Two-Stage Stochastic View Selection For Data Analysis

Rong Huang[1], Rada Chirkova[2], and Yahya Fathi[1]

[1] Operations Research Program, NC State University, Raleigh, NC 27695, {rhuang,fathi}@ncsu.edu
[2] Computer Science Department, NC State University, Raleigh, NC 27695, chirkova@csc.ncsu.edu

Abstract. Materialized views are used in many data-intensive systems to accelerate the processing of complex data-analysis queries. In certain settings, such as in presence of seasonal cycles in commerce, future (next-season) queries may be known only in a probabilistic sense. In such settings, it is possible to design materialized views that improve the processing efficiency not only of those analysis queries that are posed on the system in the current season, but also of the queries expected in the future. Designing views in this way would amortize the time-consuming materialization of beneficial views over the two stages, while improving the processing costs of data-analysis queries in each stage.
We call the problem of designing views as above *the two-stage stochastic view-selection problem.* In this paper we present a formal study of this problem, and propose a stochastic-programming approach to solving the problem. Our proposed approach allows for obtaining optimal solutions for many realistic-size problem instances. Further, the study of the properies of the problem that we present in this paper provides a formal basis for developing effective and efficient view-selection heuristics for larger problem instances that cannot be solved using exact methods.

1 Introduction

Many data-intensive systems, such as commercial or scientific data warehouses, store vast collections of data, whose scale tends to grow massively over time. Answering typical data-analysis queries in such systems may involve heavy use of summarization of large volumes of stored data [10,11]. As a result, brute-force evaluation of such queries tends to be complex and time consuming. One way to reduce the evaluation costs of data-analysis queries in relational data-intensive systems is to precompute and store extra relations, called *materialized views.* Intuitively, a materialized view would improve the efficiency of evaluating a query when the view relation represents the result of (perhaps time-consuming) precomputation of some "subexpression" of the query of interest; please see [24] and references therein. As such, materialized views with grouping and aggregation

may be especially attractive for evaluating data-analysis queries, because the relations for such views store in compact form the results of (typically expensive) preprocessing of large amounts of data.

Consider an illustration. Large retailer companies, such as Sears in the USA, maintain significant-size databases storing information about ongoing point-of-sale transactions (`Pos`). For accounting, reporting, and business-intelligence purposes, Sears' database system undergoes periodic runs of data-analysis queries on the stored information, including the queries for automatically or manually generated daily, weekly, or monthly summary reports. For instance, the following SQL query `Q1` would ask for the total sales per item category per customer type in September 2011.[3]

```
Q1: SELECT itemCategory, custType, SUM (amount)
    FROM Pos WHERE month = 'September' AND year = 2011
    GROUP BY itemCategory, custType;
```

Suppose that the `Pos` relation stores information about very large numbers of transactions per month. In this case, the costs of evaluating the query `Q1` in the Sears system could be reduced significantly by using in the evaluation (see `Q1'`) a materialized view `V1`, which stores total sales (`V1amt`) per item category per customer type per month per year.

```
V1: SELECT itemCategory, custType, SUM (amount) AS V1amt
    FROM Pos GROUP BY itemCategory, custType, month, year;

Q1': SELECT itemCategory, custType, SUM (V1amt)
     FROM V1 WHERE month = 'September' AND year = 2011
     GROUP BY itemCategory, custType;
```

To maximize query-processing efficiency in the data-analysis setting, all views that are "beneficial" in the above sense, would be materialized. However, the amount of storage space and computational constraints may place a limit on the beneficial views that can be materialized. The problem of choosing the definitions of beneficial views under such constraints and for a particular collection of high-priority queries is known as the *View-Selection Problem*. In recent years, a number of researchers have addressed the subject and developed exact and inexact methods for solving the problem in the *one-stage deterministic environment,* that is, where all queries are assumed to be known and given in advance, and in the *one-stage probabilistic environment,* that is, where we have a probability (frequency) of occurrence associated with each query in a given set of queries. For some of the projects and for recent overviews, please see [2, 12, 13, 16, 26, 27]. In both environments, the problem is addressed in the context of one fixed *query workload,* and we refer to it as the *One-Stage View-Selection Problem*. A separate

[3] To greatly simplify this example, we assume that the data in the Sears database are stored in a single relation `Pos`, with the attributes as named in the query `Q1`. The approach that we propose in this paper is applicable to the practical setting where one or more stored relations form the *star schema* [10, 11].

line of research has studied *dynamic view selection,* where views are selected continuously, to respond to the changes in the query workload over time [8, 9, 22]. Significant work has also been done on *index* selection, both on its own and alongside *view* selection, please see [1, 2, 8, 9, 13].

