Title

Answering queries using materialized views with minimum size

Permalink
https://escholarship.org/uc/item/53z9z73n

Journal
Vldb Journal, 15(3)

ISSN
1066-8888

Authors
Chirkova, R
Li, C
Li, J

Publication Date
2006-09-01

Peer reviewed

The VLDB Journal manuscript No.
(will be inserted by the editor)

Answering Queries Using Materialized Views with Minimum Size

Rada Chirkova[1], **Chen Li**[2], **Jia Li**[2] *

[1] North Carolina State University, chirkova@csc.ncsu.edu
[2] University of California, Irvine, {chenli,jiali}@ics.uci.edu

Received: date / Revised version: date

Abstract In this paper we study the following problem. Given a database and a set of queries, we want to find a set of views that can compute the answers to the queries, such that the amount of space, in bytes, required to store the viewset is minimum on the given database. (We also handle problem instances where the input has a *set* of database instances, as described by an oracle that returns the sizes of view relations for given view definitions.) This problem is important for 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. Further, for queries without self-joins we describe a very compact search space of views, which contains all views in at least one optimal viewset. We present techniques for finding a minimum-size viewset for a single query without self-joins by using the shape of the query and its constraints, and validate the approach by extensive experiments.

Keywords: views, data warehouses, minimum-size viewsets, distributed systems.

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

* Part of this article was published in [CL03]. In addition to the prior materials, this article contains new theoretical results, as well as new results on how to efficiently implement the proposed techniques (Sections 5 and 6).

the viewset) is minimum. This problem exists in many environments, such as distributed databases [BGW$^+$81, CP84, ÖV99], data integration [Len02], and the recent "database-as-a-service" model [HIM02]. For example, mediators in data-integration systems support seamless access to autonomous, heterogeneous information sources [Wie92]. 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 [HKWY97]. 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 [HILM02], 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.

In client-server applications, an important metric is minimizing data-transfer time. In contrast, in a data warehouse, minimizing data-transfer time is irrelevant, but it is crucial to keep around materialized views whose total size is as small as possible, to try to prevent accessing original stored relations in the data sources when processing and answering warehouse queries. Here the problem is how to materialize locally (in the warehouse) relations with small total size, to avoid excessive transfer time for big original stored relations.

In general, given a set of queries (a *query workload*) and a fixed database, we want to define and compute 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 minimum on the given database. In this paper we study this problem for select-project-join queries. (We also handle problem instances where the input has a *set* of databases; see Section 2.3.) We make 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, disjunctive views can

give rise to smaller viewsets than purely conjunctive views. We establish that the problem of finding a minimum-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 minimum-size viewset it is not necessary to consider nontrivial disjunctive rewritings — that is, rewritings that are unions of at least two select-project-join queries.

2. In Section 4 we study workloads of conjunctive queries without self-joins, and show that for singleton query workloads, disjunctive views cannot provide smaller viewsets than purely conjunctive views. Thus, it is enough to consider purely conjunctive views when looking for a minimum-size disjunctive viewset; moreover, it is enough to explore a very restricted search space of such views. In addition, we show that the problem of finding a minimum-size viewset is in NP, for both singleton and non-singleton workloads of queries without self-joins.
3. In Section 5 we present techniques for finding a minimum-size viewset for a single query without self-joins by using the shape of the query and its key constraints. In Section 6 we report our experimental results to evaluate these techniques.

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 [Gup97, TLS99, TS97, YKL97]; one direction is to materialize views and indexes for data cubes in online analytic processing (OLAP) [BPT97, GHRU97, HRU96]. 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 [BDD$^+$98, GL01, ZCL$^+$00]. Choosing an appropriate set of views to materialize in the database is crucial in order to obtain performance benefits from these new features [ACN00].

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 [Chi02, CG00, CHS02], 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., [ALU01, LMSS95, Hal01]), view-based query answering (e.g., [CGL00, CGLV00]), and minimizing viewsets without losing query-answering power [LBU01]. In addition, there has been a lot of work on minimizing data-communication costs in distributed database systems (e.g., [CP84, ÖV99]). Our novel approach complements existing techniques. In particular, our approach is orthogonal to data-compression approaches (see, e.g., [CS00]); combining the two approaches in client-server systems can further reduce the communication costs and thus save on the total time required to send the query result to the client.

2 Problem Formulation

In this section we formulate the problem of finding a set of views with the minimum size to answer queries. We first present and discuss a motivating example and then give a formal specification of the problem.

2.1 Motivating Example

Consider the following simplified versions of three relation schemas in the TPC-H benchmark [TPC]:

```
customer(custkey(4),name(25),mktsgmt(10))
order(orderkey(4),custkey(4),orderdate(8),shippriority(4),comment(79))
lineitem(linenum(4),orderkey(4),quantity(4),shipdate(8),shipmode(10))
```

The number after each attribute is the size of the values of the attribute, in bytes. For simplicity, we assume for each attribute that all its values are of the same size. We further assume that the relations reside on a server that accepts queries from a client.

```
SELECT   c.name, o.orderdate, o.shippriority, o.comment,
         l.orderkey, l.quantity, l.shipmode
FROM     customer c, orders o, lineitem l
WHERE    c.mktsgmnt = 'BUILDING'
         AND c.custkey = o.custkey
         AND o.orderkey = l.orderkey;
```

Fig. 1 Query Q_1.

Suppose a user at the client issues a query Q_1, shown in Figure 1. Q_1 is a variation on the Query 3 in the TCP-H benchmark [TPC]. The server computes the answer to Q_1 and sends it back to the client. Suppose there are 4000 tuples in the answer to Q_1 on the database at the server. It follows that the total number of bytes sent to the client is:

$$4,000 \times (25 + 8 + 4 + 4 + 79 + 4 + 10) = 516,000.$$

Table 1 Partial results of query Q_1.

id	name	orderkey	comment	shippriority	orderdate	quantity	shipmode
t_1	Tom	134721	...	0	3/14/1995	26	REG AIR
t_2	Tom	134721	...	0	3/14/1995	75	REG AIR
t_3	Tom	134721	...	0	3/14/1995	43	AIR
t_4	Jack	571683	...	0	12/21/1994	43	MAIL
t_5	Jack	571683	...	0	12/21/1994	33	AIR
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Let us see if we can reduce the communication costs by reducing the amount of data to be transferred to the client, while still giving the client necessary data to compute the final answer to Q_1. Table 1 shows a fragment of the answer to Q_1. The answer to Q_1 has redundancies; for instance, tuples t_1 through t_3 are the same except in the values of `l.quantity` and `l.shipmode`; tuples t_4 and t_5 have similar redundancy. One reason for the redundancy is that an order could have several lineitems with different quantities and shipmodes. In the join results, this information generates several tuples with the same values of `customer` and `order`. Based on this observation, we can decompose the answer to Q_1 into intermediate results — *views* V_1 and V_2 — as shown in Figure 2. We will obtain the answer to the query Q_1 by joining the views.

```
   View V1:
SELECT   c.name, o.orderdate, o.shippriority,
         o.comment, o.orderkey
FROM     customer c, orders o, lineitem l
WHERE    c.mktsgmnt = 'BUILDING'
         AND c.custkey = o.custkey
         AND o.orderkey = l.orderkey;
   View V2:
SELECT   l.orderkey, l.quantity, l.shipmode
FROM     customer c, orders o, lineitem l
WHERE    c.mktsgmnt = 'BUILDING'
         AND c.custkey = o.custkey
         AND o.orderkey = l.orderkey;
```

Fig. 2 Decomposing the query answer into two views.

To illustrate the motivation of work, let us assume that there are $1,200$ tuples in the answer to view V_1 and $4,000$ tuples in the answer to view V_2. By using the sizes of the attribute values in the answers to the two views, we obtain that the total size of the answers to the views is $216,000$ bytes. Recall that the size of the answer to the query Q_1 is 516,000 bytes, and that it is possible to compute the answer to Q_1 using the answers to the views V_1 and V_2. It follows that instead of sending the client the (large) answer to the query Q_1, the server can reduce the transmission costs by sending the client the results of the two views; the client can then use the view results to compute the answer to the query.

This example shows that it is possible to decompose queries into intermediate views, such that the answers to the views can be used to compute the exact answer to the query, and the total size of the answers to the views can be much smaller than the size of the answer to the query. At the same time, we make the following two observations. (1) When trying to reduce the redundancy in the query answers, we may need to add more attributes that will allow joins of the view results. (2) There is more than one way to decompose the answer into views.

Before giving the problem formulation, we briefly review important concepts of conjunctive queries and answering queries using views.

2.2 Queries and Rewritings

In this paper we consider select-project-join SQL queries and their unions. For convenience of notation in the definitions and proofs, in the remainder of the paper we use the following *conjunctive query* representation of select-project-join SQL queries:

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

In a subgoal $R_i(\bar{X}_i)$, $i \in \{1, \ldots, n\}$, predicate R_i corresponds to a base (stored) relation, and each argument in the subgoal is either a variable or a constant. A *disjunctive query* is a union of conjunctive queries; a disjunctive query is *nontrivial* if it is a union of at least two conjunctive queries that are not all pairwise equivalent. We consider views defined on base relations by safe conjunctive or disjunctive 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 assume set semantics for query answers.

For instance, query Q_1 in Section 2.1 can be represented as the following conjunctive query:

$$Q_1(N, OD, SP, C, OK, QT, SM) \text{ :- } customer(CK, N, \texttt{'BUILDING'}), \\ orders(OK, CK, OD, SP, C), lineitem(LN, OK, QT, SD, SM).$$

A query Q is *contained* in a query Q', denoted $Q \sqsubseteq Q'$, if for every database instance D, the answer to Q on D is a subset of the answer to Q' on D. The two queries are *equivalent* if $Q \sqsubseteq Q'$ and $Q' \sqsubseteq Q$. A conjunctive query Q is contained in a conjunctive query Q' if and only if there is a *containment mapping* from Q' to Q [CM77] — that is, there is a homomorphism μ from the variables of Q' to variables and constants of Q such that, after μ is applied to Q', (1) the head of Q' becomes the head of Q, and (2) each subgoal of Q' becomes *some* subgoal of Q. (Not all subgoals of Q need to be in the image of the body of Q' under μ.) Consider an abstract example:

$$\begin{array}{ll} Q(X,Y) \text{ :- } & p(X,a,Y), s(Y,b). \\ Q'(A,C) \text{ :- } & p(A,B,C), p(A,a,D). \end{array}$$

Here, a and b are constants. By definition, mapping $\mu = \{A \rightarrow X, B \rightarrow a, C \rightarrow Y, D \rightarrow Y\}$ is a containment mapping from Q' to Q. Thus, by [CM77], Q is contained in Q'. (Because Q' does not have subgoals with predicate name s, Q' is not contained in Q.)

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. Existentially quantified variables in a view are replaced by fresh variables in P^{exp}. 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."

For instance, the following are the views V_1 and V_2 from Section 2.1, in a conjunctive-query form.

$$\begin{array}{ll} V_1(N,OD,SP,C,OK) \text{ :- } & customer(CK,N, \texttt{'BUILDING'}), \\ & orders(OK,CK,OD,SP,C), \\ & lineitem(LN,OK,QT,SD,SM). \\ V_2(OD,QT,SM) \text{ :- } & customer(CK,N, \texttt{'BUILDING'}), \\ & orders(OK,CK,OD,SP,C), \\ & lineitem(LN,OK,QT,SD,SM). \end{array}$$

The following query P is an equivalent rewriting of the query Q_1 using the two views.

$$\begin{array}{ll} P(N,OD,SP,C,OK,QT,SM) \text{ :- } & V_1(N,OD,SP,C,OK), \\ & V_2(OD,QT,SM). \end{array}$$

In this paper 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.

