The VLDB Journal (2002) 11: 216–237 / Digital Object Identifier (DOI) 10.1007/s00778-002-0070-0

A formal perspective on the view selection problem

Rada Chirkova[1], **Alon Y. Halevy**[2], **Dan Suciu**[3]

[1] Stanford University, Stanford, CA 94305, USA; e-mail: rada@cs.stanford.edu
[2] University of Washington, Seattle, WA 98195, USA; e-mail: alon@cs.washington.edu
[3] University of Washington, Seattle, WA 98195, USA; e-mail: suciu@cs.washington.edu

Edited by S. Ceri. Received: November 20, 1001 / Accepted: May 30, 2002 /
Published online: September 25, 2002 –

Abstract. The view selection problem is to choose a set of views to materialize over a database schema, such that the cost of evaluating a set of workload queries is minimized and such that the views fit into a prespecified storage constraint. The two main applications of the view selection problem are materializing views in a database to speed up query processing, and selecting views to materialize in a data warehouse to answer decision support queries. In addition, view selection is a core problem for intelligent data placement over a wide-area network for data integration applications and data management for ubiquitous computing. We describe several fundamental results concerning the view selection problem. We consider the problem for views and workloads that consist of equality-selection, project and join queries, and show that the complexity of the problem depends crucially on the quality of the estimates that a query optimizer has on the size of the views it is considering to materialize. When a query optimizer has good estimates of the sizes of the views, we show a somewhat surprising result, namely, that an optimal choice of views may involve a number of views that is exponential in the size of the database schema. On the other hand, when an optimizer uses standard estimation heuristics, we show that the number of necessary views and the expression size of each view are polynomially bounded.

Keywords: Materialized views – View selection

1 Introduction

The problem of view selection has received significant attention in recent literature [1, 2, 5, 9–11, 14, 15, 20, 21]. Broadly speaking, the problem is the following: given a database schema $\mathcal{R}$, storage space B, and a workload of queries $\mathcal{Q}$, choose a set of views $\mathcal{V}$ over $\mathcal{R}$ to materialize, whose combined size is at most B. The set of views $\mathcal{V}$ is called a *view configuration*. The views can either be materialized within the database, in which case they are available to supplement normal query processing, or they can be materialized in a separate data warehouse, and in that case the workload queries should be answered *only* from the views. The goal of the view selection process is to find a set of views that minimizes the expected cost of evaluating the queries in $\mathcal{Q}$ in either of the above contexts. In addition, the view selection problem may involve choosing a set of indexes on the views, and may consider the cost of updates to the views $\mathcal{V}$.

The original motivation for the view selection problem comes from data warehouse design, where we need to decide which views to store in the warehouse to obtain optimal performance [2, 14, 20]. Another motivation is provided by recent versions of several commercial database systems which support incremental updates of materialized views and now use materialized views to speed up query evaluation. Therefore, choosing an appropriate set of views to materialize in the database is crucial in order to obtain performance benefits from these new features [1].

The view selection problem and its generalizations will play an even greater role in contexts where data needs to be placed intelligently over a wide area network. In these contexts, users are spread over a network, and each location may have different types of query characteristics and/or performance requirements. A very simple version of this problem was considered in the context of data placement in distributed databases (see [16] for a survey). For example, consider the context of peer-based data management [8], where data is both integrated and accessed from many peers on a wide-area network. Each of these peers has a local store but can also retrieve data at different rates from various points on the network. A key factor in ensuring good performance in such a context is intelligent placement and replication of data at different nodes on the network, which is akin to selecting a set of views at each node. Note that the view selection problem can be viewed as a special case of this intelligent data placement problem, in which there are only two nodes in the network (the database and the warehouse).

This paper considers the fundamental properties of the view selection problem for workload and view configurations involving conjunctive queries (i.e., queries allowing join, projection, and equality selection). Several algorithms have been proposed in the past for solving the view selection problem for such queries (e.g., [1, 20]), but they all made certain critical tacit assumptions. The first assumption is that the only views

that need to be considered for an optimal view configuration are those that are subexpressions of queries in the workload (i.e., that contain a subset of the tables with a subset of the join predicates in the query). The second assumption is that there is some low upper bound on the number of views in an optimal view configuration.

For most real applications, these assumptions are correct, but it is possible in theory to find examples where they break. The following example, adapted from [5], shows a query workload for which the first assumption does not hold. While this is essentially an abstract example, we have given it a real-life appearance to make it easier to read. (We will show later in the paper that the second assumption doesn't always hold either.)

Example 1 This example exhibits a workload of conjunctive queries and a storage limit, such that it is impossible to materialize the answers to all workload queries. We consider for materialization conjunctive views, such that each view alone satisfies the storage limit and can support all workload queries. This example shows that for the given workload, an infinite number of such views are *not* subexpressions of any workload query. Moreover, the example shows that the search space of such views is infinite. While this is essentially an abstract example, we have given it a real-life appearance to make it easier to read.

Here is how this example might come about. Consider a hypothetical shipping company that serves a number of cities, with fixed delivery schedules between pairs of cities. Suppose the company has a centralized database, with a base table T ($source$, d, $dest$) that stores all pairs of cities $source$ and $dest$, such that there is a scheduled delivery from *source* to *dest* on day d of the week (a number between 1 and 7). See Fig. 1 for a graph that illustrates an instance of T with 89 cities; the graph represents deliveries scheduled between the cities. In the graph, delivery day d is omitted to reduce clutter.

Suppose that agents of the company try to contract shipments to independent truck drivers, by attracting them with tours connecting two or more cities. The company predefines a number of tour types to offer to the truck drivers, and agents need to query the database and find out whether the tour requested by the driver exists starting at a given city. Every tour type starts and ends in the *same* city. The simplest tour is the "two-city roundtrip", for which we give the definitions in both SQL and datalog. The query returns all cities X_1, such that there exist two scheduled deliveries: one, from X_1 to some other city X_2 on Monday (day #1), and two, back from X_2 to X_1 on Tuesday (day #2).

```
select distinct T1.source
from T AS T1, T AS T2
where T1.source = T2.dest
    and T1.dest = T2.source
    and T1.day = 1 and T2.day = 2;
```

$$Q_1(X_1) :- T(X_1, 1, X_2), T(X_2, 2, X_1).$$

We are only concerned with set semantics in this paper, hence the `distinct` in the SQL statement. The equivalent datalog formulation should be self-explanatory.

A more complex tour is the "four cities in five days with a break" tour below:

$$Q_2(X_1) :- T(X_1, 1, X_2), T(X_2, 1, X_3), T(X_3, 2, X_4), T(X_4, 4, X_3), T(X_3, 5, X_1)$$

Here the break is on Wednesday (day #3) and the return trip goes through X_3 a second time.

Suppose the company predefines a large collection of such tour types, each involving up to a dozen or so cities. For each tour there will be an associated query, and the set of all such queries defines our workload: $\mathcal{Q} = Q_1, Q_2, \ldots$. Notice that the queries only inform the agents whether the desired tour is available from a given starting city, and do not return the actual tour.

As tours get longer, the corresponding queries get more complex. We consider precomputing (materializing) the answers to some queries to speed up this workload. For example, if the number of queries in the workload is small, then we may simply precompute all of them and store their answers. If the number of queries increases, however, then their combined answers may be too large, even though the answer to each individual query is relatively small. We assume that the storage limit in the problem prevents us from materializing any table that is larger than the set of all city names mentioned in the table T. (Notice that the answer to any single query in our workload is just a set of city names.) Our goal is to find a single view to materialize, such that the view satisfies the storage limit and can support all workload queries.

Obviously the choice of the best view depends on the particular statistics available on the database. From Fig. 1 it can be seen that the graph of connections among the cities is sparse, and only a small subset of cities are on some cycle. In the figure, the only twelve cities belonging to any cycle at all are represented as ovals. Then, one idea is to precompute a view with the (small) set of cities that belong to some cycle. For example, the view

$$C_5(X_1) :- T(X_1, D_1, X_2), T(X_2, D_2, X_3), T(X_3, D_3, X_4), T(X_4, D_4, X_5), T(X_5, D_5, X_1)$$

computes the set of cities on a cycle of length 5, but ignoring the days of the delivery[1]. The view C_5 can be used to speed up query Q_2, for example by rewriting Q_2 as:

$$Q_2'(X_1) :- C_5(X_1), T(X_1, 1, X_2), T(X_2, 1, X_3), C_5(X_3), T(X_3, 2, X_4), T(X_4, 4, X_3), T(X_3, 5, X_1)$$

Query Q_2' is indeed equivalent to Q_2, because in Q_2 both X_1 and X_3 must be on a cycle of length five. Indeed, the body of Q_2 contains the following circuit of length five: $X_1, X_2, X_3, X_4, X_3, X_1$. Moreover, Q_2' can be evaluated more efficiently than Q_2: for example, the plan

$$((C_5 \bowtie T) \bowtie T) \bowtie (((C_5 \bowtie T) \bowtie T) \bowtie T) \text{ for } Q_2'$$

is more efficient than the plan

$$(T \bowtie T) \bowtie ((T \bowtie T) \bowtie T)$$

for Q_2, because the answer to $C_5 \bowtie T$ is smaller than T. However, C_5 cannot be used to speed up Q_1 in a similar fashion, since there is no relationship between cycles of length 2 and those of length 5. If we could compute a view containing all

[1] Our randomly generated graph happens not to have any cycle of length 5.

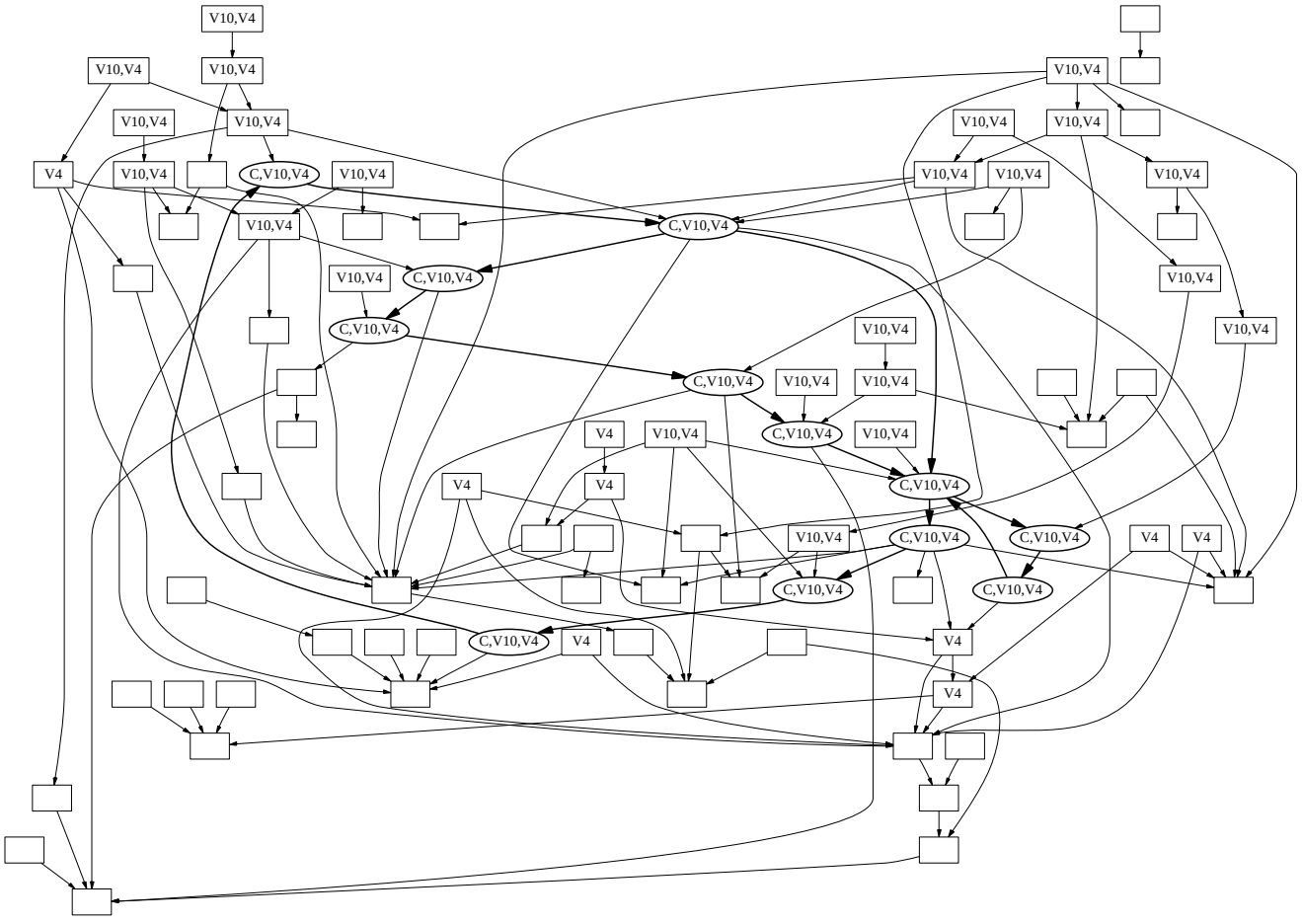

Fig. 1. A database instance for Example 1. The nodes (both rectangles and ovals) represent cities, the edges represent scheduled deliveries (the delivery day is omitted). Labels `C, V10, V4` attached to the nodes represent the views C, V_{10}, V_4 defined in the example, with $C \subset V_{10} \subset V_4$. The view C contains all cities on some circuit; the cities are represented as ovals

possible cycles, say $C = C_2 \cup C_3 \cup C_4 \ldots$, where C_n denotes the set of nodes on a cycle of length n, then that view would satisfy the storage limit and could, at the same time, be used to speed up all queries in the workload. For example, in Fig. 1 there are 12 nodes in C, and they are marked with the symbol `C`. The "view" C, however, is not conjunctive, and we restrict our consideration to conjunctive views.

Another idea is to compute a single view with all cities from which we can follow a long chain of cities, say of length 10, that does not (necessarily) end in the starting city:

$$V_{10}(X_1) :- T(X_1, D_1, X_2), T(X_2, D_2, X_3), \\ \ldots, T(X_9, D_9, X_{10})$$

Notice that V_{10} contains C, $C \subseteq V_{10}$, since a chain simply "wraps around" a cycle, but the converse is false. In Fig. 1 there are 35 more nodes in V_{10} and are marked with the symbol `V10`.

The view V_{10} satisfies the storage limit. At the same time, all queries in the workload can be sped up by using V_{10}; this can be done in several ways, we only illustrate here one possible rewriting for both Q_1 and Q_2:

$$Q_1(X_1) :- V_{10}(X_1), T(X_1,1,X_2), V_{10}(X_2), T(X_2,2,X_1).$$
$$Q_2(X_1) :- V_{10}(X_1), T(X_1,1,X_2), T(X_2,1,X_3), V_{10}(X_3), \\ T(X_3,2,X_4), T(X_4,4,X_3), T(X_3,5,X_1).$$

There is nothing special about the number 10 in V_{10}. We could have used any view V_n defining a chain of length n. The views V_n are smaller when n is large, and vice versa. There are 44 nodes in the view V_4 in Fig. 1, marked with `V4`, and indeed this set includes the 35 nodes in V_{10} . Clearly, we need to choose large values for n, since a smaller view will eliminate more false positives, and thus speed up the workload even more.

Our focus in this paper is on the choice of the view(s). First, the example illustrates that it does not suffice to consider only views that are defined as subsets of subgoals of the queries in the workload. In fact, V_{10} is not a subexpression of any query in the workload. Second, the example illustrates that it is not clear where to stop: ever larger values of n seem to produce better and better views V_n.

As we show later in the paper, it is also unclear how *many* views we need to select. Hence, we are faced with several fundamental questions regarding the view selection problem: (1) which set of views do we need to consider in an optimal view configuration? (2) what is the maximal size of an optimal view configuration? and (3) what is the complexity of the view selection problem?

We show that a key factor affecting the answer to the aforementioned problems is which statistics we may expect to have on the database relations. These statistics are crucial in order to estimate the size of the views we consider to materialize and the cost of evaluating queries over the views. We therefore distinguish two versions of the problem: the Partial Statistics Assumption (PSA) and the Complete Statistics Assumption (CSA). Under PSA, we assume that standard statistics are maintained on the database, and that cost and size estimates are obtained by some estimation function. In contrast, under CSA we assume that we have an oracle that gives us the precise size of any view over the database schema. Note that in practice such an oracle can be based on statistics collected from running queries over the database for some period of time.

After formally defining the view selection problem in Sect. 2, we begin by considering the problem under CSA in Sect. 3. We first show that the workload queries provide an exponential upper bound on the size of the view definitions we need to consider in an optimal configuration, and as a result, the view selection problem is decidable in triple exponential time. However, then we show a rather surprising result: in general, an optimal view configuration may include a number of views that is exponential in the size of the query and database schema. As a result, the view selection problem has an exponential-time lower bound. In fact, this result holds even if we further restrict the expressive power of our query language, and under different cost models.