Other settings than the above scenarios could also benefit greatly from the use of materialized views in the processing of data-analysis queries. Specifically, in the real-life setting that was introduced in [14], knowledge of the *future* query workload(s) may be available, which allows us to choose and materialize appropriate views as part of advance preparation for efficient processing of the future queries. To illustrate this environment, let us revisit the Sears example described above. While some data-analysis queries would be posed on the Sears' sales data throughout the year, some of the queries may be run only at certain times of the year, resulting in seasonal periods of high use of the system (such as the yearly Black Friday sales event in the USA). In such a situation, aside from designing and using materialized views to improve the processing performance of the routine analysis queries occurring throughout the year, it would be beneficial to materialize additional views in advance, to reduce the execution time of such seasonal analysis queries as well.

A further complicating factor in this context is the fact that sometimes it is difficult to predict the specific query workload that would become relevant and prominent in a future season *(stage).* For instance, political, economic, or environmental factors could all have an impact on the nature of the queries that would become relevant in the fixed future stage. This could lead the experts to forecast two or more possible query workloads for that future stage, with a probability associated with each set. This, in turn, would lead to the need to carry out the analysis in a probabilistic environment, where we account for various *scenarios* (i.e., query workloads) that may occur in the future.

In this paper we study this problem for a relational data-intensive system in a *two-stage probabilistic environment,* with (current) *stage 1* and (fixed-season future) *stage 2,* and propose appropriate models and algorithms for solving the problem. Our *Two-Stage Stochastic View-Selection Problem* accepts as inputs: (i) a query workload that is of importance for stage 1; (ii) a collection of two or more query workloads that have been predicted for stage 2, each workload with an associated probability of it happening at stage 2;[4] (iii) the overall storage limit b_1 for the views to be materialized; and (iv) a stage-2 limit b_2, which is a fraction of b_1, such that the views to be materialized for stage 2 must collectively satisfy the storage limit b_2. The output has (1) the set of views to be materialized for the stage-1 queries, and (2) for each stage-2 query workload, the set of views to be materialized at stage 2 (including those views that will remain around from stage 1) for that query workload. The stage-1 views are all materialized at stage 1, and once (at stage 2) one of the stage-2 query workloads becomes reality, the corresponding stage-2 views are materialized. *At stage 2, while the time-consuming materialization of the stage-2 views is taking place, those stage-1 materialized views that remain in the system at stage 2 remain around to improve*

[4] The probabilities of the stage-2 query workloads sum up to 1.

the efficiency of evaluation of stage-2 queries. (Recall that the storage limit b_2 is a fraction of the overall limit b_1.) The reason is, the objective function for our solution is to minimize the overall (i.e., for stages 1 and 2) query-response time subject to the storage limits. Not surprisingly, our *Two-Stage Stochastic View-Selection Problem* is NP-hard.

As an illustration, suppose that in the Sears example, the query `Q1` (see above) constitutes the stage-1 query workload. We are also given two stage-2 queries: (1) Query `Q2` asks for the total sales per item category per customer type per store in December 2011, and (2) Query `Q3` asks for the total sales per item category in December 2011. Suppose that either `Q2` or `Q3` will occur at stage 2 with probability 0.5. Then, at stage 1, instead of materializing the view `V1` (see above), we can materialize a view `V2` that has the total sales per item category per customer type per store per month per year, and use it to answer `Q1`. At stage 2, if `Q2` occurs, we use `V2` to answer it. Otherwise, if `Q3` occurs, we answer it by materializing *at stage 2* a view `V3` that stores the total sales per item category per month per year. The larger-size relation for `V2` can be used to evaluate the queries in *both* stages, assuming that `Q2` becomes prevalent in stage 2. In contrast, the smaller-size relation for `V3` would replace *at stage 2* the larger-size relation for `V2`, if query `Q3` becomes prevalent in stage 2; here, the use of `V3`, rather than of `V2`, would result in more efficient evaluation of the query `Q3`.

The above example illustrates that our two-stage approach turns out to be more efficient and less resource intensive than one-stage approaches to view selection. Indeed, as we prove in this paper, if we apply a one-stage solver to materialize views only at stage 1, then a great portion of the materialized views tends to be irrelevant at stage 2. On the other hand, applying a one-stage solver at each of the two stages tends to result in *re-creation* of large volumes of views at stage 2, which could be resource intensive and would be avoided using our two-stage approach.

Note also that our two-stage approach to view selection is applicable to, and sufficient for, solving multi-stage view-selection problems. The reason is, the view selection takes place during the first season, and when it comes to the second season, any modification of the set of views with respect to the new season will have been applied. During the second season, we can employ this two-stage approach for view selection for the second and third seasons, and so on.