2.3 Problem Statement

Given a set, or *workload*, $\mathcal{Q}$ of conjunctive queries on stored relations $R_1, \ldots,$ R_n, and given a fixed database D, we want to find and precompute a set $\mathcal{V}$ of intermediate results, defined as views $V_1, \ldots, V_k$ on these relations, such that there exist equivalent rewritings of all the queries in the workload $\mathcal{Q}$ in terms of the views in $\mathcal{V}$ only. Our goal is to find an *optimal solution* — to choose, among all such sets of views $\mathcal{V}$, a set $\mathcal{V}^*$ whose total size

$$\sum_{V_i \in \mathcal{V}^*} size(V_i, D)$$

is minimum on the given database. The size of a view V_i, $size(V_i, D)$, is the amount of space, in bytes, required to store the answer to V_i on the database D. 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 $\mathcal{V}$.

The above problem statement is appropriate in settings such as database as a service. At the same time, having a single database in the problem input is not practical in all settings. For instance, in data warehousing one would not like to recompute the set of actual views to materialize on each change of the data. Because our goal is to find a set of views with minimum *total size*, all the results in this paper apply to a more general problem setting, where a database in the problem input is replaced by an oracle that instantaneously gives the size of the relation for each permissible (in our setting, conjunctive or disjunctive) view on a database. By definition, such an oracle describes a *set* of databases. This form of the problem statement has been used in the literature — see, for instance, [CHS02].

3 Decidability and Complexity

In this section and in Section 4, we study decidability and complexity of the problem, and the search space of the possible solutions.

3.1 Disjunctive Rewritings Are Not Needed

In the problem of finding a minimum-size viewset for a given database and query workload, suppose we allow disjunctive views. Thus, a *minimum-size disjunctive viewset* for a workload of conjunctive queries is a minimum-size viewset for the workload in the space of disjunctive views. Some of its disjunctive views can be purely conjunctive. Suppose we allow conjunctive or disjunctive rewritings of the workload queries using the views. In this section we show that to find a minimum-size viewset under these assumptions, purely conjunctive rewritings are all we need to examine.

Theorem 1 *Let $\mathcal{Q}$ be an arbitrary finite workload of conjunctive queries, and let $\mathcal{V}$ be a disjunctive viewset that gives an equivalent* disjunctive *rewriting of each query in the workload $\mathcal{Q}$. Then there exists a subset $\mathcal{V}'$ of $\mathcal{V}$, such that each query in $\mathcal{Q}$ has an equivalent* conjunctive *rewriting in terms of $\mathcal{V}'$.*

Proof Given a query workload $\mathcal{Q}$, let $\mathcal{V}$ be a disjunctive viewset, such that $\mathcal{V}$ gives an equivalent *disjunctive* rewriting of each query in $\mathcal{Q}$. 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

$$P_i(\bar{X}) \text{ :- } V_{i1}(\bar{X}_{i1}), \ldots, V_{im_i}(\bar{X}_{im_i}), i \in \{1, \ldots, n\}.$$

Here, each view V_{ij_k} is an element of the set $\mathcal{V}$.

By definition, the conjunctive query Q is equivalent to the union of the expansions of these queries $P_i, i \in \{1, \ldots, n\}$. Each expansion is a union of conjunctive queries, because each view V_{ij_k} may be a disjunctive view.

From [SY80], the query Q is equivalent to one of the expanded queries, which we denote by E_k, and the remaining expanded queries are contained in Q. Let E_k come from the query P_i in $P_1, \ldots, P_n$. Notice that P_i is an equivalent *conjunctive* rewriting of Q using $\mathcal{V}$. We modify $\mathcal{V}$ into $\mathcal{V}'$ by removing all the views that satisfy the following condition: there is no query in $\mathcal{Q}$ whose equivalent expanded query these views can contribute to. The resulting viewset $\mathcal{V}'$ provides an equivalent *conjunctive* rewriting of each query in $\mathcal{Q}$.

In the remainder of this paper we consider the problem of finding a minimum-size set $\mathcal{V}$ of disjunctive views for a given database D and workload of conjunctive queries $\mathcal{Q}$, assuming *conjunctive* rewritings of the queries $\mathcal{Q}$ using the views. From Theorem 1, all such viewsets are also minimum-size for D and $\mathcal{Q}$ assuming *disjunctive* rewritings of the workload queries.

3.2 Different Types of Views

There are two types of views in a rewriting of a query: (1) containment-target views, and (2) filtering views [ALU01,PL00]. 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* — unlike a filtering view — "covers" at least one query subgoal. Covering all query subgoals is enough to produce a rewriting of the query.

Example 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 that there are two containment mappings: one (an identity mapping) from Q to the expansion P^{exp} of P, and another from P^{exp} to Q.

View $V_1(D, M)$ covers the query subgoal $r(D, M)$, whereas view $V_2(D, C)$ covers the subgoal $s(D, C)$. Thus V_1 and V_2 are containment-target views for the query.

Definition 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.*

3.3 Containment-Target Views Are Enough

We now show that when looking for a minimum-size disjunctive viewset for a database and workload of conjunctive queries, we only need to consider containment-target views. In addition, there is a linear upper bound, in the size of the query workload, on the number of views in each such viewset.

Lemma 1 *Given a database D, for each conjunctive query workload $\mathcal{Q}$ and for each minimum-size disjunctive viewset $\mathcal{V}$ for $\mathcal{Q}$ and D, each view V in $\mathcal{V}$ such that $size(V, D) > 0$ is a containment-target view for at least one query in the workload $\mathcal{Q}$.*

Proof Suppose a set $\mathcal{V}$ of views is a minimum-size disjunctive viewset for a given conjunctive query workload $\mathcal{Q}$ and database D. Without loss of generality, we can assume that all views in $\mathcal{V}$ are *useful views* — that is, each view in $\mathcal{V}$ is present in all equivalent rewritings of at least one query in $\mathcal{Q}$ using $\mathcal{V}$. We now prove the lemma by assuming that there exists a useful view V in $\mathcal{V}$, $size(V, D) > 0$, that is not a containment-target view for any query in the workload $\mathcal{Q}$, and by showing that under this assumption we arrive at a contradiction. Indeed, let $\mathcal{Q}'$ be the (nonempty) set of queries Q' in the workload $\mathcal{Q}$, such that all equivalent rewritings of Q' in terms of $\mathcal{V}$ use the view V and, furthermore, all occurrences of V in all these rewritings are filtering views. By definition of filtering views, from each equivalent rewriting R' of Q' in terms of $\mathcal{V}$ we can obtain another *equivalent* rewriting R'' of Q' by removing all occurrences of the view V from R'. Thus, the viewset $\mathcal{W} = \mathcal{V} - \{V\}$ can be used to produce equivalent rewritings of all the queries in $\mathcal{Q}$. At the same time, the total size of the relations for the views in $\mathcal{W}$ on D is strictly smaller than the corresponding size for $\mathcal{V}$ (recall that $size(V, D) > 0$). Thus, we arrive at a contradiction by concluding that $\mathcal{V}$ is not a minimum-size viewset for $\mathcal{Q}$ and D.

Note 1. It follows from the proof of Lemma 1 that for each conjunctive query workload $\mathcal{Q}$ and for each database D, if there exists a finite minimum-size disjunctive viewset $\mathcal{V}$ for $\mathcal{Q}$ and D, then there exists a finite minimum-size disjunctive viewset $\mathcal{W}$ for $\mathcal{Q}$ and D, such that each view in $\mathcal{W}$ is a containment-target view for at least one query in $\mathcal{Q}$. Indeed, we can construct $\mathcal{W}$ by removing from $\mathcal{V}$ all views V such that (1) V is not a containment-target view for any query in $\mathcal{Q}$, and (2) $size(V, D) = 0$.

Theorem 2 *Given a database D, for each conjunctive query workload $\mathcal{Q}$ and for each minimum-size disjunctive viewset $\mathcal{V}$ for $\mathcal{Q}$ and D, if n is the total number of subgoals in all the queries in the workload $\mathcal{Q}$, then the viewset $\mathcal{V}$ has at most n views.*

Proof (sketch)

1. By Lemma 1, given a database, for each $\mathcal{Q}$ and $\mathcal{V}$ that satisfy the conditions of this theorem, each view in $\mathcal{V}$ is a containment-target view for at least one query in $\mathcal{Q}$.

2. For each query Q in $\mathcal{Q}$, to produce an equivalent rewriting of Q it is necessary to cover all the subgoals of Q using views.
3. In a minimum-size viewset $\mathcal{V}$, there is no need to have more than one (containment-target) view to cover each subgoal of each query in $\mathcal{Q}$.

It has been shown [LMSS95] that for each 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 optimality guarantees for the views in the rewriting.

Even though Lemma 1 says that in searching for a minimum-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 2 For an integer $k \geq 1$, 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 of these containment-target views is $2^k - 1$. Notice that the set of possible containment-target views for the workload $\{Q_k\}$ will be even larger if we also consider views with nondistinguished variables.

Based on Example 2, we make the following observation. The *length of a definition* of a query (or view) is the number of its subgoals.

Observation 1 *Given a database D and a workload $\mathcal{Q}$ of conjunctive queries, for the problem of finding a minimum-size viewset for $\mathcal{Q}$ and D, the size of the search space of views can be (at least) exponential in the length of the definitions of the queries.* □

3.4 Disjunctive Views Can Provide Better Solutions

Each conjunctive query Q has a unique minimal equivalent query Q', i.e., removing any of the subgoals in Q' will yield a query that is not equivalent to Q' [CM77]. We say a query has a *self-join* if its minimal equivalent query has at least two subgoals with the same relation name. In the rest of the paper, we assume that each query is already minimal. When workload queries have self-joins, we show that it might be better to materialize disjunctive views than conjunctive views.

Proposition 1 *There exists a query workload and a database, such that a solution with disjunctive views requires strictly less storage space than each solution using conjunctive views only.*

Proof We give the proof by constructing such an example. Let *P(A,B)* be a base table. A query workload $\{Q\}$ has a single query Q that asks for all P-paths of length two: Q: $ans(X, Y, Z)$:- $p(X, Y), p(Y, Z)$. We define a disjunctive view $V = V_1 \cup V_2$, where

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

In the definition of V, V_1 gives all pairs of attribute values for the "first hop" in a P-path of length two, whereas V_2 provides the pairs of values for all "second hops." Note that the relations for V_1 and V_2 will share tuples if there exist P-paths of length at least three.

For a database $D = \{p(1, 2), p(2, 1), p(3, 4)\}$, we show that the disjunctive viewset $\{V\}$ is smaller than each possible conjunctive viewset. The relation $V(D) = \{p(1, 2), p(2, 1)\}$ has four data values. Let $\mathcal{W}$ be an optimal *conjunctive* solution for the workload. By Lemma 1, $\mathcal{W}$ consists of conjunctive containment-target views only. We show that for the database D, the total number of occurrences of data values in $\mathcal{W}$ is greater than four.

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

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

By Theorem 2, R has no more than two view literals, i.e., $k \leq 2$. There are two cases: (1) each view w_i is used once in the rewriting R, or (2) some view w_j is used more than once in R. We consider the two cases separately.