Next, we consider the view selection problem under PSA in Sect. 4. We show here that an optimal solution to the view selection problem always includes a number of views that is bounded by a polynomial in the size of the database schema, the workload of queries, and the binary representations of the relation sizes and the available space bound. We also prove that the size of the view definitions in an optimal configuration is bounded linearly in the size of the query definitions in the workload. The two upper bounds place the view selection problem in the complexity class NP. The results for the PSA assume certain properties of the size estimation function that are general enough to capture most size estimators commonly used in practice. While unreasonable choices of size estimation functions break our results, they also make the view selection problem rather uninteresting.

Our results also shed light on the problem of answering queries using views [12,13]. It is known that given a conjunctive query Q with n subgoals with no comparison predicates and a set of views $\mathcal{V}$, there exists an equivalent rewriting of Q using $\mathcal{V}$ only if there exists a rewriting with n subgoals or less [18]. However, it has been an open problem whether an *optimal* rewriting also satisfies the same bound on its size. Our results on the size of an optimal view configuration also establish bounds on the size of an optimal rewriting of a query using a set of views.

1.1 Related work

There has been relatively little theoretical analysis of the view selection problem in the literature to date. In [5], Chirkova and Genesereth considered the space requirements for the view selection problem for the context of data warehouse design. They consider certain restrictions under which they show that one can limit the search of an optimal configuration to views that are subexpressions of the queries in the workload. Gupta [9] considers the view selection problem, but he does not model the attributes of the relations being joined. That is, he considers every relation to be a proposition and looks at query plans that are AND-OR graphs over these propositions. As a consequence, his model does not capture selections, projections, or different join predicates. Our work shows that the complexity of the problem crucially depends on modeling these operations. In a later paper [11], the authors also consider the cost of view maintenance, but under the same model of inputs. Finally, in both papers, the set of relations in the warehouse is given as part of the input, whereas in our work we're only given the database schema and the workload queries.

In [20], Theodoratos and Sellis describe an algorithm for searching a space of candidate warehouses. They do not discuss the complexity of the view selection problem, and their search space does not include warehouses that contain views that are projections on the database relations. As we show in Sect. 3, considering such views has a significant impact on the complexity of the view selection problem.

Agrawal et al. [1] describe a system for view selection that is incorporated into the Microsoft SQL Server. They present several very effective heuristics for pruning the space of possible view configurations. An important aspect of their work is that they consider the problem of selecting views and indexes simultaneously. Because of that, they do not consider projection views, since those can often (but not always!) be simulated by indexes on other views.

Harinarayan et al. [14] show that the problem of view selection for data cubes is NP-hard, and describe greedy algorithms for approximating an optimal set of views. View-selection for data cubes is further elaborated in [15]. In [10] the work of [14] is extended to include index selection. Other works that considered algorithms for view selection are [2,17,21,23].

This paper is mainly based on an earlier conference paper [6]. The main result in Sect. 3.4 was described in an extended abstract in [4].

2 Problem definition

We consider the view selection problem for the case in which both queries and views may contain joins, projections and equality selections (i.e., conjunctive queries). Furthermore, we consider queries and views under set semantics, rather than bag semantics. Clearly, it is desirable to extend our analysis to more complex queries and to bag semantics. However, an in depth consideration of all the factors involved in this problem requires that we start from a limited language, and this language already yields several interesting results.

Throughout the paper we use the datalog notation for conjunctive queries. Note that in this notation, joins are expressed as multiple occurrences of the same variable. In general, we say that a variable is a *join variable* if it appears in more than one subgoal in the body of the query. Given a database instance D, the *size* of a view V over D is the number of tuples in the answer to V.

Workloads. The appropriate choice of views in a particular context is highly dependent on the set of queries we expect to be given. We model our expected queries by a query workload, which is a set of queries $\mathcal{Q} = Q_1, \ldots, Q_m$, where each query Q_i has an associated non-negative weight, w_i. The weight describes the relative frequency of Q_i within the workload. We require that the weights sum up to 1 ($\sum_{1 \le i \le m} w_i = 1$).

View configurations. Given a database schema $\mathcal{R}$ and a workload $\mathcal{Q}$, our goal is to choose a set of views $\mathcal{V}$ to materialize. We refer to a choice of views as a *view configuration* (or *configuration* for short). There are several contexts in which we may be choosing configurations:

1. Performance of query processing: we may choose to materialize a set of views over a database, such that subsequent queries can make use of these views in query processing. Many commercial database systems today support the functionality of answering queries using views [3,7,22].
2. Warehouse design: the goal is to select a set of views to materialize in a data warehouse, on which we expect to process OLAP-style queries. In this case, the query processor of the warehouse must use *only* the selected views in order to answer the queries.
3. Data placement in a distributed setting: we may want to locally cache views on data that is stored in remote locations. When processing queries, the views can be used to reduce the amount of communication between the nodes [8].

We use the following terminology when referring to view configurations. The *size* of the configuration for a given database instance D is the sum of the sizes of the views evaluated over D. Given a configuration $\mathcal{V}$, we evaluate the cost of answering a workload $\mathcal{Q}$ on $\mathcal{V}$ using a cost model C. Specifically, we assume that the specification of a schema $\mathcal{R}$ includes a set of statistics $\Sigma_{\mathcal{R}}$. The function $C(\mathcal{R}, \mathcal{V}, Q_i)$ estimates the cost of evaluating the query Q_i given the schema (and, in particular, the database statistics) and the views in $\mathcal{V}$. We elaborate on these statistics below.

In practice, before we actually materialize the chosen views, we may only be able to *approximate* their sizes. As we see later, the quality of the size estimator function plays a crucial role in the complexity of the view selection problem.

We assume that the function $E(\mathcal{R}, V)$ returns the estimated size of a view V over a database with schema $\mathcal{R}$ and its associated statistics $\Sigma_{\mathcal{R}}$. Note that the function C uses the estimates produced by E, hence the accuracy of the cost depends on the accuracy of both C and E. Given the function C and a configuration $\mathcal{V}$, the cost of the configuration, denoted by $C(\mathcal{R}, \mathcal{V}, \mathcal{Q})$, is the sum $\sum_{Q_i \in \mathcal{Q}} C(\mathcal{R}, \mathcal{V}, Q_i) \times w_i$.

Finally, our goal is to select a view configuration that satisfies a given space constraint. We denote by B an amount of memory allotted for the views, and we assume that B is given via its binary representation. We are now ready to formally define the view selection problem.

Definition 1 *View selection problem Let $\mathcal{R}$ be a database schema, B be the available storage space, $\mathcal{Q}$ be a workload on a database described by the schema $\mathcal{R}$, C be a cost estimation function for query processing, and E be a function for estimating the sizes of queries over $\mathcal{R}$. The view selection problem is to find a set of views $\mathcal{V}$ over $\mathcal{R}$ whose total size is at most B and that minimizes $C(\mathcal{R}, \mathcal{V}, \mathcal{Q})$.* □

A few points are worth noting about the definition before we proceed:

1. The input to the problem includes the database schema and the statistics associated with it. Hence, a solution applies to the *set* of databases obeying these statistics, and not only to a single database instance.
2. The input to the view selection problem does *not* include a database instance. Hence, the complexity results we present later are in terms of the size of the schema and of the workload, not the size of the database.
3. We present several complexity results concerning the view selection problem. While view selection is an optimization problem, the corresponding decision problem that we refer to in the results is: *Is there a view configuration whose cost is less than K?*, for a given number K.
4. It is important to note that materializing all the queries in $\mathcal{Q}$ is not always a possible solution because of the space limitation (even if we ignore the cost of materializing and updating the views). It is, of course, reasonable to assume that B is at least big enough to hold the result of any single query in $\mathcal{Q}$.

2.1 Cost and size estimates

A key difficulty in treating the view selection problem is that we tread a very fine line with the choice of a cost model. In previous work, the view selection problem left the cost model as a parameter (e.g., [20]), or, when actually implemented, relied on the cost estimates of the optimizer (e.g., [1]). Ideally, we would like to leave the cost model as a parameter to the problem, and obtain results that hold for *any* cost model. On the other hand, we show some interesting results when considering certain classes of cost models. This tradeoff affects the following discussion about the inputs to the view selection problem.

Cost model. For the purpose of our discussion, the critical aspect of a cost model is the estimated cost of a join. In our discussion we consider the following cost model for joins. Consider a join of two relations, R and S, of size L and M, respectively. Let N be the size of the relation $R \bowtie S$. Then the cost of the join of R and S is $\alpha \times LM + \beta \times (L+M) + \gamma \times N$. We consider two versions of this cost model. In the first version, which we call the *product cost model*, we assume $\alpha \neq 0$. We call the remaining version ($\alpha = 0$) the *sum cost model*.

This cost model faithfully describes most real-world situations:

- For nested-loop joins, the first term is the dominating factor in the cost, as α is a fraction that depends on the number of main-memory pages available.
- For hash or sort-merge joins, the I/O costs are assumed to be proportional to the sum of the sizes of the joined relations, but the main-memory costs are still proportional to the size of the product of the joined relations. Hence, here we assume that α is relatively small.

Finally, we assume that the cost of a selection is that of a scan on the relation, and the cost of a projection on a relation of size N is $N \log(N)$ (though in practice, it is rarely more than $3N$).

When a view selection algorithm examines a particular candidate configuration $\mathcal{V}$, it needs to compute $C(\mathcal{R}, \mathcal{V}, \mathcal{Q})$. In our discussion, we assume the cost can be computed in time polynomial in the size of $\mathcal{Q}$ and the configuration $\mathcal{V}$. In general, query optimization can be exponential in the size of the query, and hence, calculating $C(\mathcal{R}, \mathcal{V}, \mathcal{Q})$ could potentially be exponential in the size of $\mathcal{Q}$. The reason for our choice is twofold: (1) in practice, query optimizers rarely perform an exhaustive search of query plans; and (2) we want to isolate the effect of the view selection problem from the problem of evaluating a configuration.

In estimating costs of configurations, we also need to specify what kinds of query execution plans the optimizer will be considering. We assume standard query plans, where selections are pushed down to the leaves and the projection is the last operation; query plans differ only in the join order. In our discussion we will distinguish between cases in which the optimizer considers only left-linear query plans and cases where it considers all bushy query plans.

Size estimation. A key difficulty in evaluating the cost of a candidate view configuration is that we also need to evaluate the sizes of the views in the configuration, which are, in turn, used as inputs to the cost estimation function C. Hence, a key aspect in the analysis of the view selection problem is specifying which statistics about the database are available as input along with the schema.

The availability of statistics significantly affects the analysis of the view selection problem. Hence, we study two versions of the view selection problem, and we show that they lead to drastically different results. In the first version, discussed in Sect. 3, we make the *complete statistics assumption* (CSA), in which we assume that we have an oracle that tells us the size of any view over the database. In practice, it may often be possible to obtain such an oracle. In some cases, we can approximate such an oracle by using the results of many runs on the system during which a large body of statistics have been gathered. In other cases, we may be able to evaluate the results of the views and compute their sizes.

In the second version of the problem, we make the *partial statistics assumption* (PSA), where we rely on the size and cost estimators of the query processor, and assume their functions are parameters to the problem. In Sect. 4 we consider the PSA case under some reasonable assumptions on size estimation functions that are considered in practice.

An important distinction between CSA and PSA is the *monotonicity property* of the size estimator. We say that the size estimation function E is monotone if, whenever a view V_1 is contained in V_2, $V_1 \subseteq V_2$, then $E(\mathcal{R}, V_1) \leq E(\mathcal{R}, V_2)$ for every given statistics $\Sigma_{\mathcal{R}}$. Under CSA size estimators are always monotone, while under PSA they usually are not. This has important consequences on the view selection problem.

Additional issues. Two important issues that arise in the context of the view selection problem are selection of indexes on the view in the configuration and considering the cost of updating the views. The bulk of our discussion does not consider these two issues, but Sect. 5 discusses how our results could be affected by their consideration.

3 Complete statistics assumption

In this section we consider the view selection problem under the *complete statistics assumption* (CSA). That is, we assume that we have an oracle that can accurately estimate the size of any view over the database. We begin in Sect. 3.1 by showing that view selection is decidable. The main result of the section (Sect. 3.2) is that there exist a database schema and a query workload for which the number of views in an optimal configuration is exponential in the size of the schema. As a consequence, we obtain an exponential-time lower bound on the complexity of the view selection problem for CSA. Section 3.3 explores some of the boundaries of the lower-bound result.

3.1 Decidability of view selection

The first question that we ask about the view selection problem is whether it is even decidable. A priori, it is not even clear which views need to be considered for optimal configurations, and whether there is even a finite number of such views. In fact, in Sect. 4 we show that in certain cases the number of relevant views could be infinite.

The following theorem, which is an extension of a result in [5], shows that view selection is decidable for CSA. The theorem holds for any cost estimation function E that is monotone, in particular for CSA.

Theorem 1 *Suppose the cost estimation function, E, is monotone. Consider a view selection problem given by $\mathcal{R}$, $\mathcal{Q}$, B, and assume the problem admits at least one configuration[2] (not necessarily optimal). Then, there exists an optimal view configuration $\mathcal{V}_0$ such that each view in $\mathcal{V}_0$ has a number of subgoals that is at most exponential in the size of $\mathcal{Q}$, and the number of views in $\mathcal{V}_0$ is at most double exponential in the size of $\mathcal{Q}$. Moreover, an optimal view configuration can be found in triple exponential time.* □

Proof. We will construct a set of views $\mathcal{W}$ with the following properties: (a) each view $\bar{V} \in \mathcal{W}$ has a number of subgoals that is at most exponential in the size of $\mathcal{Q}$; (b) the number of views in $\mathcal{W}$ is at most double exponential in the size of $\mathcal{Q}$; and (c) for any view configuration $\mathcal{V}$ for the given problem, and any view $V \in \mathcal{V}$, there exists a view $\bar{V} \in \mathcal{W}$ such that $\bar{V}$ is contained in V ($\bar{V} \subseteq V$) and, denoting $\mathcal{V}'$ the configuration obtained from $\mathcal{V}$ by replacing V with $\bar{V}$ ($\mathcal{V}' = \mathcal{V} - \{V\} \cup \{\bar{V}\}$), we have $C(\mathcal{R}, \mathcal{V}', \mathcal{Q}) \leq C(\mathcal{R}, \mathcal{V}, \mathcal{Q})$. Once we have constructed $\mathcal{W}$ with these properties, the theorem is proven as follows: first, there exists at least some view configuration for our problem that consists only of views in $\mathcal{W}$: indeed, starting from any view configuration $\mathcal{V}$ (and we know there exists at least one), we replace one by one each view $V \in \mathcal{V}$ with a view $\bar{V} \in \mathcal{W}$, and notice that in doing so the total size of the view configuration will not increase, because we have $\bar{V} \subseteq V$. Next, denoting $\mathcal{V}_0$ the cheapest view configuration for our problem consisting only of views in $\mathcal{W}$, we prove that this is an optimal view configuration. Indeed, let $\mathcal{V}$ be any view configuration. We replace one by one each view $V \in \mathcal{V}$ with $\bar{V} \in \mathcal{W}$: eventually we end up with a configuration $\mathcal{V}'$ consisting only

[2] When B is too small, there may be no solution at all.

of views in $\mathcal{W}$, such that $C(\mathcal{R}, \mathcal{V}', \mathcal{Q}) \leq C(\mathcal{R}, \mathcal{V}, \mathcal{Q})$, and also $C(\mathcal{R}, \mathcal{V}_0, \mathcal{Q}) \leq C(\mathcal{R}, \mathcal{V}', \mathcal{Q})$ by definition of $\mathcal{V}_0$: this proves that $\mathcal{V}_0$ is optimal. Finally, we can search for $\mathcal{V}_0$ in triple exponential time, by just enumerating all subsets $\mathcal{V}_0 \subseteq \mathcal{W}$.