We now discuss why we have chosen the storage-limit constraint for our two-stage view-selection problem. Indeed, it would seem that for the views to be materialized at stage 2, any view-size estimates that we obtain at stage 1 would be far off by the beginning of stage 2. It turns out that this intuition is misleading for many real-life settings. For instance, in a typical commercial data warehouse, both the data-analysis queries and the views that we consider for materialization are defined using grouping and aggregation. Interestingly, the combinations of values of the `GROUP BY` attributes in the views would typically remain stable between two seasons, under the typical assumptions that (as, e.g., in the Sears scenario) the number of item categories, as well as the number of customer types,

would remain relatively large and relatively unchanged. As a result, the stage-1 estimates for the sizes of the view relations would hold relatively accurate also in stage 2. Further, for those stage-1 materialized views that end up also being used in stage 2, we assume that ongoing view maintenance takes place across the seasons/stages (see [15, 23] for overviews). Finally, we observe that once a view-selection problem is understood well in presence of a storage limit (see, e.g., [2]), then it can be addressed more easily in presence of other constraints, such as the limit on the view-maintenance costs. Studying the two-stage view-selection problem under the latter constraint, as well as under more general constraints such as introduced in [7], is part of our planned future work. We also plan to study combined selection of views and indexes.

The specific contributions of this paper are as follows:

1. We define the two-stage stochastic view-selection problem, and model this problem as a two-stage stochastic-programming (SP) model [6].
2. We study the structure and properties of the extensive form [6] of the SP model, which is an integer programming (IP) model. We develop an algorithm that effectively reduces the search spaces of potentially beneficial views in the IP model, and obtain a smaller IP model whose solution is guaranteed to be optimal for the original SP model.
3. We present our computational experiments on these IP models, and discuss the scalability of the reduced IP model. The reduced search spaces significantly reduce the size of the IP model, so that for realistic-size instances of the problem this IP model can be solved efficiently by a commercial IP solver, such as the powerful latest versions of CPLEX [20].
4. We compare our two-stage SP model with several related models, and present techniques to assess the value of the SP model and the value of perfect information in this context.

The remainder of this paper is organized as follows. In Section 2 we review related work. In Section 3, we discuss the formulation and settings for the problem. In Section 4, we define the two-stage stochastic view selection problem, and propose an SP model for it. In Section 5, we discuss the structure of the SP model, and use it to reduce the size of the problem considerably. In Section 6, we present and discuss the computational results. In Section 7, we present appropriate models and techniques to determine the value of the SP model and discuss the value of perfect information in this context, followed by several related numeric results. Finally, Section 8 contains a few concluding remarks.

The results of this paper can be used directly for multi-stage view selection for evaluating seasonal query workloads on large datasets, and can be incorporated into frameworks such as [1], for view selection for data-analysis queries.

2 Related work

The problem of view selection to improve query-evaluation costs and under a variety of constraints has been considered for relational databases for a number

of years. One line of past research considers the *one-stage deterministic environment,* that is, the environment where all queries are assumed to be known and given in advance. A variation of this setting is the *one-stage probabilistic environment,* where we have a probability (frequency) of occurrence associated with each query in a given set of queries. For some of the projects and for recent overviews, please see [2, 12, 13, 16, 26, 27]. Significant work has also been done on *index* selection in such settings, both on its own and alongside *view* selection, please see [1, 2, 8, 9, 13]. In both settings, the problem is addressed in the context of one fixed current query workload. In contrast, our work in this paper applies to a *two-stage* setting, where the second (future) stage occurs at a fixed time (season) in the future. To the best of our knowledge, view selection in this setting has not been considered in the open literature.

A separate line of research has studied *dynamic view selection,* where relational views are selected continuously, to respond to the changes in the query workload over time [8,9,22]. Our work is different in that it considers view design for a stage (season) that occurs in the future, at a fixed expected time and after the view design has been completed.

Self-managing data-intensive systems are also increasingly being studied beyond the relational setting; see, for instance, the intriguing results of [17]. In contrast, our proposed approach is applicable to the relational setting.

Returning to the relational one-stage view-selection setting, we note that a line of past work [3–5] has focused on formal approaches to selection of views (with or without indexes) in that setting. The results of that work are scalable to realistically large numbers of queries and views, and compare beneficially with several one-stage view-selection approaches in the literature, please see [3–5] for the details. For these reasons, we build our proposed approach, which focuses on the two-stage setting, on the results of [3–5], which were developed for the one-stage setting. (Please see Section 3.1 for an overview.)

To the best of our knowledge, Shaman [14] is the only project described in the open literature that considers using derived data to improve query performance in presence of a number of distinctly-identifiable query workloads, all of which are known beforehand. Our work complements well the agenda of Shaman in that in Shaman, the specific query sets are not tied to specific time points or to particular occurrence probabilities. The objective of the Shaman approach is to set up one initial and, at the same time, final *index* (as opposed to view) configuration, which would be "robust" in some sense through all the deterministic changes in the query workloads on the system, regardless of when the workload changes are to occur. This approach does not appear to be applicable to a probabilistic environment problem as we consider here, nor does it involve selection of views.