Case 1: Each view w_i is used exactly once in rewriting R. From Theorem 3 in [CG00], each view in R can be defined as a subexpression of the query Q. Thus, all views in the rewriting R come from a set S of all conjunctive containment-target views that can be defined as subexpressions of the query Q; this set has just four views, and the relation for each view has at least four data values. By considering all combinations of the views in the set S that produce equivalent rewritings of Q and by computing the sizes of the resulting viewsets $\mathcal{W}$ on the database D, we obtain that for all such $\mathcal{W}$, $size(\mathcal{W}, D) > size(\{V\}, D) = 4$.

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

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

By Theorem 3.1 in [CHS02], in this case views in R can have more subgoals than the query Q. Suppose there exists a conjunctive view w that, after minimization, has more subgoals than the query Q. Then the minimized definition of w has at least three P-subgoals. In addition, the definition of w

cannot have cross-products; otherwise, the solution $\{w\}$ would be suboptimal for Q and for the database D. By considering all possible combinations of nontrivial equality join predicates on three P-subgoals, we verify that the three subgoals in w cannot contain a subset of subgoals of the query Q. In assuming that a view can be used in the rewriting R 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 is in the set S, see case 1). Then for each equivalent rewriting of the query Q that is a self-join of w, the viewset $\{w\}$, on the given database, has more data values than the disjunctive solution $\{V\}$.

3.5 Disjunctive Views and Rewritings: The Problem Is Decidable

In this section we show that the problem of finding a minimum-size set of *disjunctive* views for a database and workload of conjunctive queries is decidable.

Theorem 3 *Given a database D and a finite workload $\mathcal{Q}$ of conjunctive queries, we can construct a finite search space of views that includes all views in at least one minimum-size disjunctive viewset for $\mathcal{Q}$ and D. The number of views in the search space is at most triply-exponential in the sum of lengths of the definitions of the workload queries.*

Proof We show that for all conjunctive query workloads $\mathcal{Q}$ and databases D, to obtain all nontrivial disjunctive views in at least one minimum-size disjunctive viewset $\mathcal{U}$, we can consider rewritings of the queries in $\mathcal{Q}$ using sets of representative *conjunctive* views that we proceed to define; unions of some of those conjunctive views are the disjunctive views in $\mathcal{U}$. The proof of the theorem follows from this claim and from the bound we give on the number of such conjunctive views.

We first show that given a conjunctive query workload $\mathcal{Q}$ and a database D, there exists a minimum-size *conjunctive* viewset $\mathcal{V}$, such that each view V in $\mathcal{V}$ has at least one definition whose length is at most singly-exponential in the length of the longest query definition in the workload $\mathcal{Q}$.

Let $\mathcal{W}$ be a minimum-size set of conjunctive views for $\mathcal{Q}$ and D. Let $max_{\mathcal{Q}}|Q|$ be the length of the longest query definition in the workload $\mathcal{Q}$, and let $size(\mathcal{W}, D)$ be the total size of the relations for the views in $\mathcal{W}$ on the database D. Suppose some view in $\mathcal{W}$ has a definition whose length is more than singly-exponential in the length of the longest query definition in $\mathcal{Q}$. By Theorem 3.1 in [CHS02], for $\mathcal{Q}$, D, $\mathcal{W}$, and for a storage limit $size(\mathcal{W}, D)$, there exists another viewset, $\mathcal{V}$, with three properties: (1) for each view V in $\mathcal{V}$, the length of the definition of V is at most singly-exponential in $max_{\mathcal{Q}}|Q|$, (2) the relations for the views in $\mathcal{V}$ on the database D satisfy the storage limit $size(\mathcal{W}, D)$, and (3) for each query Q in $\mathcal{Q}$, there is an equivalent rewriting of Q in terms of $\mathcal{V}$. Moreover, by construction, there is a homomorphism between the views in $\mathcal{W}$ and the views in $\mathcal{V}$, such that each view V in $\mathcal{V}$ is contained in the respective view W in $\mathcal{W}$. By construction, $\mathcal{V}$ is a minimum-size viewset for the query workload $\mathcal{Q}$ on the database D.

For all minimum-size viewsets $\mathcal{W}$ for $\mathcal{Q}$ and D, we call the corresponding set $\mathcal{V}$ a *representative viewset* of $\mathcal{W}$ in the space $\mathcal{S}$ of all sets of views whose definition length is at most singly-exponential in the length of the longest query definition in $\mathcal{Q}$, in the following sense. (1) $size(\mathcal{V}, D) = size(\mathcal{W}, D)$ on D, and (2) for each query Q in $\mathcal{Q}$, a rewriting of Q using $\mathcal{V}$ is the result of replacing, in the rewriting of Q using $\mathcal{W}$, the literal for each view W in $\mathcal{W}$ by the literal for the image, in $\mathcal{V}$, of W under the homomorphism from $\mathcal{W}$ to $\mathcal{V}$. Because the set $\mathcal{S}$ is finite, it takes finite time to construct all representative viewsets of all minimum-size viewsets for a given $\mathcal{Q}$ and D.

We now prove the claim of the theorem, first for singleton query workloads and then for non-singleton workloads.

Case 1: Let the query workload $\mathcal{Q}$ have just one query Q, and let $\mathcal{V}$ be a minimum-size disjunctive solution for $\mathcal{Q}$. Consider a rewriting of the query Q using $\mathcal{V}$, and consider the expansion of the rewriting in terms of the conjunctive components $\mathcal{V}'$ of the views in $\mathcal{V}$. The expansion is a union of several conjunctive queries, and one of the queries, P, is equivalent to the query Q modulo the view definitions [SY80]. We call P the *equivalent conjunct* of Q in terms of $\mathcal{V}'$; let P^{exp} be the expansion of P in terms of the schema of the database D.

Suppose the viewset $\mathcal{V}$ contains exactly one nontrivial disjunctive view, V, that is a union of two conjunctive views, V_1 and V_2. Consider an alternative conjunctive solution $\mathcal{W}$ that is obtained by replacing, in $\mathcal{V}$, this disjunctive view V by V_1 and V_2. $\mathcal{W}$ 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 $\mathcal{W}$ if $size(\mathcal{V}, D) < size(\mathcal{W}, D)$ on the database D. Note that $size(\mathcal{V}, D)$ can be less than $size(\mathcal{W}, D)$ only when the equivalent conjunct P (of Q in terms of $\mathcal{V}'$) has a join of V_1 and V_2. (If P has either just one conjunct of V, or if P has a self-join of just one conjunct of V, we could obtain a conjunctive solution $\mathcal{W}'$ for the query Q that is at least as good on the database D as $\mathcal{V}$, by replacing, in $\mathcal{V}$, the view V by that one conjunct.)

Next, we show that we can construct, from the expansion P^{exp} of the equivalent conjunct P of Q, a disjunctive view V', such that (1) V' is used in the same way as V in rewriting Q, and (2) $size(\{V'\}, D) \leq size(\{V\}, D)$ on D. The view V' is a union of the views V_1' and V_2', which are the representatives of the views V_1 and V_2 in the space $\mathcal{S}$ of all conjunctive views whose length is at most singly-exponential in the length of the query Q. From the properties of representative views, the result of replacing V with V' in the viewset $\mathcal{V}$ is at least as good a solution for $\{Q\}$ and D as the viewset $\mathcal{V}$. We can find the view V' by testing all unions of two views in $\mathcal{S}$; thus, we can construct a search space of all disjunctive views that are part of at least one optimal solution for $(\{Q\}, D)$.

We obtain V' from P^{exp} as follows. By Theorem 2, the maximal number of view subgoals in P is the number n of subgoals in the query Q. We take all partitions of the subgoals of P^{exp} into up to n parts; for each partition that has k elements, we design k conjunctive views whose bodies are the elements of the partition; we consider each resulting conjunctive viewset $\mathcal{T}$ that gives an equivalent rewriting of Q, such that P^{exp} is the expansion of the rewriting. (In particular, this procedure gives us the viewset $\mathcal{W}$.) Now, for each $\mathcal{T}$, we construct its representative viewset $\mathcal{T}'$, including the representative viewset of $\mathcal{W}$. On the database D, $size(\mathcal{T}', D) \leq size(\mathcal{T}, D)$, and the number of subgoals in each view in $\mathcal{T}'$ is at most singly-exponential in the number n of subgoals of the query Q.

Recall that, under our assumption, $size(\mathcal{V}, D) < size(\mathcal{W}, D)$. We now construct, from each viewset $\mathcal{T}'$, all disjunctive views T that are each a union of two elements of $\mathcal{T}'$. By construction, from the representative viewset of $\mathcal{W}$ we obtain a disjunctive view V' that is contained in V and can replace V in the rewriting of Q in terms of $\mathcal{V}$. Because $\mathcal{V}$ is a minimum-size viewset for $(\{Q\}, D)$, the result of replacing V by V' in $\mathcal{V}$ is also a minimum-size viewset for $(\{Q\}, D)$.

Observe that each such view V' can be generated by taking a union of two conjunctive views in $\mathcal{S}$, and that the process of generating all such views V' is finite because the number of views in $\mathcal{S}$ is finite. From this observation, we obtain the claim for Case 1 of the theorem. (The proof is extended in a straightforward way to cases where the viewset $\mathcal{V}$ has more than one disjunctive view and where a disjunctive view in $\mathcal{V}$ can have more than two conjuncts.)

Case 2. Suppose the query workload $\mathcal{Q}$ has at least two queries: $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$, $m \geq 2$. 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 ($i \in \{1, \ldots, m\}$) in the workload $\mathcal{Q}$,
- if necessary, we rename the variables in the conjuncts $P_1, \ldots, P_m$, to avoid using a 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 & ... & P_m, and the head of $\mathcal{P}$ comprises all head variables of $P_1, \ldots, P_m$.

By applying the reasoning in Case 1 to the individual conjuncts P_i in $\mathcal{P}$, we show that there exists a rewriting $\mathcal{P}'$ that is equivalent to $\mathcal{P}$ on D, whose view relations have the same size on D as the relations for minimum-size disjunctive viewsets for $\mathcal{P}$, and such that that the number of subgoals of each disjunctive view for $\mathcal{P}'$ is at most singly-exponential in the sum of the lengths of the queries in the workload $\mathcal{Q}$.

For each minimum-size disjunctive viewset $\mathcal{V}$ for the workload $\mathcal{Q}$ and assuming conjunctive rewritings only, $\mathcal{V}$ is also a minimum-size disjunctive viewset for the conjunction $\mathcal{P}$. Using the reasoning in Case 1 above, we can find all representative 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.

By construction of the viewset $\mathcal{V}'$ from $\mathcal{V}$ in Theorem 1, $size(\mathcal{V}', D) \leq size(\mathcal{V}, D)$ for each database D. It follows that for an arbitrary conjunctive query workload $\mathcal{Q}$ and database D, we can find at least one minimum-size disjunctive viewset for $(\mathcal{Q}, D)$ by searching among only those sets of views that provide equivalent *conjunctive* rewritings of all queries in $\mathcal{Q}$. Thus, the following result is a direct consequence of Theorems 1 and 3:

Corollary 1 *For each finite workload $\mathcal{Q}$ of conjunctive queries and a database D, the problem of finding a minimum-size disjunctive viewset for $\mathcal{Q}$ and D is decidable.*

Note 2. We observe that the problem has a triply-exponential upper bound: A naive algorithm will find a minimum-size viewset for a given query workload and database by exploring all subsets of the at most doubly-exponential search space of views.