In the remainder of the proof we show how to construct $\mathcal{W}$ and prove its properties (a), (b), (c). Let $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$. Consider all variables occurring in all queries in the workload: $Var(Q_1), \ldots, Var(Q_m)$. Let n_i be the number of subgoals in Q_i, for $i = 1, \ldots, m$. Define a new set of variables, $\overline{Var}$, to be:

$$\overline{Var} = \bigcup_{k_1 \leq n_1, \ldots, k_m \leq n_m} (Var(Q_1))^{k_1} \times \ldots \times (Var(Q_m))^{k_m} \quad (1)$$

A variable $\bar{x}$ in $\overline{Var}$ is $\bar{x} = \langle x_1, \ldots, x_p \rangle$, where $x_1, \ldots, x_p \in Var(Q_1) \cup \ldots \cup Var(Q_m)$, and there are at most n_i variables from $Var(Q_i)$, for every $i = 1, \ldots, m$. This should not be confused with a p-tuple, $(x_1, \ldots, x_p)$: rather, $\bar{x}$ is a new variable, and we want a one-to-one correspondence between p-tuples and such new variables. Notice that the number of variables in $\overline{Var}$ is exponential in the size of $\mathcal{Q}$. Next, consider all subgoals one can construct with variables in $\overline{Var}$. We assume for the moment that we have a fixed schema: then there are still only exponentially many subgoals, because for each predicate name of arity k one can construct $(\mid \overline{Var} \mid)^k$ subgoals; we will discuss below how we can drop this assumption. Finally, define $\mathcal{W}$ to consists of all possible views whose bodies consists of such subgoals: to obtain $\mathcal{W}$ we need to enumerate all subsets of such subgoals, and for each of them enumerate all subsets of the variables to designate head variables. This leads to another exponent. Hence $\mathcal{W}$ has properties (a) and (b).

Now we prove that $\mathcal{W}$ has property (c). Let $\mathcal{V}$ be some view configuration, and let $V \in \mathcal{V}$. We will construct $\bar{V}$ satisfying (c). Recall that $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$. For each $i = 1, \ldots, m$, consider some rewriting Q_i' of Q_i that uses V. To be more precise, the body of Q_i' has as one of its subgoals $\theta_i(V)$, where θ_i is a substitution of V's head variables, $\theta_i : Var(Head(V)) \rightarrow Var(Q_i')$:

$$Q_i' :- g_1, \ldots, g_k, \theta_i(V) \quad (2)$$

We assume for the moment that V is used exactly once in the rewriting Q_i', and denote $g_1, \ldots, g_k$ the other subgoals in the rewriting; we discuss the general case below. Here $\theta_i(V)$ is one predicate, consisting of V's head with its head variables substituted according to θ_i.

Denote $e(Q_i')$ the expanded body of Q_i', where all views have been replaced with their bodies. It is easy to extend θ_i to a substitution $\theta_i : Var(V) \rightarrow Var(Q_i')$ which is a homomorphism $\theta_i : V \rightarrow e(Q_i')$: simply define it to be the identity mapping on all variables in $Var(V) - Var(Head(V))$. Here, and in the sequel, we relax the conditions on homomorphisms from V by dropping the requirement that it maps head variables to head variables: thus, $\theta_i : V \rightarrow e(Q_i')$ maps V's body to that of $e(Q_i')$, and we don't care if it relates V's head variables to those in $e(Q_i')$. Hence, the expanded query Q_i' has the form:

$$e(Q_i') :- e(g_1), \ldots, e(g_k), \theta_i(Body(V)) \quad (3)$$

where $e(g_j)$ denotes the expansion of the subgoal g_j (its body, with head variables appropriately substituted, and all existential variables renamed).

Since Q_i and Q_i' are equivalent, there exist homomorphisms $\varphi_i : e(Q_i') \rightarrow Q_i$ and $\psi_i : Q_i \rightarrow e(Q_i')$. Consider now the homomorphism[3] $\rho_i : V \rightarrow Q_i$, $\rho_i = \varphi_i \circ \theta_i$: this is a function $\rho_i : Var(V) \rightarrow Var(Q_i)$. Recall that we are doing this for every $i = 1, \ldots, m$. Define the substitution $\bar{\rho} : Var(V) \rightarrow \overline{Var}$ to be $\bar{\rho}(z) = \langle \rho_1(z), \ldots, \rho_m(z) \rangle$, for every variable z in V. Now we can define the new view, $\bar{V}$, to be $\bar{V} = \bar{\rho}(V)$ (more precisely: $Body(\bar{V}) = \bar{\rho}(Body(V))$, $Head(\bar{V}) = \bar{\rho}(Head(V))$). Obviously $\bar{V} \in \mathcal{W}$. Since $\bar{\rho} : V \rightarrow \bar{V}$ is a homomorphism, it follows that $\bar{V} \subseteq V$.

It remains to prove $C(\mathcal{R}, \mathcal{V}', \mathcal{Q}) \leq C(\mathcal{R}, \mathcal{V}, \mathcal{Q})$, where $\mathcal{V}' = \mathcal{V} - \{V\} \cup \{\bar{V}\}$. For that we show that every rewriting Q_i' that uses V can be reformulated into a rewriting Q_i'' that uses $\bar{V}$, and with no greater cost. We use the monotonicity of the cost estimation function, E, and the fact that $\bar{V} \subseteq V$, to argue that the cost of the plan with $\bar{V}$ is not greater than that of V. In Eq. 2 replace $\theta_i(V)$ with $\pi_i(\bar{V})$, where $\pi_i : \overline{Var} \rightarrow Var_i$ is the substitution defined by $\pi_i(\langle x_1, \ldots, x_m \rangle) = x_i$, and replace every subgoal g_j with $\varphi_i(g_j)$:

$$Q_i'' :- \varphi_i(g_1), \ldots, \varphi_i(g_k), \pi_i(\bar{V}) \quad (4)$$

Here $\varphi_i(g_j)$ denotes the head of the view g_j to which we apply the substitution φ_i. Since $\pi_i(Body(\bar{V})) = \varphi_i(\theta_i(Body(V)))$, the expansion of Q_i'' is the following:

$$e(Q_i'') :- \varphi_i(e(g_1)), \ldots, \varphi_i(e(g_k)), \varphi_i(\theta_i(Body(V))) \quad (5)$$

It follows that $\varphi_i : e(Q_i') \rightarrow e(Q_i'')$ is a homomorphism: in fact, $Body(e(Q_i''))$ (Eq. 5) is precisely the image under φ_i of of $Body(e(Q_i'))$ (Eq. 3). It is easy to check now that Q_i and Q_i'' are equivalent: the identity mapping is a homomorphism from $e(Q_i'')$ to Q_i, while $\varphi_i \circ \psi_i$ is a homomorphism from Q_i to $e(Q_i'')$.

Finally, we discuss how to relax the two assumptions we made during the proof. One was that V be used exactly once in the rewriting of Q_i, $i = 1, \ldots, m$. This resulted in $\bar{V}$ using only a subset of the variables in $\overline{Var}$, namely those for which $k_1 = \ldots = k_m = 1$ in Eq. 1. The case when V occurs multiple times in some rewritings of some queries is handled similarly, by treating each occurrence differently, as we treated occurrences from different queries differently. In that case we need variables for which the $k_i \neq 1$, but we never need values of k_i that are larger than n_i, the number of subgoals of Q_i, because each rewriting using more than n_i occurrences of V can be replaced with a rewriting using at most n_i occurrences.

The second assumption was that the schema is fixed, to allow us to argue that $\mathcal{W}$ uses only exponentially many distinct subgoals. We can drop this assumption. Examining the body of the view $\bar{V}$ constructed in the proof, we notice that its subgoals have a special form. First, all variables have the same values of $k_1, \ldots, k_m$ in Eq. 1. Second, for every $j = 1, \ldots, (k_1 + \ldots + k_m)$, if we apply the substitution π_j to $\bar{V}$ then each subgoal becomes a subgoal in one of the queries Q_i in the workload. It follows that we only need to include in $\mathcal{W}$ subgoals that correspond to tuples of subgoals, each belonging to some query Q_i: there may be multiple subgoals in

[3] As discussed, we do not impose any conditions on how ρ_i maps V's head variables.

the same Q_i, but no more than n_i, the total number of subgoals. The total number of such tuples is bound by an exponential in the size of $\mathcal{Q}$, independent on the schema. □

Example 2 We illustrate the main idea in the proof of Theorem 1 on the following example. Consider the view selection problem given by the schema $\mathcal{R} = \{T\}$, the queries:

$$Q_1(X_0^1) :- T(X_0^1, X_1^1), T(X_1^1, X_0^1)$$
$$Q_2(X_0^2) :- T(X_0^2, X_1^2), T(X_1^2, X_2^2), T(X_2^2, X_0^2)$$
$$Q_3(X_0^3) :- T(X_0^3, X_1^3), T(X_1^3, X_2^3), T(X_2^3, X_3^3), T(X_3^3, X_4^3), T(X_4^3, X_0^3)$$

and some space bound B. The queries compute cycles of length 2, 3, and 5, respectively. Consider a configuration $\mathcal{V} = \{T, V\}$ where V is the following view:

$$V(Z_0) :- T(Z_0, Z_1), T(Z_1, Z_2), \ldots, T(Z_{118}, Z_{119}), T(Z_{119}, Z_0)$$

V computes cycles of length 120. Assume that the total space used by T and V is less than B, and assume the following rewritings to be the cheapest for the given configuration:

$$Q_1'(X_0^1) :- \theta_1(V(Z_0)), T(X_0^1, X_1^1), T(X_1^1, X_0^1)$$
$$Q_2'(X_0^2) :- \theta_2(V(Z_0)), T(X_0^2, X_1^2), T(X_1^2, X_2^2), T(X_2^2, X_0^2)$$
$$Q_3'(X_0^3) :- \theta_3(V(Z_0)), T(X_0^3, X_1^3), T(X_1^3, X_2^3), T(X_2^3, X_3^3), T(X_3^3, X_4^3), T(X_4^3, X_0^3)$$

where $\theta_1(Z_0) = X_0^1$, $\theta_2(Z_0) = X_0^2$, $\theta_3(Z_0) = X_0^3$. These are indeed equivalent rewritings because, for example, each cycle of length 2 is also a cycle of length 120 (simply walk 60 times around the cycle). The main idea in the theorem is to replace $\mathcal{V}$ with $\mathcal{V}' = \{T, \bar{V}\}$, such that $\bar{V}$ has "few" subgoals and is contained in V. We illustrate how the theorem's proof constructs $\bar{V}$ for this example. The homomorphisms $\rho_i : V \to Q_i$, $i = 1, 2, 3$, are defined as follows: $\rho_1(Z_j) = X_{j \bmod 2}^1$, $\rho_2(Z_j) = X_{j \bmod 3}^2$, and $\rho_3(Z_j) = X_{j \bmod 5}^3$. The view $\bar{V}$ has $2 \times 3 \times 5 = 30$ variables, which we denote U_{abc}, $0 \le a < 2$, $0 \le b < 3$, $0 \le c < 5$, in one-to-one correspondence with triples (X_a^1, X_b^2, X_c^3), and is defined to be $\bar{V} = \bar{\rho}(V)$. Notice that $\bar{V}$ has only 30 subgoals: for example the subgoals $T(Z_0, Z_1), T(Z_{30}, Z_{31}), T(Z_{60}, Z_{61})$, and $T(Z_{90}, Z_{91})$ are mapped into the same subgoal in $\bar{V}$, namely $T(U_{000}, U_{111})$. It is easy to see that $\bar{V}$ computes cycles of length 30, i.e., $\bar{V}$ is isomorphic to:

$$\bar{V}(Y_0) :- T(Z_0, Z_1), T(Z_1, Z_2), \ldots, T(Z_{28}, Z_{29}), T(Z_{29}, Z_0)$$

under the isomorphism $Z_j \longrightarrow U_{(j \bmod 2)(j \bmod 3)(j \bmod 5)}$. In addition, $\bar{V} \subseteq V$, since every cycle of length 30 is also a cycle of length 120 (simply walk four times around the cycle). Finally, each of the three rewritings Q_1', Q_2', Q_3' that use V can be modified into rewritings Q_1'', Q_2'', Q_3'' that use $\bar{V}$, by replacing the terms $V(X_0^1)$, $V(X_0^2)$, $V(X_0^3)$ with $\bar{V}(X_0^1)$, $\bar{V}(X_0^2)$, and $\bar{V}(X_0^3)$, respectively.

3.2 CSA: an exponential-time lower bound under the product cost model

In the remainder of the section, we show that the number of views in an optimal view configuration under CSA can be exponential in the size of the schema and the workload. This is a relatively surprising result and, in fact, none of the algorithms proposed for view-selection would consider such view configurations.

We start by considering a version of our cost model for joins where the cost of the join is proportional to the *product* of the sizes of the relations being joined, even though the coefficient α of the product can be arbitrarily small. For this *product* version of our cost model, we show that the number of views in an optimal view configuration can be exponential in the size of the schema and the workload. In Sect. 3.4, we describe the same exponential result for the *sum cost model*, that us, for the case where $\alpha = 0$ and the cost of a join is proportional to the *sum* of the sizes of the input relations *and* of the join result. We note that the proofs for the two cost models are substantially different, though they rely on the same general technique.

For both cost models, we show that there are an infinite number of non-isomorphic inputs to the view selection problem, such that for each of the inputs, the size of any optimal view configuration is exponential in the size of the input. We conclude that under the product and sum cost model assumptions, the view selection problem has an exponential-time lower bound.

We now consider the construction for the product-cost-model case. Assuming that $\alpha > 0$ in our cost model, we construct the inputs to the view selection problem, as follows: fix an integer parameter n. Consider a database schema $\mathcal{R}$ that includes two relations, S and T (referred to by the predicates s and t in the queries), of arity n. The workload $\mathcal{Q}$ includes three queries:

$$q_1(X_1, \ldots, X_n) :- s(X_1, \ldots, X_n)$$
$$q_2(X_1, \ldots, X_n) :- t(X_1, \ldots, X_n)$$
$$q_3(X_1, \ldots, X_n) :- s(X_1, \ldots, X_n), t(X_1, \ldots, X_n)$$

The first two queries simply ask for the database relations, while the third query asks for their intersection. If we are considering views to be materialized in a database, then the query q_3 suffices for the proof. Queries q_1 and q_2 are needed if we are creating a data warehouse. In the remainder of this section, we consider only q_3. We assume that the available storage is denoted by B, and later in the section we show that B should be

$$B = N^2 \times n^2 + N \times 2^{n(n+3)/2} + 2N + 2^{n+3} + 1,$$

where $N = \binom{n}{\lceil n/2 \rceil}$. (To simplify the notation, in the remainder of this section we only consider *even* values of the parameter n, so that $\lceil n/2 \rceil = n/2$. All the constructions and proofs can be extended in a straightforward way to the case of odd-valued n.)

Recall that one of the inputs to the view selection problem is a database schema with a set of associated statistics. Since we are making the complete statistics assumption, a query processor estimating the cost of query plans has access to

accurate estimates of sizes of views over the schema. Hence, in order to show that the number of views of an optimal view configuration can be exponential in the sizes of $\mathcal{R}$, $\mathcal{Q}$, and the binary representation of B, we need to produce a set of statistics on the database that will yield such a configuration.

The remainder of this section proceeds as follows: in Sect. 3.2.1, we fix a value of the parameter n and the corresponding triple $(\mathcal{R}, \mathcal{Q}, B)$ as above. For that triple, we construct a *database instance*, $\mathcal{D}$. We show that on the database instance $\mathcal{D}$, there exists an equivalent rewriting of the workload $\mathcal{Q}$ that uses an exponential number of views in the parameter n. More precisely, the number of views used in the rewriting is $\binom{n}{n/2}+1$. The corresponding view configuration is denoted by $\mathcal{V}$. We show that $\mathcal{V}$ satisfies the storage space constraint B.

Next, we show that there exists a constant n_0, such that for all integer values $n \geq n_0$ and when only left-linear query plans are considered, a rewriting of the workload $\mathcal{Q}$ that uses the view configuration $\mathcal{V}$ is optimal on the database instance $\mathcal{D}$ and under the storage space constraint B. This proof is rather tedious, and hence left to Appendix 6. The important aspect of this proof is that it can be generalized to any database instance $\mathcal{D}'$ that has the exact same statistics as $\mathcal{D}$. In particular, if every possible view on the database has the same size estimate on $\mathcal{D}$ and $\mathcal{D}'$, then all cost estimates that we show (in Appendix 6) to be true for database $\mathcal{D}$, will still hold on database $\mathcal{D}'$.

Hence, we show that there exist a database schema $\mathcal{R}$, set of complete statistics Σ on the database and a view configuration $\mathcal{V}$, such that $\mathcal{V}$ is an optimal view configuration for the workload $\mathcal{Q}$ on all databases with the statistics Σ (as long as $n \geq n_0$). As a result, we obtain the main result and conclude that the view selection problem has an exponential-time lower bound, under the product cost model and when only left-linear plans are considered.

3.2.1 A database with an exponential-size optimal rewriting

We now explain how to construct the specific database instance $\mathcal{D}$ for the database schema $\mathcal{R}$, the query workload $\mathcal{Q}$, and the storage space constraint B (see above), such that on this database instance $\mathcal{D}$ and under the product cost model ($\alpha > 0$), there exists a rewriting of the workload $\mathcal{Q}$ that uses a number of views that is exponential in n.