3 Preliminaries

We consider a star-schema data warehouse [10, 11] with a single fact table and several dimension tables. We assume that all the views to be materialized are defined, with grouping and aggregation but without selection, on the relation (which we call the *raw-data view*) that is the result of the "star-schema join"

of all the relations in the schema. We can show formally that for each query posed on such a database, the query can be rewritten equivalently into a query posed on the raw-data view. Using this formal result, in the remainder of the paper we assume that all the queries in the workloads that we consider are posed on the relevant raw-data view. In this context, we consider the evaluation costs of answering unnested select-project-join queries with grouping and aggregation using unindexed materialized views, such that each query can be evaluated using just one view and no other data. (This setting is the same as in [16, 21, 25, 30].) A query q can be answered using a view v only if the set of grouping attributes of v is a superset of the set of attributes in the `GROUP BY` clause of q and of those attributes in the `WHERE` clause of q that are compared with constants. We use v to represent both a view and the collection of grouping attributes for that view, and we use q to represent both a query and the collection of attributes in the `GROUP BY` clause of that query, plus those attributes in the `WHERE` clause of the query that are compared with constants. It follows that query q can be answered by view v if and only if $q \subseteq v$. To evaluate a query using a given view (if this view can indeed be used to answer the query) we have to scan all rows of the view. Hence the corresponding evaluation cost is equal to the size of the view itself; similar cost calculation is used in [16, 21, 25, 30]. One way to estimate the view sizes, as suggested in the literature, is by getting a relatively small-size sample of the raw-data view and by then evaluating the view definitions on that table, with a subsequent scaleup of the sizes of the resulting relations. We use a_i to denote the size of each view v_i in the problem input. We also use the parameter d_{ij} to denote the evaluation cost of answering query q_j using view v_i. It follows that for each query q_j we have $d_{ij} = a_i$ if $q_j \subseteq v_i$, and we set $d_{ij} = +\infty$ otherwise, implying that q_j cannot be answered by view v_i.

3.1 An integer programming method for the one-stage view-selection problem

In this subsection we briefly introduce the models and methods proposed in [3,4] for the one-stage view-selection problem. The models and algorithms that we propose for the two-stage problem are closely related to those for the one-stage problem that we describe here.

In [4], Asgharzadeh et al. consider the following problem: Given a collection Q of queries with an associated frequency for each query on a given star-schema data warehouse, and a storage limit b on the total size of the views that we may materialize, select a collection of views to materialize so as to minimize the total response time for the given queries.

Asgharzadeh et al. [3, 4] propose an IP model for solving the one-stage view-selection problem. For completeness, we introduce this IP model here. Let V denote the entire set of views on the raw-data view, which (set) is also the original search space of views for the one-stage view-selection problem. Let I and J denote the set of subscripts associated with V and Q, respectively. They define the decision variables x_i and z_{ij} for all $j \in J$ and for all $i \in I$, as follows:

$$x_i = \begin{cases} 1 & \text{if view } v_i \text{ is materialized} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{ij} = \begin{cases} 1 & \text{if we use view } v_i \text{ to answer query } q_j \\ 0 & \text{otherwise} \end{cases}$$

The one-stage view-selection problem can now be formulated as an integer programming (IP) model, denoted by $OVIP$, as follows.

$$(OVIP) \quad \text{minimize} \quad \sum_{j \in J} \sum_{i \in I} d_{ij} z_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{i \in I} z_{ij} = 1 \qquad \forall j \in J \tag{2}$$

$$z_{ij} \leq x_i \qquad \forall j \in J, \forall i \in I \tag{3}$$

$$\sum_{i \in I} a_i x_i \leq b \tag{4}$$

$$\textit{All variables are binary} \tag{5}$$

Constraint (2) states that each query is answered by exactly one view; (3) guarantees that a query can be answered by a view only if the view is materialized. Constraint (4) limits the storage space for the views to be materialized.

4 The two-stage stochastic view-selection problem

In this section, we define the scope of the two-stage stochastic view-selection problem that we consider, that is, the type of the database, queries and views. Subsequently, we propose a stochastic programming (SP) model [6] for this problem. By studying the properties of the SP model, we then rewrite it as a model with fewer variables and constraints.

4.1 Formulation of the problem SVS

For the view-selection problem in a two-stage probabilistic environment, we have a query workload Q_1 that we must answer at the present time (stage 1), and a second query workload $\mathbf{Q}_2$ that occurs at a future point in time (stage 2). We assume that the query workload Q_1 is known and given, but that the second query workload $\mathbf{Q}_2$ is a random set with a given probability distribution function. (We use boldface to emphasize that $\mathbf{Q}_2$ is a random set and to differentiate it from a deterministic set such as Q_1.)