4 Conjunctive 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 minimum-size set of disjunctive views. We first show that for a single query without self-joins, we need to consider only conjunctive views, because nontrivial disjunctive views do not add 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). For arbitrary workloads of conjunctive queries without self-joins, we show that, given a database, the problem of finding an optimal disjunctive viewset is in NP, because we can construct a rewriting of each workload query by using a small number of conjunctive views with a bounded number of subgoals (Section 4.4). We also study the case of a single conjunctive query with arithmetic comparisons (Section 4.5), and define the notion of a view associated with a set of relations (Section 4.6), which will be used in later sections.

4.1 Conjunctive Views Are Enough for a Single Conjunctive Query

Theorem 4 *Suppose a set $\mathcal{V}$ of disjunctive views is a solution for a given database $\mathcal{D}$ and a single conjunctive query Q without self-joins. Then there exists another solution $\mathcal{V}'$ for $\mathcal{D}$ and $\{Q\}$, such that all views in $\mathcal{V}'$ are conjunctive, and $size(\mathcal{V}', D) \leq size(\mathcal{V}, D)$.*

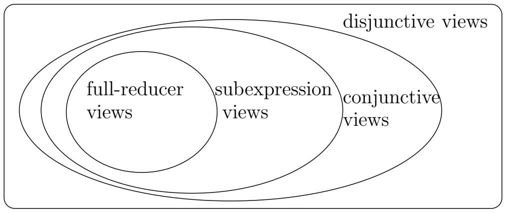

Fig. 3 View space for a single conjunctive query without self-joins.

Proof Let $\mathcal{Q}$ be a singleton query workload, $\mathcal{Q} = \{Q\}$, where the query

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

is conjunctive and does not have self-joins. Let $\mathcal{D}$ be an arbitrary database, 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 $\mathcal{P}$. 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 idea of the proof is that there exists a transformation μ of the view V that produces a new disjunctive view:

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

where $V_i' = \mu(V_i)$, $i = 1, \ldots, k$. The new disjunctive view V' is used in a new equivalent rewriting $\mathcal{P}'$ of the query Q. We show that for each conjunctive component V_i' of the view V', V_i' can replace the *entire* 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 $\bar{X}$, $\bar{X}_1$, $\ldots$, $\bar{X}_m$ is a list of arguments, and G represents the instances of other views that are not V. By replacing all the disjunctive views by their definitions as unions of conjuncts, we get a union of conjunctive queries, which is equivalent to the query Q. From [SY80], at least one of these conjunctive queries — we denote it by P — is equivalent to Q: $P^{exp} \equiv Q$, where P^{exp} is an expansion of P [CM77]. 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 $\bar{X}' = \mu(\bar{X})$, $\bar{X}_i' = \mu(\bar{X}_i)$, $i = 1, \ldots, m$. G' represents the subgoals in $\mathcal{P}'$ that do not use 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, $\mathcal{P}' = \mu(\mathcal{P})$ can be represented as:

$$\mathcal{P}' : ans(\bar{X}') \coloneq \hat{V}_1(\bar{X}'_1), \ldots, \hat{V}_m(\bar{X}'_m), G'.$$

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 the conjunctive components $\hat{V}_2, \ldots, \hat{V}_m$ of V:

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

Let β be a containment mapping from Q to H^{exp}. For each $j \geq 2$, 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 the subgoals of Q do not come from the expansion of each $\hat{V}_1(\bar{X}'_j)^{exp}$, $j \geq 2$.

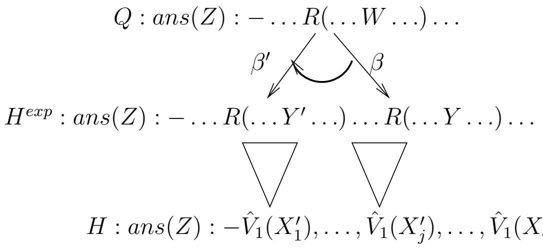
$Q : ans(Z) : - \ldots R(\ldots W \ldots) \ldots$
β'
β
$H^{exp} : ans(Z) : - \ldots R(\ldots Y' \ldots) \ldots R(\ldots Y \ldots) \ldots$
$H : ans(Z) : -\hat{V}_1(X'_1), \ldots, \hat{V}_1(X'_j), \ldots, \hat{V}_1(X'_m), G$

Fig. 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 whose l-th attribute appears in the head of $\hat{V}_1$. 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 — which used to be to W in the expansion of $\hat{V}_1(\bar{X}'_j)$ — to 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 a fresh variable 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

[1] Note that we cannot do the redirection in this manner if the relation R appears more than once in Q; see proof of Proposition 1 in Section 3.4.

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 [PL00, ALU01]. So we can redirect the mappings — which used to be to Y in $\hat{V}_1(\bar{X}'_j)^{exp}$, $j \geq 2$ — to Y' in $\hat{V}_1(\bar{X}'_1)^{exp}$.

By redirecting all the local mappings from the expansion of $\hat{V}_1(\bar{X}'_j)$, $j \geq 2$, 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 containment mapping β' implies that $H^{exp}_{new} \sqsubseteq Q$. Since $\mathcal{P}'$ is an equivalent rewriting of Q, we obtain that Q is contained in the expansion of $\mathcal{P}'$. In addition, the expansion of $\mathcal{P}'$ is also contained in H^{exp}_{new}, since the subgoals in H_{new} are a subset of those in $\mathcal{P}'$. 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 $\mathcal{V}$.

We now show that we cannot extend the result of Theorem 4 to non-singleton query workloads. Consider an example.

Example 3 Consider the problem of finding a minimum-size viewset for a database $D = \{p(1,2),\ p(3,2),\ p(1,4),\ p(3,4)\}$ and a query workload $\mathcal{Q} = \{Q_1, Q_2\}$:

$Q_1 : ans(X,Y) \ :-\ p(X,Y),\ X = 1.$
$Q_2 : ans(X,Y) \ :-\ p(X,Y),\ Y = 2.$

The size of the database is 32 bytes — 4 tuples each of size 8 bytes for P. (We assume that storage of an integer takes four bytes.) The answer to Q_1 on the database D is $\{(1,2),\ (1,4)\}$, and the answer to Q_2 on D is $\{(1,2),\ (3,2)\}$. Thus, the total size of the answers to the workload queries on D is $(2 \times 4) \times 4 = 32$ bytes. It is easy to see that for each set $\mathcal{U}$ of *conjunctive* views that is useful in rewriting the workload queries, $size(\mathcal{U}, D) \geq 32$ bytes.

Now consider a disjunctive view V:

$V : ans(X,Y) \ :-\ q_1(X,Y).$
$V : ans(X,Y) \ :-\ q_2(X,Y).$

The view $V(X,Y)$ can be used to answer each query in $\mathcal{Q}$:

$R_1 : ans(X,Y) \ :-\ v(X,Y),\ X = 1.$
$R_2 : ans(X,Y) \ :-\ v(X,Y),\ Y = 2.$

The answer to V on D is $\{(1,2),\ (3,2),\ (1,4)\}$ and is thus of size 24 bytes. Thus, $size(\{V\}, D) < size(\mathcal{U}, D)$ for each minimum-size conjunctive viewset $\mathcal{U}$ for the pair $(\mathcal{Q}, D)$.

4.2 Subexpression-Type Views Are Enough for a Single Conjunctive Query

We saw in the proof of Theorem 3 that to find a minimum-size conjunctive viewset for a workload $\mathcal{Q}$ of conjunctive queries, it is enough to consider views with definitions whose size is at most exponential in the length of the longest query definition in $\mathcal{Q}$. Now we show that for *singleton* workloads of queries without self-joins, we can further reduce the size of the search space of views by considering only those views whose definitions are subexpressions of the query. 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. In Section 4.4 we will see that, given a database and a *non-singleton* workload of queries without self-joins, all views in at least one optimal viewset can be defined as unions of conjunctive subexpression-type views for the individual workload queries.

Definition 2 *A conjunctive view V is a* subexpression-type view *for a conjunctive query Q if, for some definition of V, there exists a containment mapping to the body of the definition from a subset of subgoals of Q.*

Theorem 5 *Given a database D and a conjunctive query Q without self-joins, there is a minimum-size conjunctive viewset $\mathcal{U}$ for D and $\{Q\}$, such that each view in $\mathcal{U}$ is a subexpression-type view for some query in $\mathcal{Q}$.*

Proof (sketch) Suppose a viewset $\mathcal{V}$ is an arbitrary conjunctive viewset, such that the query Q has a rewriting using $\mathcal{V}$. From $\mathcal{V}$ we can construct another viewset, $\mathcal{W}$, such that $\mathcal{W}$ also provides a rewriting of 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 *all* databases, and each view in $\mathcal{W}$ is defined as a subexpression of the query Q, with possibly attributes projected. Thus, when looking for minimum-size conjunctive viewsets $\mathcal{U}$ for the given Q and an arbitrary database D, we can restrict our consideration to views that can be defined using subexpressions of the query Q.

Let R be an equivalent rewriting of the query Q using views $\mathcal{V}$, $R \equiv Q$. The main steps in the construction of $\mathcal{W}$ from $\mathcal{V}$ are as follows.

1. Without loss of generality, by Lemma 1 we can assume that all views in R are containment-target views of Q.
2. From R we construct a new equivalent rewriting R' of Q, such that R' does not have self-joins of a view. The construction of R' from R is done by redirecting variable and subgoal mappings from Q to R^{exp}, as described in the proof of Theorem 4.
3. From each view V in R' we can construct a view W, such that the query defining W is contained in the query that defines V. (We construct the W's from the corresponding V's using two arbitrary fixed containment mappings, μ' from Q to $R^{'\ exp}$ and ν' from $R^{'\ exp}$ to Q, by (1) finding the image $\bar{V}'$ in Q of the expansion of each subgoal $\bar{V}$ of R' under ν', and by then (2) using the image of $\bar{V}'$ in $R^{'\ exp}$ under μ' to define the view W.)

4. By Theorem 3 in [CG00], by replacing each view V in R' with the corresponding W, we obtain a new *equivalent* rewriting R'' of Q, in terms of the viewset $\mathcal{W} = \{W\}$. The properties of $\mathcal{W}$ that we mention in the beginning of this proof hold by construction of $\mathcal{W}$ from $\mathcal{V}$.

4.3 Full-Reducer Views Are Enough for a Single Conjunctive Query

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 [Yan81]. (In Section 4.4 we show that for non-singleton query workloads, full-reducer views are not always part of an optimal solution.) Consider Example 2 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 dangling tuples after the replacement. At the same time, for each such view, there exists a database where the view is part of some minimum-size viewset for the query Q_k. We formalize these observations in the following result.

Lemma 2 *Given a database D, consider a conjunctive query Q without self-joins; let $\mathcal{V}$ be a solution for Q and D. 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 D, and $size(\mathcal{V}', D) \leq size(\mathcal{V}, D)$.*

Proof For a conjunctive query Q without self-joins and for its conjunctive solution $\mathcal{V}$, consider a 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 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 each database we have $size(W, D) \leq size(V, D)$. 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, as the containment mappings between Q and the expansion of P can be extended to mappings between Q and the expansion of P'.

Theorem 6 *Given a database D and a conjunctive query Q without self-joins, there exists a minimum-size conjunctive viewset $\mathcal{V}$ for Q and D, such that each view in $\mathcal{V}$ is a full-reducer containment-target view.*

4.4 Workloads of Queries without Self-Joins: The Problem Is in NP

Recall (see Example 3 in Section 4.1) that for non-singleton workloads of queries without self-joins, it is not enough to consider conjunctive views when searching for a minimum-size viewset for the workload and some database. Nevertheless, we show in this section that the view-selection problem is in NP for arbitrary workloads of queries without self-joins.