The view configuration we use is denoted by $\mathcal{V}$, and it includes all projections of the table $S \bigcap T$ on $n/2$ of its attributes. There are $N = \binom{n}{n/2}$ such views; we denote them by $v_1, \ldots, v_N$. In addition, $\mathcal{V}$ includes a projection of $S \bigcap T$ on its first attribute; we denote this view by v_0. Therefore, the total number of views in $\mathcal{V}$ is $N + 1 = \binom{n}{n/2} + 1$.

Recall that the workload $\mathcal{Q}$ includes three queries. For our purposes, q_3 is the only query we need to consider; we reproduce it here for convenience:

$q_3(X_1, \ldots, X_n) :- s(X_1, \ldots, X_n), t(X_1, \ldots, X_n).$

The query rewriting we will consider essentially inserts all the views in the configuration between the subgoal of s and subgoal of t. Formally, it is:

$q_3'(X_1, \ldots, X_n) :- s(X_1, \ldots, X_n), v_0(X_1),$

Table 1. Case $n = 2$: tables for the relations s and t

S:	X_1	X_2
1	a_1	a_2
2	b_1	b_2
5	A	a_2
6	b_1	A
7	A	c_2
8	d_1	A
$[F]$	$\mathcal{F}$	$\mathcal{F}$
$[I_S]$	$\mathcal{I}(S)$	$\mathcal{I}(S)$

T:	X_1	X_2
3	c_1	c_2
4	d_1	d_2
5	A	a_2
6	b_1	A
7	A	c_2
8	d_1	A
$[F]$	$\mathcal{F}$	$\mathcal{F}$
$[I_T]$	$\mathcal{I}(T)$	$\mathcal{I}(T)$

$v_1(X_{11}, \ldots, X_{1n/2}), v_2(X_{21}, \ldots, X_{2n/2}),$
$\ldots, v_N(X_{N1}, \ldots, X_{Nn/2}), t(X_1, \ldots, X_n);$

where each set of arguments $\{X_{i1}, \ldots, X_{in/2}\}$ is the appropriate subset of $\{X_1, \ldots, X_n\}$. It is easy to see the the rewriting is equivalent to q_3.

Intuitively, our intention is the optimal query execution plan for the rewriting be the one that starts with s, and joins it with v_0, the result with v_1, and so on, until joining with t. We denote this plan by $\mathcal{P}^*$. Note that in effect, the plan starts with the tuples in s and the results of the intermediate joins are successively non-growing subsets of s, until finally intersecting with t. To make this the optimal plan, we want that every v_i, for $i > 0$ remove *one* special tuple t_i from the intermediate result. Furthermore, it should be the case that t_i is not removed by the join with any other view in the configuration. We call these tuples *stubborn* tuples, because t_i can only be removed by v_i. To guarantee that every stubborn tuple is eliminated by exactly one view, we construct a set of carefully crafted *eliminator* tuples.

As we explain the construction, we illustrate it for the case of $n = 2$. In this case, since v_0 is the same as v_1, the view configuration includes two views. Table 1 shows a possible database instance $\mathcal{D}$, i.e., the contents of the tables S and T. In Table 2 we show the views V_1 and V_2, and Table 3 shows two possible intermediate results ($S\&V_1$ and $S\&V_2$). Recall that V_1 and V_2 are defined as follows:

$v_1(X_1) :- s(X_1, X_2), t(X_1, X_2).$
$v_2(X_2) :- s(X_1, X_2), t(X_1, X_2).$

The database instance includes the following sets of tuples:

1. The *stubborn* tuples: as explained above, these are tuples that are in S but not in T (and symmetrically, in T but not in S). In Table 1, tuples 1 and 2 are the stubborn tuples of S and tuples 3 and 4 are the stubborn tuples of T. Consider the stubborn tuples of S. There are N stubborn tuples, t_1, t_2, ... , t_N. Each stubborn tuple t_i in R has n distinct values for each of its attributes, $a_{i1}, a_{i2}, \ldots, a_{in}$. For any two stubborn tuples t_i and t_j in R where $i \neq j$, t_i and t_j do not have values in common; in other words, each value in the domain of the stubborn part of R occurs in exactly one attribute of exactly one tuple there.
   There is a 1-1 mapping between the stubborn tuples in S and subsets of $n/2$ attributes of S, and hence with the views in the warehouse. We assume some ordering on these views

and denote by V_i the ith such view, and correspondingly by $attr(i)$ the ith such subset. Note that $attr(i)$ is the schema of the view V_i.

2. *Eliminator tuples:* these tuples guarantee that each stubborn tuple can only be eliminated by a single view in the configuration. There are two sets of eliminator tuples (one for the stubborn tuples of S and one for those in T), but both sets are in both relations S and T, and hence in the intersection (see tuples 5–8 in Table 1).
   Consider the eliminator tuples for the stubborn tuples of S. For each stubborn tuple $t_i = (a_{i1}, a_{i2}, \ldots, a_{in})$ in S, there is a set $E(t_i)$ of $n/2$ eliminator tuples (thus the total number of eliminator tuples for the stubborn tuples of S is $N * n/2$).
   Let A be a new constant that appears nowhere else in S or T. Recall that each stubborn tuple t_i has an associated set of attributes $attr(i)$ of size $n/2$. For each attribute A_i in $attr(i)$, we construct one eliminator tuple for t_i, which is the tuple resulting by replacing the value of A_i in t_i by the constant A.
   Note that the construction of the eliminator tuples ensures that the projection of S and T on $attr(i)$ (which is the view V_i) does not have a projection of the stubborn tuple t_i. On the other hand, the projection of S and T on any other set $attr(j)$ of $n/2$ attributes of S does have a projection of t_i. As a result, the tuple t_i is eliminated from the partial relation in the plan $\mathcal{P}^*$ only after the join with V_i. Furthermore, the stubborn tuples are completely eliminated only after joining S with all the views in $\mathcal{V}$ (hence, the choice of their name).
   Finally, note that the eliminator tuples for the stubborn tuples of S and those for the stubborn tuples of T are over different domains.
3. The *intimidator* tuples: here again, there is a set of intimidator tuples for S and a set for T, but neither set belongs to the intersection of S and T. The role of the intimidator tuples is to ensure that it is cheaper to eliminate the stubborn tuples than to join an intermediate result that still contains stubborn tuples with T. This is achieved by increasing the size of T so a join with it becomes expensive. Note that because we begin by joining S with V_0, we immediately lose all the intimidator tuples of S and don't carry them along through all the intermediate joins.
   Specifically, the intimidator tuples are constructed over a domain that is disjoint from any of the other constants in S or T. The domains of the intimidator tuples in S and in T are also disjoint. The number of intimidator tuples in S (and in T) should be greater than $(N + N^2 * n + N * 2^{n*(n+3)})^2$ (for reasons that become clear in the full version of the proof). In the case of $n = 2$ this translates into 4,235,364 tuples (represented by the last rows in Table 1).
4. *Necessary fat tuples:* the purpose of the necessary-fat tuples in S and in T is to prevent us from constructing a cheap plan for q_3 that includes projections on $S \cap T$ that are *wider* than $n/2$ attributes. This is done by ensuring that such a wider projection is much bigger than a projection on $n/2$ attributes, and hence more expensive to join with. The necessary fat tuples are built over a domain $\mathcal{F}$, that is disjoint from the domains of the other tuples in S and T. The size of $\mathcal{F}$ is $F = 2^{n+3}$ (also for reasons described in the proof). There are F^n necessary-fat tuples in S and in T. They are all the possible different tuples that one can build from the domain $\mathcal{F}$. In the case of $n = 2$, the domain $\mathcal{F}$ includes 32 constants, and hence there are $32^2 = 1024$ fat tuples (represented by the next to last rows in Table 1).

Table 2. Case $n = 2$: tables for the views v_1 and v_2

V_1 :	X_1
$(5,7)$	A
(6)	b_1
(8)	d_1
$([F])$	$\mathcal{F}$

V_2 :	X_2
(5)	a_2
$(6,8)$	A
(7)	c_2
$([F])$	$\mathcal{F}$

Table 3. Case $n = 2$: the tables $S\&V_1$ and $S\&V_2$

(a) $S\&V_1$:

	X_1	X_2
2	b_1	b_2
5	A	a_2
6	b_1	A
7	A	c_2
8	d_1	A
$[F]$	$\mathcal{F}$	$\mathcal{F}$

(b) $S\&V_2$:

	X_1	X_2
1	a_1	a_2
5	A	a_2
6	b_1	A
7	A	c_2
8	d_1	A
$[F]$	$\mathcal{F}$	$\mathcal{F}$

By obtaining upper bounds on the sizes of the views $v_0, \ldots, v_N$ in the view configuration $\mathcal{V}$, it is easy to show that $\mathcal{V}$ satisfies the storage space constraint B for all values of the parameter n. In the appendix, we show that the plan $\mathcal{P}^*$ is indeed the optimal query plan for q_3.

3.2.2 An exponential-time lower bound

The previous section described a particular database instance $\mathcal{D}$ for which q_3' is an optimal rewriting of q_3 under the product cost model ($\alpha > 0$). The same claim holds for any database whose statistics agree with those of $\mathcal{D}$. Recall that the input to CSA includes a set of statistics, which is a function that maps every view over $\mathcal{R}$ into an integer that specifies its size. Hence, it follows that if we denote the set of statistics of $\mathcal{D}$ by Σ, then the CSA problem with the schema $\mathcal{R}$ and the statistics Σ will have an exponentially-sized optimal configuration under the product cost model:

Theorem 2 *Consider the view selection problem $(\mathcal{R}, \mathcal{Q}, B)$ above, where Σ is the set of statistics for the database instance $\mathcal{D}$ above, and in the cost model, $\alpha > 0$ (the product cost model). Assume we restrict query plans to be left-linear. Then an optimal view configuration consists of the views $V_0, V_1, \ldots, V_N$ defined above ($N = \binom{n}{n/2}$).* □

As a result, the following theorem establishes a lower bound on the view selection problem for CSA under the product cost model.

Theorem 3 *Under CSA and the product cost model, the view selection problem has an exponential-time lower bound when only left-linear plans are considered.* □

3.3 Product cost model: extensions of the lower bound

In this section we explore the limitations of the theorem of the previous section.

The number of join and head variables. In the theorem of the previous section, the queries in the workload have a number of head and join variables that depends on the database schema. We can relax this restriction, and show that the result holds even if the number of head variables is bounded or if the number of join variables is bounded.

Suppose k is a constant upper bound on the number of join and/or head variables in queries. For each integer value n, consider a database schema $\mathcal{R}$ that consists of the schemas of two relations, s and t, each of arity $n + k$. We consider a query workload $\mathcal{Q}$ with two parts:

1. A modification of the query q_3 from the beginning of Sect. 3:

$$q(Y_1, \ldots, Y_k) :- s(X_1, \ldots, X_n, Y_1, \ldots, Y_k), \\ t(Z_1, \ldots, Z_n, Y_1, \ldots, Y_k);$$

   here q is a join of the relations s and t on k attributes $Y_1, \ldots, Y_k$, which are also the distinguished variables of the query.
2. The remainder of the workload $\mathcal{Q}$ constitutes all queries that are projections of either s or t on k attributes each; because k is a constant with respect to n, the number of such queries is polynomial in the parameter n.

With this workload, we can prove the following result:

Theorem 4 *Under CSA, under the product cost model, and considering only left-linear plans, the view selection problem has an exponential-time lower bound even if all the queries in the input have a bounded number of variables in their head or if all the queries in the input have a bounded number of join variables.* □

The shape of query execution plans. Theorems 2, 3 and 4 hold when we restrict ourselves to left-linear trees. The question of whether, under the product cost model, the exponential-time lower bound still holds when we consider bushy plans remains open. In fact, we can show that an optimal bushy plan for q_3 in D requires only a single view which is a projection of S (or T) on a single attribute.

However, $\mathcal{P}^*$ is still an optimal plan under the product cost model ($\alpha > 0$) when we consider all bushy plans in which S (or T) is the last relation being joined (that is, either S or T is the right child of the root of the join tree). This is an important observation in a distributed query processing context. In this context, we may have S and T stored in different locations, and may store views on T (or on the intersection of S and T) in S's node to perform local filtering before we send data over the network to evaluate the intersection of S and T. Hence, our result implies that the number of local views we may want to store on a remote source may be exponential in the size of the schema. □

The role of projections. Our last result in this section identifies projections as being the culprit for the exponential-time lower bound.

Theorem 5 *Given any database schema $\mathcal{R}$, any workload $\mathcal{Q}$, and any space constraint B, if an optimal view configuration includes only projection-free views, then the number of views in the configuration is at most quadratic in the size of the schema and workload, for both the product and sum cost models.* □

Proof sketch. The idea of the proof is as follows: consider any query q in the workload $\mathcal{Q}$. In the definition of q, the number M of selection and join predicates is no more than quadratic in the number of variables and constants in the body of q. Suppose $\mathcal{D}$ is any database instance with the schema $\mathcal{R}$, and $\mathcal{P}$ is any plan that uses views and computes the query q on the database $\mathcal{D}$.

Suppose a view v used in the plan $\mathcal{P}$ is defined using a selection or join predicate p from the definition of the query q. Consider the point in the computation of the plan $\mathcal{P}$ where the table for the view v is joined with the intermediate relation that has been computed so far. Because the views in the plan $\mathcal{P}$ do not use projections, once the table for v has been used in the plan $\mathcal{P}$, from that point and until the end of the computation of the plan $\mathcal{P}$ all its intermediate relations satisfy the predicate p. Therefore, it takes at most M views to compute the query q on any database, where M is quadratic in the number of variables and constants in the body of the query q. □

3.4 CSA: an exponential-time lower bound under the sum cost model

In Sects. 3.2 and 3.3 we considered the *product* cost model, that is, the case where the cost of a join is proportional to the product of the cardinalities of the joined relations. However, many efficient join algorithms have a cost that is proportional to the *sum* of the sizes of the input and output relations; we call this cost model the *sum* cost model. In this section we show that the technique described in Sect. 3.2 can be used to show that under the sum cost model, the view selection problem also has an exponential-time lower bound.

The exponential example from Sect. 3.2 cannot be carried over directly to the sum-cost-model case for the following reason. The proof of in Sect. 3.2 relied on the fact that while computing the query q_3, directly joining the relations S and T would be too expensive (since only a small number of tuples would match). Hence, we were able to show that adding an exponential number of views between S and T reduced the overall cost. In the sum-cost model, the join of S and T without views is the cheapest way of computing the query q_3. The problem is that under the sum cost model the query q_3 from Sect. 3.2 does not provide any "expensive and wasteful" computation that was key to the previous proof. In fact, it seems that under the sum cost model an exponential example cannot be constructed, because adding subgoals (views) to a rewriting of a query seems to only increase the costs of computing the query.

Surprisingly, this intuition is wrong. We now show that a different workload and query will establish an exponential lower bound for the sum-cost model case as well.

The schema and the workload. Fix an integer parameter n. Consider a database schema $\mathcal{R}$ that includes n relations, S_1 through S_n (referred to by the predicates s_1 through s_n in the queries), each of arity $n + 1$. The workload $\mathcal{Q}$ includes $n + 1$

queries:

$$\begin{aligned} r_1(X_1,\ldots,X_{n+1}) &:- s_1(X_1,\ldots,X_{n+1}).\\ r_2(X_1,\ldots,X_{n+1}) &:- s_2(X_1,\ldots,X_{n+1}).\\ &\ldots\\ r_n(X_1,\ldots,X_{n+1}) &:- s_n(X_1,\ldots,X_{n+1}).\\ r_{n+1}(X_1,\ldots,X_n,Y_1,\ldots,Y_n) &:- s_1(X_1,\ldots,X_n,Y_1),\\ &\quad s_2(X_1,\ldots,X_n,Y_2),\ldots,\\ &\quad s_n(X_1,\ldots,X_n,Y_n). \end{aligned}$$

The first n queries simply ask for the database relations, while the last query asks for their natural join on n first attributes of each relation. Notice that the last attribute of each relation goes to the answer to the query r_{n+1}; as a result, the relation r_{n+1} has n more attributes than each database relation. If we are considering views to be materialized in a database, then the query r_{n+1} suffices for the proof. Queries r_1 through r_n are needed if we are creating a data warehouse. In the remainder of this section, we consider only the query r_{n+1}.

We assume that the available storage is denoted by B. For the sum-cost-model case, B should be proportional to $N^2\times n^2+N\times 2^{n(n+3)/2}$, where $N=\binom{n}{n/2}$. Notice that the storage space constraint B here is asymptotically the same as the storage space constraint for the product-cost-model case.

Using this query workload, storage space constraint, and a database instance we are going to describe, we will be able to show that there exist a set of complete statistics Σ on the database and a view configuration $\mathcal{V}$, such that $\mathcal{V}$ is an optimal view configuration for the workload $\mathcal{Q}$ on all databases with the statistics Σ. As a result, we obtain the exponential-time lower bound under the sum cost model.