At stage 1, we materialize a collection of views S_1 and use these views to answer the queries in Q_1. We assume that we have a storage limit b_1 for these views – that is, the total size of the views selected must not exceed b_1. At stage 2, once the actual collection of queries Q_2 for this stage is known, we allow for a partial replacement of some of the view relations that we constructed at stage 1, in order to obtain the collection of views S_2 for answering the query set Q_2. (Note that Q_2 represents a realization of the random set $\mathbf{Q}_2$.) In other words,

at the end of stage 1 we keep some of the view relations that we materialized at stage 1 to use again at stage 2, while we discard other view relations from stage 1 and replace them with new view relations. We assume that the total size of the views that we replace must not exceed a given storage limit b_2. Naturally, $b_2 \leq b_1$.

In this context the technical problem that we need to address prior to stage 1 is to select all views in the set S_1 and a replacement plan associated with each possible realization Q_2 of $\mathbf{Q}_2$. The objective is to minimize the total evaluation cost for the queries at stage 1 plus the expected total evaluation cost for the queries at stage 2. Note that in order to obtain a global optimal solution for this problem, which we call the ***S***tochastic ***V***iew ***S***election *(SVS)* problem, all decisions regarding both stages (that is, the view set S_1 and the replacement plan for every possible realization Q_2 of $\mathbf{Q}_2$) must be made prior to stage 1.

To illustrate, we present the following numeric example for a data cube [16] with four attributes.

Example 1. Given a database with four attributes a, b, c and d, the corresponding view lattice, as defined in [16], is shown in Figure 1. In this lattice, each node represents a view, and a directed edge from node v_1 to node v_2 implies that v_1 is a parent of v_2, that is, v_2 can be obtained from v_1 by aggregating over one attribute of v_1. The space requirement for each view in the lattice is given next to its corresponding node. In this instance, we assume that at stage 1, we are given two queries $q_1 = \{a, b\}$ and $q_2 = \{b, c\}$. At stage 2, queries $q_3 = \{b\}$ and $q_4 = \{a, c\}$ would occur with probability 0.5, and the other workload $\{q_5, q_6\}$, with $q_5 = \{c\}$ and $q_6 = \{b, d\}$, would also occur with probability 0.5. Equivalently, $Q_1 = \big\{\{a, b\}, \{b, c\}\big\}$, $Q_2^1 = \big\{\{b\}, \{a, c\}\big\}$, and $Q_2^2 = \big\{\{c\}, \{b, d\}\big\}$. We assume the total space limit $b_1 = 30$, and the space limit $b_2 = 15$. Our objective is to minimize the cost of answering the first-stage queries Q_1 plus the expected cost of answering the second-stage queries. Thus, we are to determine a collection of views S_1 to materialize at stage 1, with the total size less than or equal to 30, and for each scenario at stage 2, we need to determine another collection of views, with the total size less than or equal to 15, to materialize as replacement for a subset of S_1.

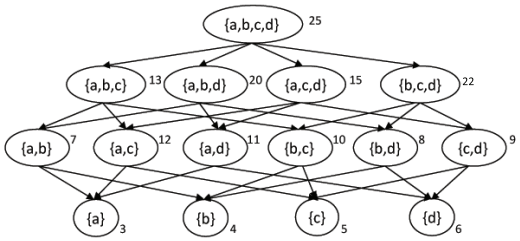

Fig. 1. View lattice for Example 1, with view sizes shown as number of bytes.

4.2 A mathematical programming model

In this subsection, we propose a stochastic programming model for the above view-selection problem SVS. Although the general form of the model that we propose is valid for any probability distribution function for $\mathbf{Q}_2$, for the ease of exposition in the following model and throughout the rest of this paper we assume that the random query set $\mathbf{Q}_2$ has a discrete distribution with L possible values. More specifically, we assume $\mathbf{Q}_2$ equals to query set Q_2^ℓ with probability p_ℓ, for $\ell = 1$ to L, where $\sum_{\ell=1}^{L} p_\ell = 1$. We refer to each collection of queries Q_2^ℓ as a *scenario*. For the first stage we define the following decision variables for all views $v_i \in V$ (where V is the set of all views) and for all queries $q_j \in Q_1$.

$$x_i = \begin{cases} 1 & \text{if view } v_i \text{ is materialized at stage 1} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{ij} = \begin{cases} 1 & \text{if we use view } v_i \text{ to answer query } q_j \text{ at stage 1} \\ 0 & \text{otherwise} \end{cases}$$

For the second stage, we define the following decision variables for all $v_i \in V$, and for all $q_j \in Q_2^\ell$, for $\ell = 1$ to L.