We first observe that in rewriting arbitrary workloads of conjunctive queries without self-joins using disjunctive views, we do not need to consider rewritings that have self-joins of views; the proof follows immediately from the proof of Theorem 4.

Proposition 2 *Let $\mathcal{Q}$ be an arbitrary workload of conjunctive queries without self-joins, and let $\mathcal{V}$ be a disjunctive viewset such that each query in $\mathcal{Q}$ has an equivalent rewriting using $\mathcal{V}$. Then for each query in $\mathcal{Q}$ there exists an equivalent rewriting R without self-joins of the views in $\mathcal{V}$.*

We now show that when looking for minimum-size viewsets for nonsingleton workloads of queries without self-joins, we can restrict our consideration to those disjunctive views whose each conjunctive component is a subexpression-type view for some workload query.

Theorem 7 *Given a database D and a workload $\mathcal{Q}$ of conjunctive queries without self-joins, there is a minimum-size disjunctive viewset $\mathcal{U}$ for D and $\mathcal{Q}$, such that each conjunctive element U of each view in $\mathcal{U}$ is a subexpression-type view for some query in $\mathcal{Q}$.*

Proof (sketch) From Proposition 2 and from the proof of Theorem 4 it follows that in each rewriting R without self-joins of each query in $\mathcal{Q}$, each occurrence in R of each disjunctive view $V \epsilon \mathcal{V}$ can be replaced by a single conjunctive component of V. From Theorem 5, each such conjunctive component is a subexpression-type view for some query in $\mathcal{Q}$.

Observe that we cannot strengthen the statement of Theorem 7 by replacing "subexpression-type view" with "full-reducer view." Consider an example.

Example 4 Consider the problem of finding a minimum-size viewset for a database $D = \{p(1,2),\ p(12,13),\ s(2,4,9),\ s(2,5,10),\ s(6,4,11),\ t(4,7),\ t(14,15)\}$ and a query workload $\mathcal{Q} = \{Q_1, Q_2\}$:

$$Q_1 : ans(X,Y,Z) \quad :-\ p(X,Y),\ s(Y,Z,T).$$
$$Q_2 : ans(Y,Z,W) \quad :-\ s(Y,Z,T),\ t(Z,W).$$

The size of the database is 68 bytes. (We assume that storage of an integer takes four bytes.) The answer to Q_1 on the database D is $\{(1,2,4),\ (1,2,5)\}$, and the answer to Q_2 on D is $\{(2,4,7),\ (6,4,7)\}$. Thus, the total size of the answers to the workload queries on D is $(2 \times 6) \times 4 = 48$ bytes. It is easy to see that for each set $\mathcal{U}$ of conjunctive full-reducer views that is useful in rewriting the workload queries, $size(\mathcal{U}, D) \geq 48$ bytes. In addition, unions of such conjunctive full-reducer views do not produce nontrivial disjunctive views that would be useful in equivalently rewriting the workload queries.

Now consider a set of conjunctive views $\mathcal{V} = \{V_1, V_2, V_3\}$:

$V_1 : ans(X, Y) \ :- \ p(X, Y), \ s(Y, Z, T).$
$V_2 : ans(Y, Z) \ :- \ s(Y, Z, T).$
$V_3 : ans(Z, W) \ :- \ s(Y, Z, T), \ t(Z, W).$

Each view in $\mathcal{V}$ is a subexpression-type view for either Q_1 or Q_2; view V_2 is not a full-reducer view for either query.

The viewset $\mathcal{V}$ can be used to answer each query in $\mathcal{Q}$:

$R_1 : ans(X, Y, Z) \ :- \ v_1(X, Y), \ v_2(Y, Z).$
$R_2 : ans(Y, Z, W) \ :- \ v_2(Y, Z), \ v_3(Z, W).$

The total size of the answers to the views $\mathcal{V}$ on D is 40 bytes, which is less than the size of each set of full-reducer views for $\mathcal{Q}$ on D.

Theorems 4 through 7, together with the results in Section 3, imply that to find a minimum-size disjunctive viewset for a database and a workload of conjunctive queries without self-joins, it is enough to consider conjunctive subexpression-type views without self-joins. As a result, we have reduced the size of the search space of conjunctive views that includes all conjunctive components of all views in some minimum-size disjunctive viewset, from doubly-exponential to singly-exponential in the size of the (singleton or non-singleton) query workload.

Corollary 2 *For all databases D and all workloads $\mathcal{Q}$ of conjunctive queries without self-joins we can construct a finite search space of views that includes all conjunctive components of all views in at least one minimum-size disjunctive viewset for $\mathcal{Q}$ and D, such that the number of views in the search space is at most singly-exponential in the size of the queries in $\mathcal{Q}$.*

We conclude with a complexity result for view selection for arbitrary workloads of conjunctive queries without self-joins:

Theorem 8 *Given a database D and an arbitrary workload $\mathcal{Q}$ of conjunctive queries without self-joins, the decision version of the problem of finding a minimum-size disjunctive viewset for $\mathcal{Q}$ and D is in NP.*

Proof Consider a workload $\mathcal{Q}$ of queries with at most n subgoals, a database $\mathcal{D}$, and an integer K. To check whether a disjunctive viewset $\mathcal{V}$ is a solution for $\mathcal{Q}$, 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 each query in $\mathcal{Q}$, 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 Section 3 and in this section, 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 workload $\mathcal{Q}$.

4.5 Queries with Arithmetic Comparisons

We can extend Theorem 6 to queries without self-joins and with arithmetic comparisons. We assume that (1) Each inequality comparison is between a variable and a constant, such as `Year >= 1999` or `price < 5000`; each comparison operator is one of $\{<, \leq, >, \geq, \neq\}$. (2) The attribute of the variable in each comparison is over a densely totally ordered domain. (3) The comparisons are not contradictory, that is, there exists an instantiation of the variables such that all the comparisons are true. (4) All the comparisons are safe, that is, each variable in the comparisons appears in some ordinary (relational) subgoal.

Theorem 9 *Given a database D and a conjunctive query Q without self-joins that may have arithmetic comparisons between variables and constants, there exists a minimum-size viewset $\mathcal{V}$ for Q and D, such that each view in $\mathcal{V}$ is a full-reducer conjunctive view, possibly with arithmetic comparisons with constants in the query. This result holds assuming conjunctive views and rewritings that may have arithmetic comparisons.*

Before giving the proof, we first revisit related results on testing containment of two conjunctive queries with arithmetic comparisons [GSUW94, Klu88]. Let Q_1 and Q_2 be two conjunctive queries with arithmetic comparisons. We *normalize* both queries as follows.

- For all occurrences of a shared variable X in the ordinary subgoals except the first occurrence, replace the occurrence of X by a new distinct variable X_i, and add $X = X_i$ to the comparisons of the query; and
- For each constant c in an ordinary subgoal, replace the constant by a new distinct variable Z, and add $Z = c$ to the comparisons of the query.

Theorem 10 *([GSUW94, Klu88]) Let Q_1 and Q_2 be two normalized conjunctive queries with arithmetic comparisons. Let β_1 and β_2 be the comparisons in Q_1 and Q_2, respectively. Let $\mu_1, \ldots, \mu_k$ be all the containment mappings from the ordinary subgoals of Q_1 to the ordinary subgoals of Q_2. Then $Q_2 \sqsubseteq Q_1$ if and only if β_2 can logically imply $\mu_1(\beta_1) \vee \ldots \vee \mu_k(\beta_1)$.*

Now we give the proof of Theorem 9.

Proof Let $\mathcal{W} = \{W_1, \ldots, W_k\}$ be a set of conjunctive views with arithmetic comparisons that is an arbitrary (not necessarily optimal) solution for a given query workload $\{Q\}$ and database D. Let P be an equivalent rewriting (with comparisons) of the query Q using these views. That is, P^{exp}, the expansion of P, is equivalent to Q. We construct from $\mathcal{W}$ a new set $\mathcal{V} = \{V_1, \ldots, V_k\}$ of views, such that (1) each view in $\mathcal{V}$ is a full-reducer conjunctive view, possibly with comparisons with constants, (2) we can equivalently rewrite Q using $\mathcal{V}$ as a conjunctive query with comparisons, and (3) the size of $\mathcal{V}$ on the database D does not exceed the size of $\mathcal{W}$. It follows that at least one optimal viewset for $\{Q\}$ on D is a set of full-reducer views for Q.

We normalize queries Q and P^{exp} to $\hat{Q}$ and $\hat{P}^{exp}$, respectively. Since $\hat{Q}$ does not have self-joins, each relation appears in $\hat{Q}$ only once. Therefore, for each ordinary subgoal g of $\hat{P}^{exp}$, there can be only one subgoal in $\hat{Q}$ to which g can be mapped in all containment mappings from the ordinary subgoals of $\hat{P}^{exp}$ to the ordinary subgoals of $\hat{Q}$. So there can be at most one such containment mapping. By Theorem 10, there must be *one and only one* such containment mapping μ, such that

$$\beta_2 \Rightarrow \mu(\beta_1),$$

where β_2 and β_1 are the comparisons in $\hat{Q}$ and $\hat{P}^{exp}$, respectively, and "$\Rightarrow$" stands for logical implication.

We do the following modifications on $\hat{P}^{exp}$ and the corresponding modifications on the views used in P:

1. We apply the mapping μ on those variables in $\hat{P}^{exp}$ that are not fresh during the expansion. Correspondingly we apply the mapping on the definition of each view. After this application, if a view is used more than once in the rewriting, then their instances in the new rewriting become the same, because Q does not have self-joins and the mapping is unique. Let $W_1^{(1)}, \ldots, W_k^{(1)}$ be the new views, and $E^{(1)}$ be the new expansion of the rewriting, which has comparisons $\mu(\beta_1)$.
2. Each comparison in β_2 is either $X = Y$ or $X \; op \; c$, where X and Y are variables, c is a constant, and op is a comparison operator. Since $\beta_2 \Rightarrow \mu(\beta_1)$, each comparison in $\mu(\beta_1)$ must also be either $Z \; op \; U$ or $Z \; op \; d$, where Z and U are variables and d is a constant. We replace the comparisons in the expansion $E^{(1)}$ with comparisons in β_2. Clearly the logical implication $\beta_2 \Rightarrow \beta_2$ is still true. We add these comparisons in β_2 to all the views whose expansion uses a variable in the comparisons. Let $W_1^{(2)}, \ldots, W_k^{(2)}$ be the new views, and $E^{(2)}$ be the new expansion of the rewriting.
3. For each view $W_i^{(2)}$, if its body does not have a relation used in Q, we add this relation to this view's body. In addition, if there is a condition $X \; op \; U$ in Q, where X is a variable, and U is a variable or a constant, we add the corresponding condition to the view's body. We do the modification for all the views. We can show that for these modified views, we can add their additional subgoals and conditions to their expansions in $E^{(2)}$. In addition, there is a containment mapping μ' from the ordinary subgoals of the new expansion, such that $\beta_2 \Rightarrow \mu'(\beta_2')$, where β_2' is the comparisons in the new expansion.

Notice that each modification makes the views more contained, and the final expansion more contained (due to the additional conditions). In addition, the final expansion still contains Q. Therefore, we obtain an equivalent rewriting of the query Q using a new set of views $\mathcal{V} = \{V_1, \ldots, V_k\}$, such that (1) each V_i is constructed from the corresponding view W_i in $\mathcal{W}$, (2) each view in $\mathcal{V}$ satisfies the conditions in the theorem, and (3) the size of each V_i on the database D does not exceed the size of W_i on D.