The view configuration. We now describe the view configuration $\mathcal{V}$ we use in this section. Consider again the query

$$\begin{aligned} r_{n+1}(X_1,\ldots,X_n,Y_1,\ldots,Y_n) &:- s_1(X_1,\ldots,X_n,Y_1),\\ &\quad s_2(X_1,\ldots,X_n,Y_2),\ldots,\\ &\quad s_n(X_1,\ldots,X_n,Y_n), \end{aligned}$$

and consider a projection of r_{n+1} on the first n of its attributes:

$$p(X_1,\ldots,X_n) : -r_{n+1}(X_1,\ldots,X_n,Y_1,\ldots,Y_n).$$

The view configuration $\mathcal{V}$ that we use comprises all projections of the table P on $n/2$ of its attributes. There are $N=\binom{n}{n/2}$ views in the configuration $\mathcal{V}$; we denote them by $v_1,\ldots,v_N$. Recall that N is exponential in the parameter n.

The query rewriting we consider essentially inserts all the views in the configuration $\mathcal{V}$ between the subgoal of s_1 and subgoal of s_2. Formally, it is:

$$\begin{aligned} r_{n+1}(X_1,\ldots,X_n,Y_1,\ldots,Y_n) : -&s_1(X_1,\ldots,X_n,Y_1),\\ &v_1(X_{11},\ldots,X_{1n/2}), v_2(X_{21},\ldots,X_{2n/2}),\ldots,\\ &v_N(X_{N1},\ldots,X_{Nn/2}),\\ &s_2(X_1,\ldots,X_n,Y_2),\ldots,s_n(X_1,\ldots,X_n,Y_n). \end{aligned}$$

The database instance. We now explain how to construct the specific database instance $\mathcal{D}$ for the database schema $\mathcal{R}$, query workload $\mathcal{Q}$, and storage space constraint B, such that on this database instance $\mathcal{D}$ and under the sum cost model, the rewriting of the workload $\mathcal{Q}$ that uses all the views in the view configuration $\mathcal{V}$, is optimal on $\mathcal{D}$.

Intuitively, our intention is the optimal query execution plan for the rewriting be the one that starts with S_1, joins it with V_1, joins the result with V_2, and so on, until joining with the remaining database relations $S_2, S_3,\ldots,S_n$. We denote this plan by $\mathcal{P}^*$. Note that just like in Sect. 3.2, the plan starts with the tuples in a database relation (S_1) and the results of the intermediate joins are successively non-growing subsets of the relation, until finally joining with the remaining database relations (S_2 through S_n). To make this an optimal plan, we want that every view v_i remove only special (stubborn) tuples t_i from the intermediate result. Just like in Sect. 3.2, to guarantee that every stubborn tuple is eliminated by exactly one view, we construct a set of carefully crafted *eliminator* tuples. Here are the details of the construction.

To build the relations $S_1, S_2,\ldots,S_n$, we start by building n relation templates $T_1, T_2,\ldots,T_n$, where each T_i is an n-ary relation; all templates are constructed on pairwise disjoint domains. Each relation template T_i has N stubborn tuples and $Nn/2$ corresponding eliminator tuples. The stubborn and eliminator tuples for each T_i are built exactly as described in Sect. 3.2.

The next step in building the relations $S_1, S_2,\ldots,S_n$ is to introduce fat tuples, also described in Sect. 3.2. The domain $\mathcal{F}$ of the fat tuples is disjoint with the union of the domains of the relation templates T_1 through T_n.

We now construct the projection P_i of each database relation S_i on its first n attributes (recall that each S_i has $n+1$ attributes). Consider the case $n=3$. We have three relation templates, T_1, T_2, T_3, and a set of fat tuples, all built on pairwise disjoint domains. The relations P_1, P_2, and P_3 share the fat tuples and all the eliminator tuples of the templates T_1 through T_3; in other words, all fat and eliminator tuples we have constructed so far go to the intersection of the relations P_1, P_2, and P_3. Let the remaining tuples of the relation P_1 be the union of the stubborn tuples of T_2 and T_3. (Note that the relation P_1 does not include any of the stubborn tuples of T_1, because the relations T_1, T_2, and T_3 are built on pairwise disjoint domains.) The stubborn tuples of P_2 and P_3 are constructed analogously. This concludes the construction of the relations P_1, P_2, and P_3. The construction can be extended to any value of n in a straightforward way; its output comprises n n-ary relations $P_1,\ldots,P_n$, where each P_i is a projection of the database relation S_i on its first n attributes. We will see shortly how to construct the remaining part of each relation S_i.

Note that even though we don't know the values of the last attribute of any relation S_i, by construction of the projections of S_i on their first n attributes we have the following guarantees on the plan $\mathcal{P}^*$. First, each view v_i in the plan $\mathcal{P}^*$ removes only a certain part of stubborn tuples of the relation S_1 (recall that the join of the relation S_1 with each v_i in the plan $\mathcal{P}^*$ is a natural join on the first n attributes of S_1). Second, *all* views in the configuration $\mathcal{V}$ are required to remove all stubborn tuples from the relation S_1. Finally, if we modify the plan $\mathcal{P}^*$ by

swapping the relation S_1 with any other S_i, $i > 0$, these two properties will still hold.

So far we have been able to reuse parts of the technique from the product-cost-model case. Namely, we have constructed (parts of) database relations S_1 through S_n using stubborn and eliminator tuples, and we have built a plan $\mathcal{P}^*$ that has an exponential number of *useful* views $v_1, \ldots, v_N$. What we don't have yet is the expensive-and-wasteful component in the computation of the query r_{n+1}, such that we could use that component to offset the costs of using an exponential number of views in the plan $\mathcal{P}^*$.

Recall that we haven't yet specified the values of the $(n+1)$st attribute in each of the database relations $S_1, \ldots, S_n$. We are going to specify these values in a way that would create an expensive and wasteful computation in all plans for the query r_{n+1} that have fewer than $N = \binom{n}{n/2}$ views.

We take a unary relation R that has k different tuples; we will determine the value of k shortly. We finalize the construction of each database relation S_i, $1 \leq i \leq n$, by making S_i a cross-product of the corresponding relation P_i (we have built P_i as a projection of S_i on its first n attributes) with the unary relation R. (Now each stubborn, eliminator, or fat tuple in the relation P_i corresponds to k tuples in the relation S_i.) This concludes the construction of the database $\mathcal{D}$ that comprises relations $S_1, \ldots, S_n$.

Optimality of the configuration. The details of the proof for the sum-cost model case are similar in spirit to those of the product-cost model case. In the rest of this section we provide a brief explanation why the view configuration we designed is indeed the optimal one.

Let us see how the structure of the relations $S_1, \ldots, S_n$ in the database $\mathcal{D}$ creates the expensive and wasteful computation in all plans for the query r_{n+1} that have fewer than N views. By construction of the relations $S_1, \ldots, S_n$, each stubborn tuple in the relations is shared by exactly $n-1$ of the n relations, and for each subset of the set $\{S_1, \ldots, S_n\}$ of size $n-1$ (for example, the subset $\{S_2, \ldots, S_n\}$), there exists a stubborn tuple that is present in each of the $n-1$ relations but is not in the remaining nth relation (S_1 in the example).

Consider the relations $S_1, S_2, \ldots, S_{n-1}$. From the observation above, there exists a stubborn tuple t that is present in each of the $n-1$ relations but is not in the relation S_n. Therefore, the tuple t does not go to the answer to the query r_{n+1} on the database $\mathcal{D}$. Recall the construction of the relations $S_1, \ldots, S_n$: each relation S_i is a cross-product of a relation P_i with a unary relation R that has k tuples. Therefore, in any of the relations $S_1, \ldots, S_{n-1}$, there are k tuples that have the same projection t' on the first n attributes as t does. We denote this set of k tuples by $\mathcal{T}$.

Now consider a view v in the view configuration $\mathcal{V}$, such that v is the only view that removes the tuples $\mathcal{T}$ (by removing their projection t') from the relation S_1. Suppose a plan $\mathcal{P}$ computes the query r_{n+1} by joining the relations $S_1, S_2, \ldots, S_n$, in this order. The plan $\mathcal{P}$ can use views; our only restriction is that the plan do not use the view v. Then, in the plan $\mathcal{P}$, the tuples $\mathcal{T}$ generate k^2 tuples in the intermediate relation $S_1 \& S_2$ (more precisely, in the intermediate relation that results from joining the relation S_2 with the preceding intermediate relation – recall that the plan $\mathcal{P}$ can have views). The reason is that the join of the relations S_1 and S_2 in the query r_{n+1} is a natural join on the first n attributes, but the last attribute of each of S_1 and S_2 goes unchanged to the relation $S_1 \& S_2$. Recall that all the tuples in $\mathcal{T}$ has the same projection on the first n attributes, and all the tuples have different values in the last attribute. Therefore, after the k tuples $\mathcal{T}$ in S_1 are joined with the k tuples $\mathcal{T}$ in S_2, the relation $S_1 \& S_2$ has k^2 tuples that have the same projection (t') on the first n attributes as $\mathcal{T}$ does, but have all possible pairs of the k values in the last two attributes.

By the same token, the k tuples $\mathcal{T}$ result in k^3 tuples in the relation $S_1 \& S_2 \& S_3$, in k^4 tuples in $S_1 \& S_2 \& S_3 \& S_4$, and so on. (Recall that each of $S_1, \ldots, S_{n-1}$ has the tuples $\mathcal{T}$). Finally, the k tuples $\mathcal{T}$ result in k^{n-1} tuples in the relation $S_1 \& \ldots \& S_{n-1}$. Under the sum cost model, the cost of obtaining the k^{n-1} tuples in the relation $S_1 \& \ldots \& S_{n-1}$ is *at least* k^{n-1}.

Recall that the tuples $\mathcal{T}$ all have the same projection t' on the first n attributes. In addition, recall that by construction of the database $\mathcal{D}$, the relation S_n does *not* have any tuples with projection t'! Therefore, the k^{n-1} tuples that are generated by the tuples $\mathcal{T}$ in the relation $S_1 \& \ldots \& S_{n-1}$, are *dangling* w.r.t the relation S_n (i.e., the tuples $\mathcal{T}$ do not contribute to the result of the join $S_1 \& \ldots \& S_n$). It follows that when the relation $S_1 \& \ldots \& S_{n-1}$ is joined with the relation S_n, via a natural join on the first n attributes, none of the tuples in the result are generated by the tuples $\mathcal{T}$! We conclude that in the plan $\mathcal{P}$ for the query r_{n+1}, we have an expensive and wasteful computation with the cost at least k^{n-1}.

Note that by construction of the stubborn tuples in the relations $S_1, \ldots, S_n$, the reasoning above also holds for all plans for the query r_{n+1} that involve arbitrary permutations of the relations $S_1, \ldots, S_n$ and that do not use all the views in the view configuration $\mathcal{V}$.

To summarize the above, we have constructed the relations $S_1, \ldots, S_n$ in such a way that whenever a plan for the query r_{n+1} does not include all the views in the view configuration $\mathcal{V}$, the cost of the plan involves an expensive and wasteful computation, with the cost at least k^{n-1}. Note that we have so far refrained from assigning a value to k. Now the time has come: we want the value k^{n-1} to offset some costs in the computation of the plan $\mathcal{P}^*$, which first joins the relation S_1 with all the views in the view configuration $\mathcal{V}$, and then joins the result with the remaining database relations. Because the plan $\mathcal{P}^*$ joins the relation S_1 with all the views in $\mathcal{V}$, the result of the join does not have any stubborn tuples. As a result, the evaluation of the plan $\mathcal{P}^*$ does not involve the expensive and wasteful computation of k^{n-1} tuples. If we make the value k^{n-1} greater than the cost of joining the relation S_1 with all the views in $\mathcal{V}$ in the plan $\mathcal{P}^*$, it is easy to see that under the sum cost model, the plan $\mathcal{P}^*$ will be cheaper than any plan that does not use *all* views in the set of views $\mathcal{V}$. We conclude that if a plan for the query r_{n+1} uses a proper subset of the views in $\mathcal{V}$, the plan is bound to be more expensive on the database $\mathcal{D}$ than the plan $\mathcal{P}^*$ with all the views in $\mathcal{V}$ (an exponential number of views).

It remains to consider plans for the query r_{n+1} on the database $\mathcal{D}$ that use other views than the views in $\mathcal{V}$. Similarly to the product-cost-model case, because of our choice of the storage space constraint B and by construction of the database $\mathcal{D}$, we can show that any view not in $\mathcal{V}$ either does not satisfy one of the constraints of view selection (in particular,

the storage space constraint), or cannot be part of an equivalent rewriting of the query r_{n+1}. We therefore establish the following theorem.

Theorem 6 *Under CSA and the sum cost model, the view selection problem has an exponential-time lower bound when only left-linear plans are considered.* □

Similarly to the product-cost model case, the lower bound holds also for bushy trees as long as the last relation in the plan is a stored relation (i.e., not a view). As in the product cost model, we can strengthen the lower bound result by showing that it holds even with certain restrictions on the query:

Theorem 7 *Under CSA, under the sum cost model, and considering only left-linear plans, the view selection problem has an exponential-time lower bound even if all the queries in the input have a bounded number of variables in their head or if all the queries in the input have a bounded number of join variables.* □

4 The partial statistics assumption

We consider now the view selection problem under the *partial statistics assumptions*, PSA, when only limited statistics over the database are available for estimating the view sizes. In this case our goal is to find a view configuration $\mathcal{V}$ whose total *estimated* size is at most B (the space bound) and that optimizes the *estimated* cost $C(\mathcal{R}, \mathcal{V}, \mathcal{Q})$ of the workload. This scenario is more likely in practice, and is similar to that of an optimizer in a relational database system that has to base its cost estimation on limited database statistics. The main results in this section are that the number of views in an optimal configuration of a view selection problem is bounded by a polynomial, and that the number of subgoals in any view that is part of an optimal solution is bounded linearly in the largest number of subgoals occurring in some query in the workload. The results, however, hold only under certain assumptions of the selectivity estimation function, which we need to discuss first.

Database systems maintain a collection of statistics on the database to perform size estimates. Statistics range significantly in complexity, from simple numbers to complex histograms. Examples include: the cardinality of a base table; the number of distinct values in a given field; the most frequent values in a given field; an equiwidth histogram on a certain attribute of some base table. Size estimators use heuristics to compute the size of query answers from these statistics. Examples of well-known simple heuristics are those for joins and selections. Given cardinalities N_R and N_S of the tables R and S, the size of $R \bowtie S$ is estimated to be $c \times N_R \times N_S$ where c is the *join selectivity factor*, and the size of a selection $\sigma_b(R)$ is estimated to be $d \times N_R$, where d is the selectivity of the Boolean condition b.

The database statistics, denoted $\Sigma_{\mathcal{R}}$, play now a central role in the size estimator function, and we denote by $E(\mathcal{R}, V)$ the estimated size of a view V over the database with schema $\mathcal{R}$. Our goal is to analyze the complexity of the view selection problem when the function E is a parameter.

Obviously, artificially chosen functions E could place the view selection problem anywhere between trivial and extremely complex. For example, if E estimates the size of every view with one or more joins to be prohibitively large, then there exists at most one valid view configuration, namely that materializing exactly the base relations (in a warehouse design setting, where we have access only to the views), or not materializing anything (in the query optimization setting, where we have access to the base tables anyway): in this case the problem is trivial. On the other hand E may estimate the size of very complex views, with thousands of columns, as being artificially low, forcing us to inspect an extremely large search space: here the problem becomes overly complex.

Without restricting too much the choice of the estimator function E, we focus our discussion on functions that are more likely to be used in practice. Our results in this section hold even under very generous assumptions about E.

Modeling statistics. We model database statistics as a collection of two kinds of numbers: factors and cardinalities. Factors are real numbers between 0 and 1, e.g., join selectivity factors, the frequency of a given value in a field, etc. Examples of cardinalities are the number of distinct tuples in a base table, the number of distinct values in a field, the number of values in a certain interval of a histogram, etc. We write $\Sigma = \{c_1, \ldots, c_p, N_1, \ldots, N_q\}$, where $c_1, \ldots, c_p$ are factors and $N_1, \ldots, N_q$ are cardinalities. Statistics are always associated with a particular database schema $\mathcal{R}$. Thus, the numbers p and q in Σ depend on $\mathcal{R}$. The complexity of the view selection problem, which is a function on $\mathcal{R}$, $\mathcal{Q}$ and B, will implicitly also be a function of p and q. Furthermore, statistics can be specific to the particular relations involved (e.g., the join selectivity of R and S), or can be generic and used whenever more specific statistics are not available.