$$u_i^\ell = \begin{cases} 1 & \text{if view } v_i \text{ is materialized at stage 1 and} \\ & \quad \text{used in stage 2 for query set } Q_2^\ell \\ 0 & \text{otherwise} \end{cases}$$

$$y_i^\ell = \begin{cases} 1 & \text{if view } v_i \text{ is materialized at stage 2 for } Q_2^\ell \\ 0 & \text{otherwise} \end{cases}$$

$$t_{ij}^\ell = \begin{cases} 1 & \text{if we use view } v_i \text{ to answer query } q_j \text{ at} \\ & \quad \text{stage 2 for } Q_2^\ell \\ 0 & \text{otherwise} \end{cases}$$

The cardinality of the view set V is 2^K, where K is the number of dimension attributes in the database. Let $I = \{1, 2, \ldots, 2^K\}$ be the set of subscripts for all $v_i \in V$. Let J_1 be the set of subscripts for all queries $q_j \in Q_1$, and J_2^ℓ be the set of subscripts for all queries $q_j \in Q_2^\ell$, for $\ell = 1, 2, \ldots, L$. The problem can now be written as the following stochastic programming model [6] that we denote by SP.

$$(SP) \quad \text{minimize} \quad \sum_{j \in J_1} \sum_{i \in I} d_{ij} z_{ij} + \mathbf{E}_{\mathbf{Q_2}} \Psi(\mathbf{x}, \mathbf{Q}_2) \tag{6}$$

$$\text{subject to} \quad \sum_{i \in I} z_{ij} = 1 \quad \forall j \in J_1 \tag{7}$$

$$z_{ij} \leq x_i \quad \forall i \in I \; \forall j \in J_1 \tag{8}$$

$$\sum_{i \in I} a_i x_i \leq b_1 \tag{9}$$

All variables are binary

where $\mathbf{E}_{\mathbf{Q_2}}$ denotes mathematical expectation of response time at the second stage with respect to $\mathbf{Q}_2$. If the probability distribution of $\mathbf{Q}_2$ is as discussed above, we have

$$\mathbf{E}_{\mathbf{Q_2}}\Psi(\mathbf{x}, \mathbf{Q_2}) = \sum_{\ell=1}^{L} p_\ell \Psi(\mathbf{x}, Q_2^\ell) \tag{10}$$

where $\Psi(\mathbf{x}, Q_2)$ is the minimum response time for a given set of values of the first-stage variables $\mathbf{x} = \left(x_1, \cdots, x_{|V|}\right)$ and for a realization of the second-stage queries Q_2. For each value of ℓ, the corresponding value of $\Psi(\mathbf{x}, Q_2^\ell)$ is obtained by solving the following view-selection problem.

$$\Psi(\mathbf{x}, Q_2^\ell) = \min \sum_{j \in J_2^\ell} \sum_{i \in I} d_{ij} t_{ij}^\ell \tag{11}$$

$$\text{subject to} \quad \sum_{i \in I} t_{ij}^\ell = 1 \qquad \forall j \in J_2^\ell \tag{12}$$

$$t_{ij}^\ell \leq u_i^\ell + y_i^\ell \quad \forall i \in I \;\; \forall j \in J_2^\ell \tag{13}$$

$$u_i^\ell \leq x_i \qquad \forall i \in I \tag{14}$$

$$\sum_{i \in I} a_i y_i^\ell \leq b_2 \tag{15}$$

$$\sum_{i \in I} a_i (u_i^\ell + y_i^\ell) \leq b_1 \tag{16}$$

$$\textit{All variables are binary} \tag{17}$$

Constraints (7) and (12) state that each query is answered by exactly one view in the set of materialized views. Constraints (8) and (13) guarantee that a query can be answered by a view only if the view is already materialized. Constraints (9), (15), and (16) pertain to the storage limits on the views. Constraint (14) guarantees that the view kept from stage 1 to stage 2 is already materialized at stage 1.

In practice, for each decision variable with a subscript i (associated with view v_i) we only need to consider those views v_i that are relevant in the context of that decision variable. For instance, at stage 1 we do not need to define the decision variable x_i if view v_i is not a superset of at least one query in our entire query set. In the remainder of this section we define various subsets of the overall view set V (and its corresponding subscript set I) so as to rewrite the above model with fewer variables and constraints.

