloads of conjunctive queries; the results differ significantly depending on whether the queries have self-joins. We developed an algorithm for finding an optimal solution, and discussed heuristics to make the algorithm efficient.

7. REFERENCES

[1] F. Afrati, C. Li, and J. D. Ullman. Generating efficient plans using views. In *SIGMOD*, pages 319–330, 2001.

[2] S. Agrawal, S. Chaudhuri, and V. Narasayya. Automated selection of materialized views and indexes in Microsoft SQL Server. In *Proc. of VLDB*, pages 496–505, Cairo, Egypt, 2000.

[3] E. Baralis, S. Paraboschi, and E. Teniente. Materialized view selection in a multidimensional database. In *Proc. of VLDB*, 1997.

[4] R. Bello, K. Dias, A. Downing, J. Feenan, J. Finnerty, W. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized views in Oracle. In *Proc. of VLDB*, pages 659–664, 1998.

[5] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. R. Jr. Query processing in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems (TODS)*, 6(4):602–625, 1981.

[6] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *PODS*, pages 386–391, July - August 2000.

[7] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Answering regular path queries using views. In *ICDE*, pages 389–398, 2000.

[8] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems*. McGraw-Hill Book Company, 1984.

[9] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. *STOC*, pages 77–90, 1977.

[10] Z. Chen and P. Seshadri. An algebraic compression framework for query results. In *ICDE*, pages 177–188, 2000.

[11] R. Chirkova. The view-selection problem has an exponential-time lower bound for conjunctive queries and views. *PODS*, pages 159–168, 2002.

[12] R. Chirkova and M. R. Genesereth. Linearly bounded reformulations of conjunctive databases. *DOOD*, 2000.

[13] R. Chirkova, A. Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. *Proc. of VLDB*, pages 59–68, 2001.

[14] J. Goldstein and P.-A. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *SIGMOD*, pages 331–342, 2001.

[15] H. Gupta. Selection of views to materialize in a data warehouse. In *ICDT*, 1997.

[16] H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman. Index selection in olap. In *ICDE*, 1997.

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

[18] H. Hacıgümüş, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.

[19] H. Hacıgümüş, B. Iyer, and S. Mehrotra. Providing database as a service. In *ICDE*, 2002.

[20] A. Halevy. Answering queries using views: A survey. In *Very Large Database Journal*, 2001.

[21] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *SIGMOD*, 1996.

[22] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

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

[24] C. Li, M. Bawa, and J. D. Ullman. Minimizing view sets without losing query-answering power. In *ICDT*, pages 99–113, 2001.

[25] J. Li, R. Chirkova, and C. Li. Minimizing data-communication costs by decomposing query results in client-server environments. Technical report, Information and Computer Science, UC Irvine, 2003.

[26] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, 1999.

[27] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. In *Proc. of VLDB*, 2000.

[28] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.

[29] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34, 1979.

[30] D. Theodoratos, S. Ligoudistianos, and T. Sellis. Designing the global data warehouse with spj views. In *CAiSE*, 1999.

[31] D. Theodoratos and T. Sellis. Data warehouse configuration. In *Proc. of VLDB*, 1997.

[32] TPC-H. `http://www.tpc.org/tpch/`.

[33] J. D. Ullman. *Principles of Database and Knowledge-base Systems, Volumes I: Classical Database Systems*. Computer Science Press, New York, 1988.

[34] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[35] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proc. of VLDB*, 1997.

[36] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of VLDB*, pages 82–94. IEEE Computer Society Press, 1981.

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