Example 3 Consider the schema $\mathcal{R}$ with a binary relation R, a ternary relation S, and the statistics $\Sigma = \{c_{R_2R_1}, c_{S_2}, c_{\bowtie}, c_{\sigma}, N_R, N_S\}$. Here $c_{R_2R_1}$ is the join selectivity of $R(X,Y) \bowtie R(Y,Z)$; c_{S_2} is the selectivity factor for a selection on the second attribute of S; $c_{\bowtie}$ is a generic join selectivity that the function E will use for all other joins; c_{σ} is a generic selectivity factor for all other selections. N_R, N_S are the cardinalities of R and S, respectively. Consider the view:

$$V(X,Y,Z,U) :- R(X,Y), R(Y,Z), S(U, \text{“}Smith'', Z).$$

An estimator may compute the size of the view by the following product:

$$E(\mathcal{R}, V) = c_{\bowtie} \times c_{R_2R_1} \times c_{\sigma} \times N_R^2 \times N_S.$$

Thus, the size is a product of the selectivity factors, for the joins and the selections, and of the cardinalities of the base tables.

Example 4 The following example involves a projection. We note that in practice, estimating the size of a projection (i.e., the `select distinct ...` SQL statement) is much harder than for selections and joins, and there are no widely used robust techniques to address this problem. The following is one of the commonly used estimators. Consider the view:

$$V(X,U) :- R(X,Y), S(Y,Z), T(Z,U)$$

the following size estimation may be used:

$$E(\mathcal{R}, V) = (c_{\bowtie})^2 \times N_{R_1} \times N_S \times N_{T_2}$$

where N_{R_1} is the number of distinct values in the first column of R, N_S is the number of tuples in S, and N_{T_2} is the number of distinct values in the second column of T.

In summary, our discussion considers the class of *multiplicative size estimators*, which captures most estimators used in practice:

Definition 2 *Multiplicative estimators A size estimator E on the statistics $\Sigma = \{c_1, \ldots, c_p, N_1, \ldots, N_q\}$ is called* multiplicative *if, for every view V:*

$$E(\mathcal{R}, V) = c_1^{\gamma_1} \times \ldots \times c_p^{\gamma_p} \times N_1^{\delta_1} \times \ldots \times N_q^{\delta_q} \quad (6)$$

Here $\gamma_1, \ldots, \gamma_p, \delta_1, \ldots, \delta_q$ are natural numbers that depend on the view V and satisfy the following conditions, where s is the number of subgoals in V: (1) $1 \leq \sum_i \delta_i \leq s$; (2) when V is projection free then $\sum_i \delta_i = s$; and (3) $\sum_i \gamma_i \leq \lambda s$, for some fixed integer $\lambda > 0$.

Given the results in Sect. 3 for CSA, one wonders if for PSA we still need an exponential number of views to in an optimal view configuration. The answer is "no", as we shall prove in a moment, but first we notice that PSA introduces a new kind of problem: some multiplicative estimators E may lead to view selection problems that do not have optimal solutions! This is illustrated by the following example. Recall that the cost of a join between two tables of (estimated) sizes L and M is $\alpha LM + \beta(L + M) + \gamma N$.

Example 5 Consider the schema $\mathcal{R}$ with two binary relations R and S and the queries:

$$Q_1(X, Y) :- R(X, Y), S(Y, Y)$$
$$Q_2(X, Y) :- R(X, Y)$$
$$Q_3(X, Y) :- S(X, Y)$$

Assume the statistics to be $\Sigma = \{c, d, N_R, N_S, N_{S_1}\}$, where N_R, N_S, N_{S_1} are the cardinalities of R, S, and $\pi_1(S)$, respectively, c is the selectivity for $S(Y, Y)$, i.e., $E(\mathcal{R}, S(Y, Y)) = c \times N_S$, and d is explained below. Assume a space bound B such that we can store R and S, but not R, S and Q_1, or R, S and $S(Y, Y)$. In particular, $N_R + N_S < B < N_R + N_S + c \times N_S$. Hence, in the warehouse context, we are forced to choose R and S as views to materialize, and have some space left for some additional view(s) to reduce the cost of Q_1. Consider the following infinite sequence of views, for $n = 0, 1, 2, \ldots$:

$$V_n(X) :- S(X, Y_0), S(Y_0, Y_1), S(Y_1, Y_2), \ldots, S(Y_{n-1}, Y_n)$$

Suppose E estimates the sizes of the views V_n to be:

$$E(\mathcal{R}, V_n) = d^n \times N_{S_1}$$

Hence d indicates by what factor the cardinality of the semijoin is reduced by each additional S. Since $0 < d < 1$, for all values of n large enough, the estimator will conclude that there is enough space to store V_n in addition to R and S: $N_R + N_S + d^n \times N_{S_1} < B$. Moreover, consider the following rewriting of Q_1:

$$Q_1(X, Y) :- R(X, Y), V_n(Y), S(Y, Y)$$

and the following plan for it: $(R \bowtie V_n) \bowtie S$. As n goes to infinity and the estimated sizes of both V_n and $R \bowtie V_n$ converge to 0, the cost of this plan converges to $\beta(N_R + N_S)$. There is no optimal solution to the problem in this case, since the cost of *any* plan for Q is greater than $\beta(N_R + N_S)$.

This example shows that Theorem 1 from CSA fails under PSA, the reason being that the size estimation function E here is not monotone. Indeed, for every $n \geq 1$, the view $V(Y) : -S(Y, Y)$ is contained in the view V_n ($V \subseteq V_n$). However, for n large enough, the estimates are $E(\mathcal{R}, V) = c \times N_S > d^n \times N_{S_1} = E(\mathcal{R}, V_n)$. In general, as this discussion suggests, it is virtually impossible to find robust estimators that are monotone.

The assumption of large cardinalities. The lack of an optimal solution in the example above is due to the fact that E estimates the cardinalities of certain views V_n to be excessively small (even less than 1, for large n). To avoid such anomalies, we will make the following assumption: *all cardinalities are large*. Technically, this translates into two conditions, one on the function E and the other on the statistics Σ. First, we will assume that the function E always returns a cardinality that is at least L, some lower bound. That is, there exists a multiplicative estimator E' such that:

$$E(\mathcal{R}, V) = \begin{cases} \lceil E'(\mathcal{R}, V) \rceil & \text{when } E'(\mathcal{R}, V) > L \\ 0 \text{ or } L & \text{otherwise} \end{cases}$$

In the gray zone, when the multiplicative formula is less than L, we let the estimator choose whether to believe that the view is empty, and return 0, or that it is non-empty, and return L. For such estimators, we revisit the view selection problem slightly, but allowing only views with a non-zero size estimate to be included in any view configuration[4].

The second technical condition for the large cardinalities assumption is that there exist two numbers c and N such that for every set of statistics $\Sigma = \{c_1, \ldots, c_p, N_1, \ldots, N_q\}$ we have $c \leq \min(c_1, \ldots, c_p)$ and $N \leq \min(N_1, \ldots, N_q, L)$. Recall that the statistics vary with the schema $\mathcal{R}$. Our condition implies that, as the schema increases in complexity, all new cardinalities and/or factors added to Σ are larger than the fixed values N and c. The two theorems below also assume two technical conditions: $c \times N > 1$ and $c^\lambda \times N^\mu > 1$, respectively, where λ is from Def. 2 and μ will be introduced below. These conditions are justified by the same assumption on large cardinalities, and read: "if we apply one selection, or a succession of λ/μ selections, respectively, to a table of cardinality N, we still obtain an answer with at least 1 tuple".

A polynomial bound on the number of views. We first prove that, when a solution to the view selection problem exists under PSA, then there exists one with a polynomial number of views. This result should be contrasted to Theorem 3, which proves that under CSA certain problem instances require an exponential number of views.

For the remainder of this section, given a view selection problem $\mathcal{R}, \mathcal{Q}, B$, a *view configuration*, $\mathcal{V}$, means one in which every query in the workload $\mathcal{Q}$ is rewritten only in terms of the

[4] Views with estimated size 0 would, of course, artificially reduce the cost of any plan in which they participate.

views, and whose total size does not exceed B. In addition, we use a product cost model, in which the cost of the join of two relations R and S of size L and M, respectively, is $\alpha \times LM + \beta \times (L+M)$ where $\alpha > 0$. Notice that, compared to the cost formula in Sec. 2.1 we dropped the term $\gamma \times N$, where N is the size of the result: this is without loss of generality since $N \leq LM$, hence the term $\gamma \times N$ is dominated by $\alpha \times LM$, after increasing α by 1.

We start with the following Lemma.

Lemma 1 *Consider a view selection problem given by $\mathcal{R}$, $\mathcal{Q}$, and B, and let s be the largest number of subgoals in the body of any query Q_i in $\mathcal{Q}$. Assume the problem admits at least one view configuration*[5] *(not necessarily optimal). Then, there exists a view configuration $\mathcal{V}_0$, whose cost is bounded by:*

$$C(\mathcal{R}, \mathcal{V}_0, \mathcal{Q}) \leq (s-1)(\alpha B^{2s} + 2\beta B^s) \tag{7}$$

Proof. Let $\mathcal{V}$ be some view configuration for our problem. Consider every query Q_j in $\mathcal{Q}$ and its rewriting using only views in $\mathcal{V}$. Based on the results on query rewriting using views in [18] there exists a rewriting of Q_j that only uses a subset of those views, namely at most one for every subgoal in Q_j. Let $\mathcal{V}_0 \subseteq \mathcal{V}$ be the set of views used in these rewritings for all queries $Q_j \in \mathcal{Q}$. $\mathcal{V}_0$ is obviously a view configuration for our problem, since its total size does not exceed that of $\mathcal{V}$ (and, hence, does not exceed B).

In particular, each view V in $\mathcal{V}_0$ has an estimated size which is at most B, and there are at most $s-1$ joins between such views in each query Q_j. Moreover, the size of each intermediate result is estimated to be at most B^s (this can be shown using (2) in Definition 2), hence, each join costs at most $\alpha B^{2s} + 2\beta B^s$, resulting in $C(\mathcal{R}, \mathcal{V}_0, Q_j) \leq (s-1)(\alpha B^{2s} + 2\beta B^s)$. □

Theorem 8 *Assume $c \times N > 1$. Consider a view selection problem given by $\mathcal{R}$, $\mathcal{Q}$, and B, and assume $\mathcal{V}$ is its optimal view configuration, either under the left-linear plan restriction, or under bushy plans. Then the number of views in $\mathcal{V}$ is bounded by a polynomial in the size of $\mathcal{R}$, $\mathcal{Q}$, and the binary representations of B, c, and N.*

Proof. Given an optimal configuration $\mathcal{V}$, for each query $Q_i \in \mathcal{Q}$ we have a rewriting of Q_i in terms of the views in $\mathcal{V}$. Let i be such that the rewriting of Q_i uses the largest number of views (break ties arbitrarily) and let k be the number of views used in the rewriting of Q_i. Our goal is to prove that k is bounded by a polynomial, since then the total number of views in $\mathcal{V}$ is also bounded by a polynomial. By Lemma 1 there exists some configuration whose cost is at most $(s-1)(\alpha B^{2s} + 2\beta B^s)$, hence the cost of $\mathcal{V}$ is also bounded by this value. The idea is that if k is "too big", then the cost of Q_i alone will exceed this value.

Formally, consider an optimal plan for Q_i that uses $\mathcal{V}$, which, under our assumptions in Sect. 2, has the form $\pi(A \bowtie B)$. Here $A \bowtie B$ represents the last join in the plan, and $C(\mathcal{R}, \mathcal{V}, Q_i) \geq \alpha \times N_A \times N_B + \beta(N_A + N_B) \geq \alpha \times N_A$, where N_A, N_B are the estimated sizes of A and B. The plans A and B correspond to subsets of k subgoals in $body(Q_i)$ under the rewriting with $\mathcal{V}$. Suppose A has more subgoals than B, hence A has at least $k/2$ subgoals (or at least $k-1$, in the case of left-linear joins). Since A has no projections and all views in A have an estimated size at least L, the estimated size of A is:

$$\begin{aligned} N_A &\geq c_1^{\gamma_1} \times \ldots \times c_p^{\gamma_p} \times L^{\frac{k}{2}} \\ &\geq c^{\gamma_1 + \ldots + \gamma_p} \times L^{\frac{k}{2}} \\ &\geq c^{\lambda \frac{k}{2}} \times L^{\frac{k}{2}} \end{aligned}$$

and we obtain:

$$\alpha \times c^{\frac{k}{2}} \times L^{\frac{k}{2}} \leq C(\mathcal{R}, \mathcal{V}, Q_i) \leq \frac{1}{w_i} C(\mathcal{R}, \mathcal{V}, \mathcal{Q}) \tag{8}$$

Obviously:

$$C(\mathcal{R}, \mathcal{V}, \mathcal{Q}) \leq C(\mathcal{R}, \mathcal{V}_0, \mathcal{Q}) \tag{9}$$

since $\mathcal{V}$ is an optimal solution. Inequalities (8), (9), and (7), together with $c \times L \geq c \times N > 1$, give us the polynomial upper bound on k. □

Join-sensitive size estimators. Under our current assumptions for size estimators, some view selection problems may have view configurations but no optimal ones. We show here that, with some additional restrictions on the estimator function, every view selection problem admits an optimal view configuration, if it admits any at all, and that finding such a configuration places the problem in NP.

Referring to the notations in Definition 2, we call a multiplicative estimator *join-sensitive* if there exists a positive constant $\mu > 0$ such that

$$\delta_1 + \ldots + \delta_q \geq \mu \times s,$$

where s is the number of subgoals in the view V. Here the constant μ, together with λ, c, and N, are fixed for all database schemas and associated statistics.

A join-sensitive estimator will penalize us if we try to use views with too many joins, because it will estimate their size as being very large. Definition 2 already does this for projection-free views, while a multiplicative estimator extends this to arbitrary views. As we have seen in Example 5, considering estimators that are not join-sensitive may make sense in practice. Indeed, the size estimator E there is not join-sensitive, since in the estimated size of V_n a single factor is a cardinality. However, most estimators in practice are chosen to be join-sensitive both for simplicity and because they are computed bottom up. For example, for V_n, they would first consider the expression $\pi(S \bowtie S \bowtie \ldots \bowtie S)$. The estimator then computes the number of tuples in the join, yielding some expression of the form $f^{n-1} \times N_S^n$, and finally computes the number of tuples in the projection. The result has the form $gf^{n-1} \times N_S^n$, where g and f are some numbers between 0 and 1. Such an estimator is join-sensitive.

The view selection problem always has a solution when the estimator is join-sensitive. Moreover, in searching for a solution, one only needs to consider views whose size is linearly bounded by the largest number of subgoals of any query in $\mathcal{Q}$; this is a strengthening of Theorem 8, which only gives a polynomial upper bound.

[5] When B is too small, there may be no solution at all.

Theorem 9 *Assume E to be a join-sensitive estimator, and assume $c^\lambda \times N^\mu > 1$. Consider a view selection problem given by $\mathcal{R}$, $\mathcal{Q}$, and B, and assume it admits at least one view configuration. Let s be the largest number of subgoals in any query in $\mathcal{Q}$. Then the following hold, both under the restriction to left-linear join plans and under arbitrary bushy plans: (1) the view-selection problem always has an optimal solution; (2) if a view V is part of an optimal solution then the number of subgoals in V is $O(s)$; and (3) the view selection problem is in NP.*

Proof. By Lemma 1 there exists a view configuration $\mathcal{V}_0$ whose cost satisfies:

$$C_0 = C(\mathcal{R}, \mathcal{V}_0, \mathcal{Q}) \leq (s-1)(\alpha B^{2s} + 2\beta B^s) \quad (10)$$

We first show that, for any configuration $\mathcal{V}$, if it contains at least one view V that is not equivalent to any query in the workload $\mathcal{Q}$ such that the estimated size of V is larger than a certain bound (namely $\max(1, \frac{1}{\alpha})C_0$), then the cost of $\mathcal{V}$ is larger than C_0. In other words, when searching for an optimal configuration we can restrict our consideration to views that either are queries in the workload or have an estimated size smaller than $\max(1, \frac{1}{\alpha})C_0$.

Assume V is such a view, i.e., V is not equivalent to any query in the workload $\mathcal{Q}$ and has an estimated size larger than $\max(1, \frac{1}{\alpha})C_0$. Let V be used in the rewriting of some query Q_i (of course, views that are never used can simply be dropped from the configuration). V must be used either in a selection or in a join inside Q_i (since V is not equivalent to Q_i). In the first case the cost of the selection is at least $E(\mathcal{R}, V) > C_0$. In the second case, the costs of the joins make the cost of computing Q_i at least $\alpha \times E(\mathcal{R}, \mathcal{V}) \times M$ (M is the size of the relation with which V is joined). Since $M \geq 1$, we have the cost of Q_i exceeds C_0.