Let $\widehat{Q} = Q_1 \cup \left\{\cup_{\ell=1}^{L} Q_2^\ell\right\}$ be the collection of all queries (either in stage 1 or in stage 2). We define

$$V_1 = \left\{v_i \in V : v_i \supseteq q \text{ for some } q \in \widehat{Q}\right\} \tag{18}$$

$$V_2^\ell = \left\{v_i \in V : v_i \supseteq q \text{ for some } q \in Q_2^\ell\right\}, \ell = 1 \text{ to } L \tag{19}$$

It follows that V_1 is the set of views that are relevant for stage 1, or equivalently, V_1 is the search space for the views to be materialized at stage 1. V_2^ℓ is the set of views that are relevant for the ℓ^{th} sub-problem (scenario) in stage 2, for $\ell = 1$ to L. Equivalently, V_2^ℓ is the search space for views that are kept from stage 1 to stage 2, as well as those to be materialized at stage 2. In addition, for

each query $q_j \in Q_1$ we define the relevant view set $V_{1j} = \{v_i \in V_1 : v_i \supseteq q_j\}$, and for each query $q_j \in Q_2^\ell$ we define the relevant view set $V_{2j}^\ell = \{v_i \in V_2^\ell : v_i \supseteq q_j\}$, for $\ell = 1$ to L.

EXAMPLE 1 (CONTINUED). *In order to define V_1, V_2^1 and V_2^2, we only consider the views that could answer at least one query in the query set $\widehat{Q} = \{q_1, q_2, q_3, q_4, q_5, q_6\}$, Q_2^1 and Q_2^2, respectively. Thus, we obtain that*

$$\begin{aligned} V_1 = \big\{&\{b\},\{c\},\{a,b\},\{a,c\},\{b,c\},\{b,d\},\{c,d\},\{a,b,c\},\\ &\{a,b,d\},\{a,c,d\},\{b,c,d\},\{a,b,c,d\}\big\}\\ V_2^1 = \big\{&\{b\},\{a,b\},\{a,c\},\{b,c\},\{b,d\},\{a,b,c\},\{a,b,d\},\\ &\{a,c,d\},\{b,c,d\},\{a,b,c,d\}\big\}\\ V_2^2 = \big\{&\{c\},\{a,c\},\{b,c\},\{b,d\},\{c,d\},\{a,b,c\},\{a,b,d\},\\ &\{a,c,d\},\{b,c,d\},\{a,b,c,d\}\big\}. \end{aligned}$$

Each corresponding relevant *view set can be obtained as defined above. E.g., for $q_1 = \{a,b\} \in Q_1$ we have $V_{11} = \big\{\{a,b\}, \{a,b,c\}, \{a,b,d\}, \{a,b,c,d\}\big\}$, and for $q_6 = \{b,d\} \in Q_2^2$ we have $V_{26}^2 = \big\{\{b,d\}, \{a,b,d\}, \{b,c,d\}, \{a,b,c,d\}\big\}$.*

We now redefine the first-stage decision variables x_i and z_{ij}, only for views in V_1 and for views in V_{1j}, respectively. Similarly, we redefine the second-stage decision variables u_i^ℓ and y_i^ℓ, only for views in V_2^ℓ, and the second-stage decision variables t_{ij}^ℓ, only for views in V_{2j}^ℓ, for $\ell = 1$ to L. Thus, we can rewrite the model SP as a stochastic programming model with fewer variables and constraints without affecting its optimal value. We refer to this smaller model as SP'.

5 Solving the model SP

In this section we introduce an integer programming (IP) model [29] obtained from the extensive form [6] of the stochastic programming model SP'. We then study properties of this IP model, and use these properties to remove some variables and constraints, to obtain a significantly smaller model. The optimal solution of the resulting model is guaranteed to be optimal for the original model SP. This, in turn, allows us to solve larger instances of the problem SVS.

5.1 An integer programming model

To solve our problem SVS, we write the extensive form [6] of the model SP' by explicitly substituting Equations (11) through (17) for each value of ℓ in Equation (10) and ultimately in Equation (6) of model SP'.

For an instance with K attributes, n_1 queries in Q_1 at stage 1 and n_2^ℓ queries in Q_2^ℓ at stage 2, for $\ell = 1$ to L, there are up to $(n_1 + \sum_{\ell=1}^{L} n_2^\ell + 2L + 1)|V_1|$ variables and up to $(n_1 + \sum_{\ell=1}^{L} n_2^\ell + 2L)(|V_1| + 1) + 1$ constraints in the model $IP1$, where V_1 is as defined in Section 4.2. Thus, even for relatively small values of K, L, n_1 and $n_2^1, \cdots, n_2^L$, the size of the model could be very large and the

execution time for solving the IP model could be excessively long even with a relatively fast IP solver such as CPLEX 11 [20]. The applicability of this model is thus limited to only small-size instances. We now introduce several techniques to reduce the size of the model, by reducing the view sets V_1, V_{1j}, V_2^ℓ and V_{2j}^ℓ, for $\ell = 1, \ldots, L$. In Section 6, we will provide numerical results for solving the model $IP1$.