4.6 View Associated with a Subset of Relations

Finally, we observe that to find an optimal solution for a single conjunctive query, it is enough to consider those full-reducer views whose head projects only "necessary" body attributes, that is, just head attributes of the rewriting and the attributes required for joins with the remaining views. Formally, let G be an arbitrary subset of the relations in the query. Let G' be the remaining relations in Q. We say an attribute A of a relation in G is a *relevant* attribute of G if either (1) A is an output (i.e., distinguished) attribute of Q, or (2) A is a join attribute in the query, i.e., in Q, this attribute appears in a subgoal of a relation in G and a subgoal of a relation in G'. A full-reducer view V_G is called the *associated view of* G if the head arguments $\bar{Y}_G$ of V_G are exactly the relevant attributes of G.

For example, the view V_1 in Figure 2 is associated with the subset G_1={`customer`, `order`}. The relevant attributes of the view V_1 include output attributes of the query Q_1 (Figure 1) — `c.name, o.orderdate, o.shippriority, o.comment` — as well as a join attribute `o.orderkey`. Table 2 shows all possible subsets of the relations in the query Q_1 that need to be considered in searching for an optimal rewriting of the query Q_1.

Table 2 Possible views for query Q_1.

Relation subset	Associated view
$\{customer\}$	W_1
$\{order\}$	W_2
$\{lineitem\}$	$W_3 = V_2$
$\{customer, order\}$	$W_4 = V_1$
$\{customer, lineitem\}$	W_5
$\{order, lineitem\}$	W_6
$\{customer, order, lineitem\}$	$W_7 = Q_1$

In the rest of this paper, when the query is given, unless otherwise specified, if $\{R_1, \ldots, R_k\}$ is a set of relations, we use "$V\{R_1, \ldots, R_k\}$" to represent the corresponding full-reducer view associated with this subset. For instance, for query Q_1, we use "$V\{customer\}$" to represent the view W_1.

5 Computing a Viewset Efficiently

So far we have studied the problem of constructing search spaces of views that contain a minimum-size solution for a query workload and the corresponding complexity issues. In this section we study how to compute such a viewset efficiently. The efficiency is especially important for cases where we need to compute a good solution online. We focus on the case of a single conjunctive query without self-joins. We study issues related to computing a viewset, including pruning rules to reduce the number of solutions and views that need to be considered, estimating the size of each view, and heuristics for finding a solution.

5.1 Exhaustive-Search Algorithm

Consider a single conjunctive query Q without self-joins. An exhaustive-search algorithm for finding a minimum-size set of conjunctive views would consider all partitions of the relational subgoals in the query Q. Each partition $T = \{G_1, \ldots, G_m\}$ yields a decomposition plan (called *solution*) $P_T = \{V_{G_1}, \ldots, V_{G_m}\}$, where each view V_{G_i} is the associated view of G_i, see Section 4.6. The algorithm searches for a minimum-size partition and generates the corresponding decomposition plan.

If the query has N relational subgoals, the total number of plans considered by the basic search algorithm is $\sum_{i=1}^{N} S(N, i)$, where $S(m, n)$ is the Stirling number that represents the number of ways of partitioning a set of N elements into i nonempty sets. For instance, this total number is 52 for $N = 5$, and is 203 for $N = 6$. Since the number of relational subgoals in many queries tends to be small, we could afford to use this algorithm to search for an optimal solution. In Section 5.4 we will study how to find a solution using heuristics when the number N is large. Notice that even without heuristics, the size of the search space here is dramatically smaller than the number of physical plans considered by a standard System-R-style query optimizer for relational database systems [SAC$^+$79], since we consider the total number of partitions of (the relational) subgoals, while traditional optimizers also need to consider other factors such as different join methods and join orders.

5.2 Reducing the Numbers of Views and Solutions

We develop pruning rules to reduce the number of views and solutions that need to be considered while still allowing us to find an optimal plan. These rules are based on specific properties of the query, such as its shape, output attributes, and possible key constraints where join conditions involve keys. We use the following simplified schemas of the TPC-H relations. The underlined attribute(s) of each relation form(s) a primary key for this relation.

```
part(partkey,mfgr,type,size)
partsupp(partkey,suppkey,availqty)
supplier(suppkey,name,nationkey)
nation(nationkey,name,regionkey)
region(regionkey,name)
```

We use the query Q_2 in Figure 5, which is a slight variation on “Query 2” in the TPC-H benchmark [TPC]. Recall (Section 4.3) that a full-reducer view is associated with a subset of the relations used in the query.

```
SELECT   p.mfgr, ps.partkey, ps.suppkey, s.name, n.name
FROM     part p, supplier s, partsupp ps, nation n, region r
WHERE    p.partkey = ps.partkey AND s.suppkey = ps.suppkey
         AND p.size = 24 AND s.nationkey = n.nationkey
         AND n.regionkey = r.regionkey AND r.name = 'AMERICA';
```

Fig. 5 Query Q_2

We now review the concept of "RI-join graphs" that will be used in the pruning rules. Formally, the *RI-join graph* of a query Q is a directed graph $G(Q) = (V, E)$, in which the set of vertices V represents the set of relations used in Q. There is a directed edge $R_i \rightarrow R_j$ in E if the query has a join condition $R_i.B = R_j.A$, and B is a foreign key referring to $R_j.A$.[2]

The RI-join graph of the query Q_2 is shown in Figure 6. The edge `nation` → `region` in the graph represents the fact that there is a join condition `nation.regionkey = region.regionkey`, and `nation.regionkey` is a foreign key referring to `region.regionkey`.

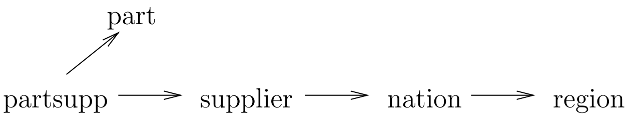

Fig. 6 RI-Join graph $G(Q_2)$ of query Q_2.

Because the `FROM` clause of the query Q_2 has five relations, the number of plans for Q_2 considered by the exhaustive-search algorithm is $\sum_{i=1}^{5} S(5, i) = 52$. The number of views considered by the algorithm is $2^5 - 1 = 31$.

5.2.1 Pruning Rule 1: Ignoring plans that use "disconnected" views Using this pruning rule, we never consider one class of views, which we call "disconnected views." As discussed in Section 4.6, given a query, we associate one view with each subset G of the relations. *Disconnected views* correspond to those subsets G that can be partitioned into two groups of relations, $G^{(1)}$ and $G^{(2)}$, such that the query does not have join conditions between the relations in the two groups. Rather than considering a disconnected view for a subset G of relations, we can always project the answer to the extended query on the relevant attributes of the groups $G^{(1)}$ and $G^{(2)}$ *separately.* In addition, in each decomposition plan for the query, the two views for the groups $G^{(1)}$ and $G^{(2)}$, taken together, have the same functionality as the disconnected view for the subset G. After we apply this rule on Q_2, we still have $2^4 = 16$ possible plans, with $\sum_{i=1}^{5} i = 15$ views.

[2] "RI" stands for "Referential Integrity." The concept of "RI-join graph" is similar to the concept of "join graph" introduced in [QGMW96], which requires that A is a key of R_j, not necessarily a foreign key.

5.2.2 Pruning Rule 2: Removing ear relations without output attributes Let us look at the `region` relation in query Q_2. This relation does not have output attributes in Q_2, and its only join condition `r.regionkey = n.regionkey` involves another relation, `nation`. The only contribution of `region` to the query is in providing a selection condition `r.name = 'AMERICA'` and a join condition `n.regionkey = r.regionkey`. Therefore, we can ignore the `region` relation when enumerating relation subsets for views. We further reduce the number of plans for Q_2 that need to be searched to $2^3 = 8$, with $\sum_{i=1}^{4} i = 10$ views.

In general, we look for *ear relations* that do not have output attributes in Q. A relation is an ear relation if all its join attributes are shared with just one other relation [Gra79]. If an ear relation does not have output attributes, then its only contribution to the query's results is to provide selection or join conditions in the query. After applying these conditions, we can ignore this relation when generating relation subsets for views. (Notice that the ignored relation's conditions still remain in Q; we just do not consider it when enumerating relation subsets.) By ignoring an ear relation R, we could make another relation R' an ear relation in a more general sense, i.e., R' does not have output attributes of Q, and all its join attributes are shared with either R or one other remaining relation. We repeat this process until we cannot eliminate more ear relations that do not have output attributes. Note that we may not be able to remove those relations that do not have output attributes but are connected to more than one other relation, since those relations could be connecting two other relations.

5.2.3 Pruning Rule 3: Using Key Constraints This rule is based on key constraints in the query relations. We use query Q_2 to illustrate the rule. After ignoring relation `region` in enumerating relation subsets (pruning rule 2), we notice that for every database instance, no matter how relations `nation`, `supplier`, and `partsupp` are grouped in views, the data-size difference between the following two plans is always the same: (1) a plan with `partsupp` and `part` in the same relation subset; and (2) a plan that uses the same partition of the relations, except that `partsupp` and `part` are in different subsets. In particular, consider the following decomposition plans.

Plan	Relation subsets of views in the plan
P_1	$\{nation, supplier\}, \{partsupp\}, \{part\}$
P_2	$\{nation, supplier\}, \{partsupp, part\}$
P_3	$\{nation\}, \{supplier\}, \{partsupp\}, \{part\}$
P_4	$\{nation\}, \{supplier\}, \{partsupp, part\}$
P_5	$\{nation\}, \{supplier, partsupp\}, \{part\}$
P_6	$\{nation\}, \{supplier, partsupp, part\}$
P_7	$\{nation\ supplier, partsupp\}, \{part\}$
P_8	$\{nation\ supplier, partsupp, part\}$

Interestingly, independently of the database instance D,

$$\begin{aligned} & size(P_1, D) - size(P_2, D) = size(P_3, D) - size(P_4, D) \\ = \; & size(P_5, D) - size(P_6, D) = size(P_7, D) - size(P_8, D) \end{aligned}$$

is always true. Therefore, we can *locally* decide whether to consider a view whose relation subset includes both `partsupp` and `part`, by comparing the sizes of the views for {{`partsupp`}, {`part`}} and for {{`partsupp`, `part`}}. If the latter is smaller, then we only need to consider plans that include just one view for these two relations. Otherwise, we only need to consider the other four plans.

By analyzing the query shape closely, we see the following reasons for this effect: (1) the `partsupp` relation has a directed path to each relation in the RI-join graph; and (2) the `part` relation is a "leaf" relation joined with the `partsupp` relation. That is, in the RI-join graph, the relation `part` is connected to only one relation. Now we generalize this relationship as follows.