Thus, an optimal configuration, if it exists, consists only of views in the sets $\mathcal{Q}$ and $\mathcal{W} = \{V \mid E(\mathcal{R}, V) \leq \max(1, \frac{1}{\alpha})C_0\}$. When E is a join-sensitive estimator we can show that the latter set is finite. For that we prove something stronger: that for each view in $\mathcal{W}$, the number of subgoals in the body of the view is $O(s)$, where, recall, $s = \max_i(|\ body(Q_i)\ |)$. Let $k = |\ body(V)\ |$, for some view V in $\mathcal{W}$. Recall that $c \leq \min(c_1, \ldots, c_p)$, $N \leq \min(N_1, \ldots, N_q)$, and $c^\lambda \times N > 1$. Since E is join-sensitive, the estimated size of V is:

$$E(\mathcal{R}, V) = c_1^{\gamma_1} \times \ldots \times c_p^{\gamma_p} \times N_1^{\delta_1} \times \ldots \times N_q^{\delta_q} \geq c^{\gamma_1 + \ldots + \gamma_p} \times N^{\delta_1 + \ldots + \delta_q} \geq c^{\lambda k} \times N^{\mu k}.$$

This, together with $E(\mathcal{R}, V) \leq \max(1, \frac{1}{\alpha})C_0$ and with Eq. 10 gives us an upper bound for k, which is essentially the product of s and the number of bits needed to represent B. Then, the claims in the theorem follow immediately: (1) holds since we have reduced the search space to a finite set of views; we have shown (2) already; and, finally, (3) holds since it suffices to guess the views in the configuration. □

5 Conclusions

View selection is becoming a critical problem in several data management applications: query optimization, data-warehouse design, data placement in distributed environments, and ubiquitous computing. This paper answered several fundamental questions about the view selection problem: which views need to be considered in an optimal view configuration, what is the cardinality of an optimal view configuration, and what is the complexity of the view selection problem.

As we have shown, the answer depends critically on whether we can accurately estimate the size of views over the given database. When we have accurate size estimates, i.e., under the complete statistics assumption, we have shown that the cardinality of an optimal view configuration may be exponential in the size of the database schema and query workload. As a result, we have established an exponential-time lower bound on the view selection problem, and a double-exponential upper bound. Under the partial statistics assumption, when we use multiplicative size estimators, we have shown that the cardinality of an optimal view configuration is polynomially bounded, and, hence, the view selection problem is in NP. We have also shown that under certain conditions, the view selection problem may not have an optimal solution.

Index selection. An important issue in the formulation of the view selection problem is the effect of choosing indexes on the views in the configuration. While our results have not considered the selection of indexes, it is easy to show that the results still hold if we assume that the number of views in an optimal configuration does not change if we consider indexes.

The intuitive justification for this assumption is that indexes *cannot* increase the number of views needed for an optimal configuration. That is, if an indexed view is part of an optimal plan for a query, then there should be some database instance such that the view is useful in an optimal plan that does not use indexes. The justification for using a view (if it's not needed for correct query semantics) is that using the view reduces the overall cost of the query. There are two factors affecting the usefulness of a view. The main one is whether the size of the intermediate result is smaller after the join, and the secondary one is the cost of performing the additional join (which must be compensated for by the benefit of the result). The presence or absence of an index only affects the second factor, not the first, because it does not affect the resulting size of the join result. Hence, the presence of an index may change the cost-benefit threshold locally, but has little effect on the rest of the plan.

Updating costs. Another important practical issue is the cost of maintaining the views that have been materialized (though in some contexts it is sufficient to assume views are updated in an off-line process done periodically). In most of the cases we discussed, it suffices to model the cost of updates by assuming some of the queries in $\mathcal{Q}$ are update queries. The subtle issue, however, is that in the presence of updates, the view configuration may contain queries whose sole purpose is to speed up updates, rather than support any of the queries in $\mathcal{Q}$ [19]. In future work, we will extend our analysis to cover such cases.

Implications of our results. This paper describes mostly theoretical results concerning the view selection problem. A natural question that arises is how the results impact the design of practical algorithms. The first implication is that practitioners

should be aware that pitfalls indeed exist in this problem space. The craftiness that was required to prove the lower-bound theorems hints that in practice, exponential configurations may not be common. Our results show that the size of the configuration may become an issue mostly when there is heavy use of views with projections and when views provide different vertical partitions of a particular relation. Hence, we believe that design and analysis of efficient view-selection algorithms for applications of this nature presents an interesting area of future work. Of particular interest is the direction that would explore the design of approximation algorithms for view selection and would analyze how close to optimal are small sets of views. Another direction of future research is to discover more special cases where optimality is attainable with a reasonable number of views. (The paper mentions one such case, views defined without using projections.)

6 Appendix: Proof of Optimality of the View Configuration

We now prove that the view configuration $\mathcal{V}$ that we described in Sect. 3.2.1 is the optimal on the database instance $\mathcal{D}$. We recall that we only consider left-linear plans, and the product cost-model ([4] generalizes this result to the sum cost-model).

More precisely, we show that there exists an integer n_0, such that for all $n \geq n_0$, a plan for the query q_3 that uses all the views in $\mathcal{V}$ is optimal on the database instance $\mathcal{D}$ under the left-linear plan constraint. Later, we generalize the result to the case in which the plan may be bushy, as long as either S or T is the last term.

We recall that our query rewriting q_3' of the query q_3 is

$$\begin{aligned} q_3'(X_1, \ldots, X_n) :- \; & s(X_1, \ldots, X_n), v_0(X_1), \\ & v_1(X_{11}, \ldots, X_{1n/2}), \\ & v_2(X_{21}, \ldots, X_{2n/2}), \ldots, \\ & v_N(X_{N1}, \ldots, X_{Nn/2}), \\ & t(X_1, \ldots, X_n), \end{aligned}$$

and the query plan $\mathcal{P}^*$ that we consider is:

$\mathcal{P}^* = S \bowtie V_0 \bowtie V_1 \bowtie \ \ldots \ \bowtie V_N \bowtie T.$

We need to show that under the product cost model assumption and on the database instance $\mathcal{D}$, the cost of our preferred plan $\mathcal{P}^*$ cannot exceed the cost of any other left-linear plan for the query q_3.

We first note that if a rewriting q_3' of q_3 does not mention both S and T, then it cannot be equivalent to q_3. Hence, we can consider only rewriting (and hence plans) that refer to both tables.

To show that $\mathcal{P}^*$ is an optimal left-linear plan for the query q_3 on the database instance $\mathcal{D}$, we need to cover several cases. We proceed as follows, considering first plans in which S precedes T:

- We first show that $\mathcal{P}^*$ is cheaper than a direct intersection of the tables S and T (Sect. 6.2).
- Then, we show that if we permute the views $v_0, \ldots, v_N$ in $\mathcal{P}^*$, or even remove some of the views from the plan, the resulting plan cannot be cheaper than the plan $\mathcal{P}^*$ (Sect. 6.3).
- Next, we consider all other types of views that can be defined and are not in the workload $\mathcal{V}$ (Sect. 6.4). For each one, we show one of the following:
  - The view cannot be stored in the available space B,
  - The view is useless in some way in the computation of q_3, or
  - The resulting plans are clearly more expensive than $\mathcal{P}^*$.
- Finally, in a symmetric fashion, we can repeat the same arguments for the case in which T precedes S, and show that no such plan can be cheaper than $\mathcal{P}^*$.

We begin by estimating the cost of $\mathcal{P}^*$.

6.1 The cost of the preferred plan $\mathcal{P}^*$ on the database instance $\mathcal{D}$

Our proof will use the fact that our preferred plan $\mathcal{P}^*$ for the query q_3 can be represented as a join of a sub-plan $\mathcal{P}_1^*$ with the stored table T,

$\mathcal{P}^* = \mathcal{P}_1^* \bowtie T,$

where

$\mathcal{P}_1^* = S \bowtie V_0 \bowtie V_1 \bowtie \ \ldots \ \bowtie V_N.$

Under the product cost model, the cost of the sub-plan $\mathcal{P}_1^*$ of $\mathcal{P}^*$ is

$$\begin{aligned} |\mathcal{P}_1^*| = {} & |S| \times |V_0| + |S \bowtie V_0| \times |V_1| + |S \bowtie V_0 \bowtie V_1| \times |V_2| \\ & + |S \bowtie V_0 \bowtie V_1 \bowtie V_2| \times |V_3| + \ \ldots \\ & + |S \bowtie V_0 \bowtie V_1 \bowtie \ \ldots \ \bowtie V_{N-1}| \times |V_N|. \end{aligned} \tag{11}$$

Intuitively, when the plan $\mathcal{P}_1^*$ is computed on the database instance $\mathcal{D}$, each view v_i strictly reduces the size of the intermediate result $S \bowtie V_0 \bowtie \ \ldots \ \bowtie V_i$, until only the tuples from $S \bigcap T$ remain in the relation $S \bowtie V_0 \bowtie \ \ldots \ \bowtie V_N$. More precisely, the view v_0 removes all intimidator tuples from S, and each view v_i, $i > 0$, removes exactly one stubborn tuple from S. After computing the upper bounds on the sizes of the views and intermediate relations in $\mathcal{P}_1^*$ on the database instance $\mathcal{D}$, we obtain the following upper bound on the cost of the sub-plan $\mathcal{P}_1^*$:

$$|\mathcal{P}_1^*| \leq |S| \times |V_0| + M, \tag{12}$$

where M is the number of intimidator tuples in S. As the cost of the preferred plan $\mathcal{P}^*$ is

$$|\mathcal{P}^*| = |\mathcal{P}_1^*| + |S \bowtie V_0 \bowtie \ \ldots \ \bowtie V_N| \times |T|, \tag{13}$$

we obtain the following upper bound on the cost of the preferred plan $\mathcal{P}^*$:

$$|\mathcal{P}^*| \leq |S| \times |V_0| + M + |S \bigcap T| \times |T|. \tag{14}$$

6.2 The preferred plan $\mathcal{P}^*$ is cheaper than the plan without views

Consider the original plan for q_3, that is, the direct intersection of the tables S and T. In estimating the cost of this plan on $\mathcal{D}$, we will be using the following facts that are all true by

construction of the tables S and T in the database instance $\mathcal{D}$. Each of S and T includes at least M tuples (the intimidator tuples I_S and I_T, respectively); S and T are of equal sizes; the table $S \bigcap T$ (and consequently, any view that is a projection on $S \bigcap T$) has less than M tuples; in each of S and T, there are N stubborn tuples ($N = \binom{n}{n/2}$). From these facts, it follows that on $\mathcal{D}$, the cost of the direct intersection of S and T is

$$\begin{aligned} |S| \times |T| &= (|I_S| + N + |S \bigcap T|) \times |T| = |I_S| \\ &\times |S| + N \times |T| + |S \bigcap T| \times |T| \qquad (15) \\ &\geq |V_0| \times |S| + N \times M + |S \bigcap T| \times |T| \\ &> |S| \times |V_0| + M + |S \bigcap T| \times |T|. \end{aligned}$$

From inequality 14, it is clear that the direct intersection of S and T is more expensive than the preferred $\mathcal{P}^*$ on the database instance $\mathcal{D}$.

6.3 Plans where all views are in $\mathcal{V}$

We now focus on all plans for q_3 where S is the first term, T is the last term, and that use only the views in the view configuration $\mathcal{V}$.

First consider any plan $\mathcal{P}^{(ij)}$ that is obtained from the preferred plan $\mathcal{P}^*$ by swapping two views v_i and v_j, $0 < i < j \leq N$. By construction of the table S and of the views, the view v_i removes from S exactly one tuple – a stubborn tuple s_i that no other view in $\mathcal{V}$ can remove from S. Similarly, v_j removes from S some other stubborn tuple s_j and nothing else. We conclude that the intermediate relation $S \bowtie V_0 \bowtie \ ... \ \bowtie V_j \bowtie \ ... \ \bowtie V_i$ in the plan $\mathcal{P}^{(ij)}$ is exactly the same as the intermediate relation $S \bowtie V_0 \bowtie \ ... \ \bowtie V_i \bowtie \ ... \ \bowtie V_j$ in the plan $\mathcal{P}^*$. We observe that the costs of computing these two relations, as well as the remaining parts of the two plans, are the same. (By construction of the table $S \bigcap T$ in the database instance $\mathcal{D}$, all tables V_i, $i > 0$, are of the same size.) We conclude that the cost of the plan $\mathcal{P}^{(ij)}$ on the database instance $\mathcal{D}$ is the same as the cost of the preferred plan $\mathcal{P}^*$.

It is straightforward to generalize this case to the case when the views $v_1, \ldots, v_N$ in the plan $\mathcal{P}^*$ are permuted in an arbitrary way. We conclude that each resulting plan has the same cost, on the database instance $\mathcal{D}$, as the preferred plan $\mathcal{P}^*$.

Now consider a plan $\mathcal{P}^{(01)}$ that is the same as the preferred plan $\mathcal{P}^*$, except that the views v_0 and v_1 are swapped. Recall that in the database instance $\mathcal{D}$, the domain of the table $S \bigcap T$ is disjoint from the domain of the intimidator tuples of S. It follows that the view v_1 removes all the intimidator tuples from S, just as v_0 does. (In addition, v_1 removes one stubborn tuple from S.) It follows that the table $S \bowtie V_1$ does not have any intimidator tuples. Therefore, joining $S \bowtie V_1$ with V_0 does not remove any further tuples from S.

We conclude that the intermediate relation $S \bowtie V_1 \bowtie V_0$ in the plan $\mathcal{P}^{(01)}$ is the same as the relation $S \bowtie V_1$. Therefore, in the sequence $S \bowtie V_1 \bowtie V_0$ in the plan $\mathcal{P}^{(01)}$, V_0 is useless and can be removed.

Having shown that the view v_0 is useless in the plan $\mathcal{P}^{(01)}$ and can be removed, we proceed to compare the cost of computing the intermediate relation $S \bowtie V_1$, in the modified plan $\mathcal{P}^{(01)}$, to the cost of computing the intermediate relation $S \bowtie V_0 \bowtie V_1$ in the plan $\mathcal{P}^*$. Recall that the table V_0 is a projection of the table $S \bigcap T$ on just one attribute, while the table V_1 is a projection of $S \bigcap T$ on $n/2$ attributes. By construction of the fat tuples in the intersection of S and T, the size of the table V_1 grows exponentially (as n increases) in the size of the table V_0. It follows that the cost of computing the table $S \bowtie V_1$, in the plan $\mathcal{P}^{(01)}$, also grows exponentially faster than the cost of computing the table $S \bowtie V_0 \bowtie V_1$ in the plan $\mathcal{P}^*$.

After observing that the remaining parts ($\bowtie V_2 \bowtie \ ... \ \bowtie V_N \bowtie T$) in the plans $\mathcal{P}^{(01)}$ and $\mathcal{P}^*$ are the same, we conclude that there exists an integer value n_1, such that for all $n > n_1$, the cost of the plan $\mathcal{P}^{(01)}$ on the database instance $\mathcal{D}$ is greater than the cost of the preferred plan $\mathcal{P}^*$.

Now consider any plan $\mathcal{P}^{(0i)}$ that is obtained from the preferred plan $\mathcal{P}^*$ by swapping the view v_0 with some view v_i, $i > 1$. This plan can be obtained from the preferred plan $\mathcal{P}^*$ in two steps. First, the views v_1 and v_i are swapped; we know that the resulting plan $\mathcal{P}^{(1i)}$ has the same cost as $\mathcal{P}^*$. Second, the view v_0 in the plan $\mathcal{P}^{(1i)}$ is swapped with the view v_i to obtain the plan $\mathcal{P}^{(0i)}$, which is similar to the case above when the views v_0 and v_1 are swapped in $\mathcal{P}^*$ to obtain the plan $\mathcal{P}^{(01)}$. We conclude that the plan $\mathcal{P}^{(0i)}$ is asymptotically more expensive on the database instance $\mathcal{D}$ than the plan $\mathcal{P}^*$.

Now we consider plans that result from removing one or more views from the preferred plan $\mathcal{P}^*$. We have already seen above that swapping the view v_0 with any other view in $\mathcal{P}^*$ is equivalent to removing the view v_0 from the plan $\mathcal{P}^*$ altogether and results in a more expensive plan. Consider the case when the view v_N is removed from the plan $\mathcal{P}^*$, to obtain the plan $\mathcal{P}^{(-N)}$. We know that the view v_N removes exactly one stubborn tuple s_N from the table S, and no other view in $\mathcal{V}$ can remove the tuple s_N from the table S. We conclude that the stubborn tuple s_N remains in the intermediate relation $S^{(-N)} = S \bowtie V_0 \bowtie \ ... \ \bowtie V_{N-1}$.