Lemma 3 *In the relations used in a query Q, suppose there are two relations R_1 and R_2 such that: (1) R_1 has a directed path to each relation in the RI-join graph $G(Q)$; and (2) R_2 is a leaf relation joined with R_1. For a given database instance, consider all pairs of plans P_1 and P_2 that are the same, except that R_1 and R_2 are in the same relation subset in P_1 and in different subsets in P_2. Then $size(P_1, D) - size(P_2, D)$ is a constant, regardless of how other relations are grouped in the plans.*

The main intuition behind this lemma is that the relations on the directed path from R_1 are just "padding" additional values to the tuples in the view $V(R_1)$, without increasing the number of tuples in the result. We call the relation R_1 in the lemma a *core relation* of the query (see [BKSW91] for the related notion of "pivot relation"), and the edge from R_1 to R_2 an *independent edge*. If a query has an independent edge from R_1 to R_2, Lemma 3 allows us to make use of the edge to prune plans in the search space. In particular, when searching for an optimal plan on a database D, we compare $size(V(R_1), D) + size(V(R_2), D)$ and $size(V(R_1, R_2), D)$. Once we determine which of them is smaller, say, $size(V(R_1), D) + size(V(R_2), D)$, we do not need to consider views in which relations R_1 and R_2 are in the same relation subset. Using this pruning rule, by making this local decision in the example we can reduce the number of solutions from 8 to 4.

In summary, the pruning rules can help us reduce the number of solutions and views *without sacrificing the optimality* of the outcome. We will show these advantages experimentally in Section 6.

5.3 Computing View Sizes

In searching for an optimal solution, we need to know the size of a full-reducer view. One way to get the size of a view is to execute the view as a query on the database. This approach can be computationally expensive, especially when there are many views. An alternative is to *estimate* the size of each view. There have been many studies on size-estimation function using database statistics (e.g., [IP95, GM98, Cha98]). However, these approaches are not effective in estimating the size of a full-reducer view with more than just a few join relations.

Our proposed approach to estimating the sizes of views is the following. The size of each view V on a database D can be computed as:

$$size(V, D) = distinct(V, D) \times (\text{size of each tuple in } V)$$

where $distinct(V, D)$ is the number of distinct tuples in the view V on D. (Recall that we are using set semantics in the query results.) The size of each tuple is the sum of sizes of the output attributes of the view. All that remains to be computed is the number of distinct tuples in the view V.

To estimate the sizes of full-reducer views of a query Q, we modify the query as follows. To the output attributes of Q, we add all its join attributes that are not among its output attributes, and denote the new query $\hat{Q}$. We execute this new query to get its results, denoted $D_{\hat{Q}}$. To estimate the number of distinct tuples in a full-reducer view V, we treat all the values, taken together, in a tuple of $\hat{Q}$ as a single value, and the attributes of these values correspond to the output attributes of the view V. This way, we can reduce the view-size estimation problem to that of estimating the number of distinct values in a relation. This problem has received a lot of attention in query optimization. For instance, [HNSS95] proposed two estimators, Smoothed Jackknife Estimator and Shlosser's Estimator. One advantage of our approach is that we only need to run a single query $\hat{Q}$, and its results can be used to estimate the sizes of different views using multiple sampling phases. In the experiments (Section 6) we show that this approach can estimate the size of each view efficiently and accurately. The accuracy depends on the number of tuples we sample from the results of $\hat{Q}$.

Finding views with the same number of distinct tuples: Estimating the number of distinct tuples in a view by sampling could still be expensive. We could further reduce the number of times we need to do sampling, by using this easy observation.

Lemma 4 *Given a database D, suppose that for each subset of connected relations $\{R, S_1, \ldots, S_k\}$ of a query, in the RI-join graph of the query relation R has a directed path to each relation in the subset. Then the view associated with this relation subset has the same number of distinct tuples as that of the view associated with $\{R\}$, that is, $distinct(V(R, S_1, \ldots, S_k), D) = distinct(V(R), D)$.*

It follows from this lemma that we can use the number of distinct tuples in the view $V(R)$ on a database D to obtain the number of distinct tuples in the view associated with $\{R, S_1, \ldots, S_k\}$. This lemma helps us reduce the amount of effort in estimating the number of distinct tuples in views.

For instance, for the query Q_2 above, its RI-join graph has a directed path `partsupp` → `part`. Thus $distinct(V(partsupp), D) = distinct(V(partsupp, part), D)$. Similarly, consider the subset of relations of `partsupp`, `supplier`, and `nation`. The graph has a directed path `partsupp` → `supplier` → `nation`. Thus $distinct(V(partsupp), D) = distinct(V(partsupp, supplier), D) = distinct(V(partsupp, supplier, nation), D)$. After applying pruning rules 1 and 2, we can estimate the number of distinct tuples for just four views: $V(nation)$, $V(supplier)$, $V(partsupp)$, and $V(part)$. For other views, the number of tuples in each view can be found from these four estimates.

5.4 Heuristic-Based Algorithms

When the number of relations in a query is large and the pruning rules cannot effectively reduce the search space of solutions and views, an exhaustive-search algorithm can be computationally prohibitive. We propose two heuristic-based algorithms for searching for a solution. They greedily modify a viewset by "merging" or "splitting" the views, trying to reduce the total size of the viewset. These algorithms can also reduce the number of views whose sizes need to be estimated. This reduction is useful when each view-size estimation is expensive, such as when it is done by sampling.

Bottom-Up Heuristic: This heuristic starts from a set of single-relation full-reducer views, that is, each view is associated with a single relation only. In each step, the algorithm considers combinations of two views whose relation sets are connected in the current solution, and finds one pair of views that, if "merged" into a single view, can reduce the total size the most. The "merged" view is associated with the set of relations that is the union of the relation sets with which the two views are associated. The algorithm repeats the step until it cannot find two views whose merged view has a size smaller than the total size of the two views.

Top-Down Heuristic: This heuristic starts from the single full-reducer view associated with all the relations in the query, that is, the view is the query itself. In each step, for each view in the current solution, the algorithm considers different ways to partition the relations into two connected sets of relations, and considers the corresponding associated views. The algorithm chooses the partition that can reduce the total view size the most. The algorithm repeats this step until it cannot find a "splitting" view which can further reduce the total view size.

In the experiments we evaluate these two heuristics on the degree to which they reduce the number of views whose sizes need to be estimated, and on the quality of their generated solutions.

6 Experiments

We have conducted experiments to evaluate the techniques proposed in Section 5. We focus on the case where we are given a database instance, a single query without self-joins and possibly with arithmetic comparisons, and we want to find a minimum-size viewset that can be used to form an equivalent rewriting of the query using the views. (The results in Section 5 can be extended to queries with comparisons based on Theorem 9 in Section 4.5.)

To choose a set of realistic queries with reasonably large result sizes, we generated queries based on the TPC-H benchmark [TPC], which represents typical queries in a wide range of decision-support applications. The definitions of the queries are given in the Appendix. The first four queries $P_1, \ldots, P_4$ were used to show the pruning effect of the rules. The last three queries $P_5, \ldots, P_7$ had relatively more subgoals and were used to compare the exhaustive-search algorithm with the two heuristic-based algorithms.

In the experiments, we varied the database size from 10MB to 100MB using different scaling factors. We used Microsoft SQL Server Version 8.00.760, running on a PC with a Celeron 2.7GHz CPU, 760MB memory, and a 200GB hard disk. The sampling program was implemented in C++. It interacted with the database using the standard ODBC interface.

6.1 Benefits of Decomposing a Query into Views

We first studied how much size reduction we can achieve by decomposing the results of a query to views. For each query, we ran the exhaustive-search algorithm to find an optimal solution and computed the total size of the resulting views. We also ran the query to get its result size. Table 3 shows the results for the queries on the 100MB database, including the size of each query's results, the size of an optimal decomposition solution, and the size-reduction ratio. For instance, for query P_1, the size of the query results was 8.8MB, while the size of an optimal solution was only 0.78MB, with a reduction ratio about 11. These experiments show that, if a query has a lot of redundancy in its results, by decomposing the query results into view results we could reduce the size significantly.

Table 3 Size reduction for the queries on the 100MB database.

Query	Query result size (MBytes)	Optimal View-Set Size (MBytes)	Size-reduction Ratio
P_1	8.8	0.78	11.28
P_2	4.7	2.08	2.28
P_3	3.2	2.18	1.47
P_4	33.1	9.37	3.53
P_5	9.1	3.97	2.29
P_6	10.0	2.46	4.08
P_7	2.3	1.29	1.81

6.2 Effect of Pruning Rules

We applied the four pruning rules on the queries. Table 4 presents the results of applying the pruning rules in Section 5.2 to queries $P_1, \ldots, P_4$. It shows the number of relations in each query and the number of solutions that need to be considered after applying each rule. It also shows the number of views whose sizes need to be estimated. Notice that if pruning rule 3 is applicable, the number of distinct tuples in one view can be used to estimate the sizes of multiple views. Thus this rule can help us reduce the number of times we need to do sampling.

Table 4 Effect of pruning rules. In each "(a,b)" entry, "a" and "b" are the number of solutions and views that need to be considered, respectively.

Query	Number of Relations	Exhaustive Search	After pruning rule 1	After pruning rule 2	After pruning rule 3
P_1	5	(52,31)	(16,15)	(8,10)	(4,8)
P_2	6	(203,63)	(32,35)	(16,23)	(8,16)
P_3	4	(15,15)	(8,10)	(4,6)	(4,6)
P_4	4	(15,15)	(8,10)	(8,10)	(4,8)
P_5	8	(4140,255)	(128,66)	(64,49)	(64,49)
P_6	7	(877,127)	(64,54)	(64,54)	(32,38)
P_7	6	(203,63)	(32,36)	(16,24)	(16,24)

Take query P_2 as an example. The exhaustive search algorithm needed to consider 203 possible solutions and 63 different views. After applying pruning rule 1, we needed to consider 32 solutions and 35 views. After applying pruning rule 2, we needed to consider 16 solutions and 23 views. Finally, pruning rule 3 reduced the number of solutions that needed to be considered to 8. Notice that after applying all these rules we can still guarantee to find an optimal solution.

6.3 Estimating View Sizes

To estimate the size of each full-reducer view of a query P_i, as described in Section 5.3, we modified the query P_i by adding to its output attributes all its join attributes. We executed the extended query and got its results. We estimated the number of tuples in a full-reducer view by sampling the tuples in the extended-query results, and by then estimating the number of distinct values for the output attributes of this view. We have implemented both the Smoothed Jackknife Estimator and the Shlosser's Estimator developed in [HNSS95]. We report the results of the Smoothed Jackknife estimator since it returned more accurate results for our dataset and queries.

In doing the estimation for each query, to avoid sampling a large number of tuples while keeping enough sampling data, we randomly sampled either $8,000$ tuples or 40% of the total number of tuples in the answer to the extended query, whichever was smaller. We ran the queries on database instances with different sizes. Figure 7 shows how the accuracy changed for some selected views for queries P_5, P_6, and P_7, where accuracy is defined as

$$1 - |\frac{size_{estimate} - size_{real}}{size_{real}}|.$$

The experiments corroborated good accuracy of the estimators. In most cases, accuracy was close to 90%.

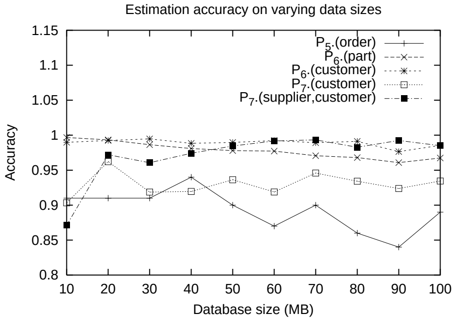

Fig. 7 Accuracy of view-size estimates on different database sizes.

Table 5 shows the sizes of these selected views and corresponding size-estimation times for the 100MB database. For example, the size of the view `V{supplier, customer}` of query P_7 is about 1.5MB, which was estimated in 3.64 seconds with an accuracy of 98.5%.

Table 5 Size and estimation time of views for the 100MB database.

Query	View	View size (KB)	Estimation time (sec)
P_5	V{order}	625	0.34
P_6	V{part}	342	0.25
P_6	V{customer}	16	6.84
P_7	V{customer}	496	0.23
P_7	V{supplier,customer}	1,458	3.64

We also evaluated the effect of Lemma 4 in Section 5.3 to reduce the number of times we need to do sampling to estimate view sizes. It was very effective. For instance, for queries $P_1, \ldots, P_7$, after applying the three pruning rules, the lemma allowed us to reduce the number of sampling times from 8, 16, 6, 8, 49, 38, and 24 to 4, 5, 3, 4, 7, 8, and 6 respectively.

6.4 Exhaustive Search and Heuristics

We have implemented the exhaustive-search algorithm and the two heuristic-based algorithms described in Section 5. Since the pruning rules can effectively reduce the number of views and solutions for queries $P_1, \ldots, P_4$, we used the three algorithms on queries P_5, P_6, and P_7. The size of each view was estimated by sampling, which was computationally expensive. On the other hand, the number of solutions was small for all three algorithms, and the time to search for the best solution in the search space was relatively small compared to the time required to estimate view sizes. Therefore, we focus on reporting the number of views considered by each algorithm; the running time of each algorithm can be determined from this number.

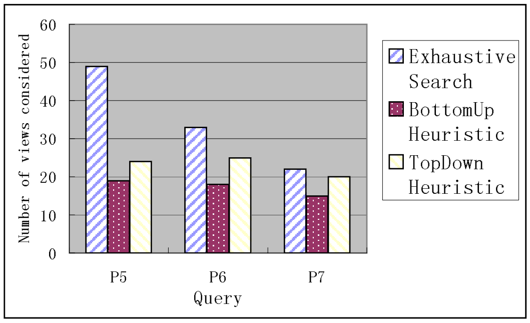

Fig. 8 Views considered by the exhaustive and heuristic algorithms.

Figure 8 shows the results. Take query P_5 as an example. The exhaustive-search algorithm needed to consider 49 views in order to find an optimal solution, while the bottom-up heuristic and the top-down heuristic only needed to consider 19 and 24 views, respectively. For all three queries, the two greedy algorithms found an optimal solution. The results also indicate that the bottom-up heuristic tends to consider fewer views than the top-down heuristic.

7 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 on a given database is minimum. We gave decidability and complexity results for workloads of conjunctive queries; the results differ significantly depending on whether the queries have self-joins. We studied the complexity of the problem for query workloads without self-joins and developed techniques to compute a view decomposition plan efficiently for a single query without self-joins. Our experiments show that the techniques can reduce the size of the relations needed to compute the query answer, and that our proposed pruning rules and heuristic-based algorithms can efficiently find a viewset with a small size.

References

[ACN00] Sanjay Agrawal, Surajit Chaudhuri, and Vivek Narasayya. Automated selection of materialized views and indexes in Microsoft SQL Server. In *Proc. of VLDB*, pages 496–505, Cairo, Egypt, 2000.

[ALU01] Foto Afrati, Chen Li, and Jeffrey D. Ullman. Generating efficient plans using views. In *SIGMOD*, pages 319–330, 2001.

[BDD$^+$98] Randall Bello, Karl Dias, Alan Downing, James Feenan, Jim Finnerty, William Norcott, Harry Sun, Andrew Witkowski, and Mohamed Ziauddin. Materialized views in Oracle. In *Proc. of VLDB*, pages 659–664, 1998.

[BGW$^+$81] Philip A. Bernstein, Nathan Goodman, Eugene Wong, Christopher L. Reeve, and James B. Rothnie Jr. Query processing in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems (TODS)*, 6(4):602–625, 1981.

[BKSW91] Thierry Barsalou, Arthur M. Keller, Niki Siambela, and Gio Wiederhold. Updating relational databases through object-based views. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 248–257, 1991.

[BPT97] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized view selection in a multidimensional database. In *Proc. of VLDB*, 1997.

[CG00] Rada Chirkova and Michael R. Genesereth. Linearly bounded reformulations of conjunctive databases. *DOOD*, 2000.

[CGL00] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views over description logics knowledge bases. In *PODS*, pages 386–391, July - August 2000.

[CGLV00] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In *ICDE*, pages 389–398, 2000.

[Cha98] Surajit Chaudhuri. An overview of query optimization in relational systems. In *PODS*, pages 34–43, 1998.

[Chi02] Rada Chirkova. The view-selection problem has an exponential-time lower bound for conjunctive queries and views. *PODS*, pages 159–168, 2002.

[CHS02] Rada Chirkova, Alon Y. Halevy, and Dan Suciu. A formal perspective on the view selection problem. *The VLDB Journal*, 11(3):216–237, 2002.

[CL03] Rada Chirkova and Chen Li. Materializing views with minimal size to answer queries. *PODS*, 2003.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. *STOC*, pages 77–90, 1977.

[CP84] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems.* McGraw-Hill Book Company, 1984.

[CS00] Zhiyuan Chen and Praveen Seshadri. An algebraic compression framework for query results. In *ICDE*, pages 177–188, 2000.

[GHRU97] Himanshu Gupta, Venky Harinarayan, Anand Rajaraman, and Jeff Ullman. Index selection in olap. In *ICDE*, 1997.

[GL01] Jonathan Goldstein and Per-Ake Larson. Optimizing queries using materialized views: a practical, scalable solution. In *SIGMOD*, pages 331–342, 2001.

[GM98] Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD Conference*, pages 331–342, 1998.

[Gra79] M. Graham. On the universal relation. Technical report, University of Toronto, Canada, 1979.

[GSUW94] Ashish Gupta, Yehoshua Sagiv, Jeffrey D. Ullman, and Jennifer Widom. Constraint checking with partial information. In *PODS*, pages 45–55, 1994.

[Gup97] Himanshu Gupta. Selection of views to materialize in a data warehouse. In *ICDT*, 1997.

[Hal01] Alon Halevy. Answering queries using views: A survey. In *Very Large Database Journal*, 2001.

[HILM02] Hakan Hacıgümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.

[HIM02] Hakan Hacıgümüş, Bala Iyer, and Sharad Mehrotra. Providing database as a service. In *ICDE*, 2002.

[HKWY97] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *Proc. of VLDB*, pages 276–285, 1997.

[HNSS95] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Lynne Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. of VLDB*, pages 311–322, 1995.

[HRU96] Venky Harinarayan, Anand Rajaraman, and Jeff Ullman. Implementing data cubes efficiently. In *SIGMOD*, 1996.

[IP95] Yannis E. Ioannidis and Viswanath Poosala. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD Conference*, pages 233–244, 1995.

[Klu88] Anthony Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, 35(1):146–160, January 1988.

[LBU01] Chen Li, Mayank Bawa, and Jeffrey D. Ullman. Minimizing view sets without losing query-answering power. In *ICDT*, pages 99–113, 2001.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[LMSS95] Alon Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.

[ÖV99] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems.* Prentice-Hall, 1999.

[PL00] Rachel Pottinger and Alon Levy. A scalable algorithm for answering queries using views. In *Proc. of VLDB*, 2000.

[QGMW96] Dallan Quass, Ashish Gupta, Inderpal Singh Mumick, and Jennifer Widom. Making views self-maintainable for data warehousing. In *PDIS*, pages 158–169, 1996.

[SAC$^+$79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34, 1979.

[SY80] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.

[TLS99] Dimitri Theodoratos, Spyros Ligoudistianos, and Timos Sellis. Designing the global data warehouse with spj views. In *CAiSE*, 1999.

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

[TS97] Dimitri Theodoratos and Timos Sellis. Data warehouse configuration. In *Proc. of VLDB*, 1997.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proc. of VLDB*, pages 82–94. IEEE Computer Society Press, 1981.

[YKL97] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *Proc. of VLDB*, 1997.

[ZCL$^+$00] Markos Zaharioudakis, Roberta Cochrane, George Lapis, Hamid Pirahesh, and Monica Urata. Answering complex SQL queries using automatic summary tables. In *SIGMOD*, pages 105–116, 2000.

A Queries Used in the Experiments

```
Query P1:
select  s_acctbal, s_name, s_address, s_phone, s_comment,
        p_partkey, p_mfgr, n_name
from    part, supplier, partsupp, nation, region
where   p_partkey = ps_partkey and s_suppkey = ps_suppkey
        and s_nationkey = n_nationkey and n_regionkey = r_regionkey
        and p_size < 30 and r_name != 'ASIA';

Query P2:
select  p_name, p_brand, o_orderdate,  ps_supplycost,
        l_extendedprice, l_discount, l_quantity
from    part, orders, partsupp, supplier, lineitem, nation
where   s_suppkey = l_suppkey and ps_suppkey = l_suppkey
        and ps_partkey = l_partkey and p_partkey = l_partkey
        and o_orderkey = l_orderkey and s_nationkey = n_nationkey
        and p_retailprice < 1000 and n_nationkey > 4;

Query P3:
select  c_custkey, c_name, c_acctbal, c_address, c_phone, c_comment,
        l_extendedprice, l_discount, l_orderkey, l_linenumber
from    customer, lineitem, nation, orders
where   c_custkey = o_custkey and c_nationkey = n_nationkey
        and l_orderkey = o_orderkey and o_orderdate >= '1995-06-01'
        and o_orderdate < '1996-06-01' and l_returnflag = 'N'
        and n_regionkey != 0;
```

```
Query P4:
select  c_custkey, c_name, c_address, c_phone, c_acctbal, c_comment,
        l_quantity, l_extendedprice, l_orderkey, l_linenumber,
        o_orderpriority, o_clerk, p_name, p_brand, p_type, p_comment
from    customer, orders, lineitem, part
where   c_custkey = o_custkey and o_orderkey = l_orderkey
        and l_partkey = p_partkey and l_quantity < 10
        and l_discount <= 0.08;

Query P5:
select  n_name, c_name, c_phone, p_name, p_brand,
        o_orderdate, ps_supplycost, l_extendedprice, l_discount, l_quantity
from    part, orders, customer, partsupp, supplier, lineitem, nation, region
where   o_orderdate  > '1996-01-01' and r_name = 'Asia'
        and s_suppkey = l_suppkey and ps_suppkey = l_suppkey
        and ps_partkey = l_partkey and p_partkey = l_partkey
        and o_orderkey = l_orderkey and o_custkey = c_custkey
        and s_nationkey = n_nationkey and n_regionkey = r_regionkey;

Query P6:
select  p_name, o_orderdate, o_orderpriority, o_clerk,
        l_extendedprice, l_discount, l_shipmode, n_name, r_comment
from    part, supplier, lineitem, orders, customer, nation, region
where   p_partkey = l_partkey and s_suppkey = l_suppkey
        and s_nationkey != n_nationkey and l_orderkey = o_orderkey
        and o_custkey = c_custkey and c_nationkey = n_nationkey
        and n_regionkey = r_regionkey and r_name = 'ASIA'
        and p_retailprice < 1200;

Query P7:
select  n_name, c_name, c_address, o_orderdate, o_orderstatus,
        l_quantity, l_extendedprice, l_discount
from    region, nation, supplier, customer, orders, lineitem
where   c_custkey = o_custkey and l_orderkey = o_orderkey
        and l_suppkey = s_suppkey and c_nationkey = s_nationkey
        and s_nationkey = n_nationkey and n_regionkey = r_regionkey
        and r_name != 'Middle East' and o_orderdate >=  '1990-01-01'
        and o_orderdate < '1998-06-01';
```