Consider the cost of the plan $\mathcal{P}^{(-N)}$; it is the sum of the cost of computing the table $S^{(-N)}$ and of the cost of joining the table $S^{(-N)}$ with the table T. The computation of the table $S^{(-N)}$ involves computing the join of the table S with the view v_0, and therefore the cost of computing $S^{(-N)}$ can be bound from below by the expression $|S| \times |V_0|$. At the same time, the table $S^{(-N)}$ has one more tuple (the stubborn tuple S) than the table $S \bowtie V_0 \bowtie \ ... \ \bowtie V_N = S \bigcap T$. Therefore, the cost of computing the join of the tables $S^{(-N)}$ and T is

$$\begin{aligned} |S^{(-N)}| \times |T| &= (|S \bigcap T| + 1) \times |T| \\ &> |S \bigcap T| \times |T| + M \end{aligned}$$

(the table T has at least M tuples).

Therefore, the cost of computing the plan $\mathcal{P}^{(-N)}$ can be bound from below by

$$(|S| \times |V_0|) + (|S \bigcap T| \times |T| + M).$$

The comparison of this lower bound on the cost of the plan $\mathcal{P}^{(-N)}$ with the upper bound on the cost of the preferred plan $\mathcal{P}^*$ in inequality 14 allows us to conclude that the plan $\mathcal{P}^{(-N)}$

is more expensive on the database instance $\mathcal{D}$ than the preferred plan $\mathcal{P}^*$.

The case when a view $v_i, 1 \leq i < N$, is removed from the plan $\mathcal{P}^*$, is treated similarly to the case above after we observe that we can first swap the views v_i and v_N to obtain a plan with the same cost as the cost of the preferred plan $\mathcal{P}^*$, and then remove the last view v_i from the resulting plan, to obtain a plan whose cost is the same as that of $\mathcal{P}^{(-N)}$.

In the case when at least two views (besides v_0) are removed from the preferred plan $\mathcal{P}^*$, we estimate the cost of the resulting plan by starting with $\mathcal{P}^*$ and removing one view at a time. Observe that at each step, one more stubborn tuple is added to the intermediate relation R that is joined with the table T. Therefore, the cost of joining that intermediate relation R with T grows at each step by at least $1 \times M$ (1 tuple is added to the intermediate relation R, and T has at least M tuples). Therefore, at each step the output plan is strictly more expensive on the database instance $\mathcal{D}$ than the input plan (recall that the cost of joining the relation $S \bowtie V_0$ with all other views in $\mathcal{V}$ does not exceed M). We conclude that the plan output by the whole process is strictly more expensive than the preferred plan $\mathcal{P}^*$.

Finally, we consider left-linear plans where two or more views in $\mathcal{V}$ occur before S. Suppose that the views $v_0, \ldots, v_m$, $m > 0$, occur before S in the plan $\mathcal{P}^m$.

Consider the intermediate relation $V_0 \bowtie \ \ldots \ \bowtie V_m$ in the plan $\mathcal{P}^m$. There are three cases, based on the number of attributes in this relation.

First, the relation $V_0 \bowtie \ \ldots \ \bowtie V_m$ can have fewer than $n/2$ attributes. As we will see from Sect. 6.4, views with fewer than $n/2$ attributes do not remove any stubborn tuples from tables S or T. Therefore, a join of the relation $V_0 \bowtie \ \ldots \ \bowtie V_m$ with the table S can only remove the intimidator tuples from S. We transform the plan $\mathcal{P}^m$ by removing the views $v_1, \ldots, v_m$. It is easy to see that the resulting plan (i) is cheaper than the plan $\mathcal{P}^m$ and (ii) is equivalent to the plan $S \bowtie V_0 \bowtie V_{m+1} \bowtie \ \ldots \ \bowtie V_N \bowtie T$. As we saw above, this plan is more expensive on the database instance $\mathcal{D}$ than the preferred plan $\mathcal{P}^*$. We conclude that $\mathcal{P}^m$ is also more expensive than $\mathcal{P}^*$.

The second case is when the intermediate relation $V_0 \bowtie \ \ldots \ \bowtie V_m$ in the plan $\mathcal{P}^m$ has exactly $n/2$ attributes. In this case, we can replace $V_0 \bowtie \ \ldots \ \bowtie V_m$ by one of the views $v_i \, \epsilon \, \mathcal{V}$, $i > 0$, that has exactly the same attributes as $V_0 \bowtie \ \ldots \ \bowtie V_m$. This will transform the plan $\mathcal{P}^m$ into a cheaper left-linear plan where exactly one view (v_i) precedes S. It is easy to see that the resulting plan is equivalent to a left- linear plan where S is the first term and that only includes views from $\mathcal{V}$. We have shown above that any such plan is at least as expensive on the database instance $\mathcal{D}$ as the preferred plan $\mathcal{P}^*$.

The remaining case is when the intermediate relation $V_0 \bowtie \ \ldots \ \bowtie V_m$ in the plan $\mathcal{P}^m$ has at least $n/2 + 1$ attributes. Then the relation $V_0 \bowtie \ \ldots \ \bowtie V_m$ must contain a projection W of the table $S \bigcap T$ on at least $n/2 + 1$ attributes. Therefore, the table W contains a projection of all the fat tuples on at least $n/2 + 1$ attributes. By construction of the fat tuples, the size of the table W, on the database instance $\mathcal{D}$, is at least $F \times |V_i|$, $i > 0$, where F is the size of the domain of the fat tuples, and v_i is a view in the view configuration $\mathcal{V}$. As the sizes of all projections of S on $n/2$ attributes are the same on the database instance $\mathcal{D}$, we can choose V_i to be V_1. We conclude that the size of the intermediate relation $V_0 \bowtie \ \ldots \ \bowtie V_m$ in the plan $\mathcal{P}^m$ is also at least $F \times |V_1|$:

$$|V_0 \bowtie \ \ldots \ \bowtie V_m| \geq |W| \geq F \times |V_1|.$$

Recall that in the plan $\mathcal{P}^m$, the intermediate relation $V_0 \bowtie \ \ldots \ \bowtie V_m$ is joined with the table S. The cost of $\mathcal{P}^m$ is therefore at least the sum of the cost of joining $V_0 \bowtie \ \ldots \ \bowtie V_m$ with the table S and of the cost of joining the resulting relation $V_0 \bowtie \ \ldots \ \bowtie V_m \bowtie S$ with the table T:

$$|\mathcal{P}^m| \geq |V_0 \bowtie \ldots \bowtie V_m| \times |S| + |V_0 \bowtie \ldots \bowtie V_m \bowtie S| \times |T|.$$

Notice that the size of the relation $V_0 \bowtie \ \ldots \ \bowtie V_m \bowtie S$ is at least the size of $S \bigcap T$, otherwise the rewriting is not equivalent to q_3. Finally, we observe that on the database instance $\mathcal{D}$, the table V_1 has at least one more tuple than the table V_0.

Taking together all the estimates, we obtain the following lower bound on the cost of the plan $\mathcal{P}^m$:

$$|\mathcal{P}^m| \geq F \times (|V_0| + 1) \times |S| + |S \bigcap T| \times |T| > |V_0| \times |S| + 1 \times M + |S \bigcap T| \times |T|$$

(the size of S is at least M). From inequality 14, we conclude that the plan $\mathcal{P}^m$ is more expensive on the database instance $\mathcal{D}$ than the preferred plan $\mathcal{P}^*$.

It is easy to see that all plans that result from permuting the views in the plan $\mathcal{P}^m$, are at least as expensive on the database instance $\mathcal{D}$ as the plan $\mathcal{P}^m$ itself. We conclude that all such plans are more expensive on the database instance $\mathcal{D}$ than the preferred plan $\mathcal{P}^*$.

6.4 Left-linear plans that use other types of views

Consider a conjunctive view w that can be defined on the schema $\mathcal{R}$ of the database instance $\mathcal{D}$ and such that w is not in the view configuration $\mathcal{V}$. We now consider all left-linear plans for the query q_3 where S precedes T and that include at least one such view w.

We show that each such plan falls into one of three categories. In the first category are plans that do not compute the query q_3 correctly. In the second category, the view w is useless in the sense that if w is removed from a plan $\mathcal{P}$, or replaced by a view from $\mathcal{V}$, the resulting plan correctly computes the query q_3 and is cheaper on the database instance $\mathcal{D}$ than the plan $\mathcal{P}$. In the third category are all plans where the view w cannot fit into the storage space B. Hence, we conclude that for any conjunctive view w that is not in the view configuration $\mathcal{V}$, the view w cannot be used to compute the query q_3 in our problem setting.

Views with selections. We first consider all conjunctive views that are defined using equality selection (we ignore trivial selections of the form $X_i = X_i$). Using any such view w in a rewriting q_3'' of the query q_3 is equivalent to applying the selection to the answer to the query q_3. (Recall that we only consider conjunctive rewritings of the query q_3.) Suppose the table Q_3'' is the result of computing the rewriting q_3'' on the database instance $\mathcal{D}$. Because of the selection, the table Q_3'' should either have two equal columns, or should have a column with only one value. Recall that the "real" answer to the

query q_3 is the intersection of the tables S and T. By construction of the database instance $\mathcal{D}$, the table $S \bigcap T$ does not have equal columns or columns with only one value. We conclude that the rewriting q_3'' is not equivalent to the query q_3 and thus need not be considered.

Views with projections on less than $n/2$ attributes. Consider a view that is a projection of $S \bigcap T$ on fewer than $n/2$ attributes. By construction of the eliminator tuples in $S \bigcap T$ in the database instance $\mathcal{D}$, any such view w includes projections of all stubborn tuples, both of S and T. Therefore, any such view w cannot remove stubborn tuples from the table S. We conclude that no such view besides v_0 (or besides any single projection of $S \bigcap T$ on one attribute) can be useful in a plan for computing the query q_3.

Views with projections on more than $n/2$ attributes. Now consider a conjunctive view w that is a projection of $S \bigcap T$ on more than $n/2$ attributes. Take any view $v_i \in \mathcal{V}$, say v_1 (all such views are of same size on the database instance $\mathcal{D}$). By construction of fat tuples in $S \bigcap T$, the size of the table W for the view w is at least $F \times |V_1|$ on the database instance $\mathcal{D}$; here, F is the size of the domain of the fat tuples.

The storage space constraint B has been chosen in such a way that it can store all the tables (on the database $\mathcal{D}$) for the views in the configuration $\mathcal{V}$ and nothing else. Therefore, B is proportional to $N \times |V_1|$ (there are N views $v_1, \ldots, v_N$ in $\mathcal{V}$, where $N = \binom{n}{n/2}$).

Notice that the value F is greater than N. We conclude that if a view w is defined as a projection of $S \bigcap T$ on more than $n/2$ attributes, the table for w cannot satisfy the storage space constraint B as soon as the value of the parameter n is greater than some value n_2. Therefore, for all $n \geq n_2$, any view configuration that includes w does not satisfy B.

Views on a single table. Finally, consider a view w that is defined using the table S only (or the table T only). We replace S by $S \bigcap T$ in the definition of w, to obtain a view w'. The view w' is contained in w, therefore the size of the table for w' does not exceed the size of the table for w on the database instance $\mathcal{D}$. Consider a rewriting $\tilde{q}_3$ of the query q_3 that uses the view w. We transform the rewriting $\tilde{q}_3$ by replacing the view w with the view w'. The cost of computing the resulting rewriting $\tilde{q}_3'$ (using a left-linear plan) cannot be greater on the database $\mathcal{D}$ than the cost of computing $\tilde{q}_3$ itself. At the same time, the rewriting $\tilde{q}_3'$ falls into one of the categories that we have already considered. From the preceding sections, the cost of an optimal left-linear plan for the rewriting $\tilde{q}_3'$ cannot exceed the cost of the preferred plan $\mathcal{P}^*$ on the database instance $\mathcal{D}$. We conclude that the cost of computing the rewriting $\tilde{q}_3$ cannot exceed the cost of the preferred plan $\mathcal{P}^*$ on the database instance $\mathcal{D}$.

6.5 Bushy plans where T (or S) is the last term

Here we consider plans that may be bushy as long as T (or S) is the last term in the plan. Let $\mathcal{P}$ be such a plan. We want to show that the plan $\mathcal{P}$ is never cheaper on the database $\mathcal{D}$ than the preferred plan $\mathcal{P}^*$. We have already established that if $\mathcal{P}$ includes any other views besides the views in $\mathcal{V}$ then $\mathcal{P}$ is more expensive on the database $\mathcal{D}$ than $\mathcal{P}^*$. Therefore, all that remains to be done is to show that for any set of views $\mathcal{V}' \subseteq \mathcal{V}$ and for any (bushy) plan $\mathcal{P}$ that uses only the views in the set of views $\mathcal{V}'$, $\mathcal{P}$ is not cheaper on the database $\mathcal{D}$ than the preferred plan $\mathcal{P}^*$.

Suppose the plan $\mathcal{P}$ is not equivalent to a left-linear plan (otherwise, we have already shown that $\mathcal{P}$ is not cheaper on the database $\mathcal{D}$ than the preferred plan $\mathcal{P}^*$). The main idea in the proof is that the plan $\mathcal{P}$ is bound to have a sub-plan that joins the tables for at least two views v_i and v_j, $0 < i < j$, from the set of views $\mathcal{V}$. In this case, the result V_{ij} of the sub-plan contains the table $V_i \bowtie V_j$, which has at least $1 + n/2$ attributes. The plan $\mathcal{P}$ joins the table V_{ij} with either the table $S \bigcap T$ or the table T. In each case, using the results we have obtained for left-linear plans, we can show that the plan $\mathcal{P}$ is more expensive on the database $\mathcal{D}$ than the preferred plan $\mathcal{P}^*$.

Acknowledgements. We would like to thank Surajit Chaudhuri, Michael Genesereth, Zack Ives, Henry Kautz, and Rachel Pottinger for very stimulating discussions regarding this work.

References

1. Agrawal S, Chaudhuri S, Narasayya V (2000) Automated selection of materialized views and indexes in Microsoft SQL Server. In: Proc VLDB, pp 496–505, Cairo, Egypt
2. Baralis E, Paraboschi S, Teniente E (1997) Materialized views selection in a multidimensional database. In: Proc VLDB, pp 156–165
3. Bello R, Dias K, Downing A, Feenan J, Finnerty J, Norcott W, Sun H, Witkowski A, Ziauddin M (1998) Materialized views in Oracle. In: Proc VLDB, pp 659–664
4. Chirkova R (2001) The view selection problem has an exponential-time lower bound for conjunctive queries and views. In: Proc PODS
5. Chirkova R, Genesereth M (2000) Linearly bounded reformulations of conjunctive databases. In: Proc DOOD, pp 987–1001
6. Chirkova R, Halevy A, Suciu D (2001) A formal perspective on the view selection problem. In: Proc VLDB
7. Goldstein J, Larson P-A (2001) Optimizing queries using materialized views: a practical, scalable solution. In: Proc SIGMOD, pp 331–342
8. Gribble S, Halevy A, Ives Z, Rodrig M, Suciu D (2001) What can databases do for peer-to-peer? In: ACM SIGMOD WebDB Workshop
9. Gupta H (1997) Selection of views to materialize in a data warehouse. In: Proc ICDT, pp 98–112
10. Gupta H, Harinarayan V, Rajaraman A, Ullman JD (1997) Index selection for OLAP. In: Proc ICDE, pp 208–219
11. Gupta H, Mumick IS (1999) Selection of views to materialize under a maintenance cost constraint. In: Proc ICDT, pp 453–470
12. Halevy AY (2000) Theory of answering queries using views. SIGMOD Record
13. Halevy AY (2001) Answering queries using views: a survey. VLDB J 10(4)
14. Harinarayan V, Rajaraman A, Ullman JD (1996) Implementing data cubes efficiently. In: Proc SIGMOD, pp 205–216
15. Karloff HJ, Mihail M (1999) On the complexity of the view-selection problem. In: Proc PODS, pp 167–173, Philadelphia, Penn., USA

16. Kossmann D (2000) The state of the art in distributed query processing. Submitted for publication.
17. Lee M, Hammer J (1999) Speeding up warehouse physical design using a randomized algorithm. In: Proc. Int. Workshop on Design and Management of Data Warehouses (DMDW-99)
18. Levy AY, Mendelzon AO, Sagiv Y, Srivastava D (1995) Answering queries using views. In: Proc PODS, pp 95–104, San Jose, Calif., USA
19. Ross KA, Srivastava D, Sudarshan S (1996) Materialized view maintenance and integrity constraint checking: trading space for time. In: Proc SIGMOD, pp 447–458
20. Theodoratos S, Sellis T (1997) Data warehouse configuration. In: Proc VLDB, pp 126–135, Athens, Greece
21. Yang J, Karlapalem K, Li Q (1997) Algorithms for materialized view design in data warehousing environment. In: Proc VLDB, pp 136–145, Athens, Greece
22. Zaharioudakis M, Cochrane R, Lapis G, Pirahesh H, Urata M (2000) Answering complex SQL queries using automatic summary tables. In: Proc SIGMOD, pp 105–116
23. Zhang C, Yang J (1999) Genetic algorithm for materialized view selection in data warehouse environments. In: Proc. Int. Conf. on Data Warehousing and Knowledge Discovery (DaWak-99)