5.2 Reducing the search space of views

We now make several observations regarding properties of the views that appear in an optimal solution for a given problem SVS. For brevity we do not include the proofs for these observations here, and refer the readers to [19] where all the proofs are given in detail. These observations allow us to reduce the search space of views by identifying relatively smaller subsets of the view sets V_1, V_{1j}, V_2^ℓ and V_{2j}^ℓ, which contain at least one set of the optimal views. This, in turn, allows us to reduce the size of the model $IP1$, hence enabling us to solve significantly larger instances of the problem SVS optimally within reasonable execution times.

The first reduction of the search space of views is based on the relationship between the attributes of a view and the queries that the view can answer. Given a view v and a set of queries Q, let $Q(v)$ denote the set of queries in Q that the view v can answer, that is, $Q(v) = \{q \in Q : q \subseteq v\}$. We make the following observation.

Observation 1 *In an instance of problem SVS, given a view v in the view set V_1, if the number of attributes of v is strictly greater than the number of attributes in the union set of the queries in $\widehat{Q}$ that v could answer, that is, if $|v| > |\cup_{q \in \widehat{Q}(v)} q|$, then there exists an optimal solution such that v is not materialized at stage one.*

We make a similar observation concerning the attributes of each view v in the set V_2^ℓ and the attributes of the associated queries in the query set $Q_2^\ell(v)$ (see [19] for all the proofs and other details). These observations allow us to remove from the view sets V_1 and V_2^ℓ, for $\ell = 1$ to L, all the views that satisfy the stated conditions, hence reducing the search spaces of views.

The second reduction is based on the relationship between the *size* of a view and the total size of the queries that it can answer. Let $S(\cdot)$ be the estimated size of the answer to a query or view. For each view v for query set Q we define the corresponding *benefit* $d(v, Q)$ as follows:

$$d(v, Q) = \sum_{q \in Q(v)} S(q) - S(v) \tag{20}$$

Note that $d(v, Q)$ is the maximum *benefit* that we may obtain by materializing the view v, that is, the amount of space that we can save by materializing v instead of materializing all the queries that v can answer. We observe the following.

Observation 2 *In an instance of problem SVS, given a view v in the view set V_1, if v is not a query in $\widehat{Q}$ and v satisfies the condition that the size of v is greater than or equal to the total size of the queries in $\widehat{Q}$ that v can answer, that is, $d(v, \widehat{Q}) < 0$, then there exists an optimal solution in which v is not materialized at stage one.*

We make a similar observation regarding the size of each view v in the set V_2^ℓ and the total size of all the associated queries in the query set $Q_2^\ell(v)$ (see [19] for all the proofs and other details). These observations allow us to further reduce the search spaces of views V_1 and V_2^ℓ, for $\ell = 1$ to L, and this constitutes our second reduction.

We derive the third reduction using the relationship among the maximum benefits of views in the view lattice, see Eq. (20). The reduction is based on the following observation.

Observation 3 *In an instance of problem SVS, given a view v in V_1, if there exists a view v' in V_1 such that $v' \subset v$ and $d(v', \widehat{Q}) \geq d(v, \widehat{Q})$, then there exists an optimal solution such that v is not materialized at stage one.*

Again, we make a similar observation concerning each view v in V_2^ℓ, by comparing the maximum benefit of v for Q_2^ℓ with the maximum benefit of each subset of v (see [19]). These observations allow us to further reduce the search spaces of views for V_1 and V_2^ℓ, for $\ell = 1$ to L, and this constitutes our third reduction.

EXAMPLE 1 (CONTINUED). *Consider view $v_1 = \{b, c\}$ in the set V_1: The view answers only query $q_5 = \{c\}$ in $\widehat{Q}$, yet v_1 has more attributes than q_5. Hence, by Observation 1, there exists an optimal solution that does not contain view v_1. Consider now view $v_2 = \{a, b, d\}$ in the set V_1; the queries that v_2 can answer are $q_1 = \{a, b\}$, $q_3 = \{b\}$, and $q_6 = \{b, d\}$. Since the total size of these three queries $(4 + 7 + 8 = 19)$ is smaller than the size (20) of v_2, Observation 2 implies that there exists an optimal solution that does not contain v_2. Finally, consider view $v_3 = \{b, c, d\}$ in the set V_1; view v_3 is a superset of view $v_4 = \{b, d\}$. We have that $d(v_4, \widehat{Q})$ $(4+5+10-10 = 9)$ is greater than $d(v_3, \widehat{Q})$ $(4+5+8+10-22 = 5)$. Thus, by Observation 3, there exists an optimal solution that does not contain view v_3.*

Note that the conditions for the first, second, and third reductions are independent from each other. We can show formally that the results of the three reductions do not depend on the order of their application. Thus, the three reductions on the same set of views could be conducted simultaneously. Due to the page limit, we refer the reader to [19] for the details of our proposed reduction algorithm. The complexity of the algorithm is $O(|V|^2) = O(4^K)$, where K is the number of attributes in the database.

We denote the reduced view sets of V_1 and V_2^ℓ as $\overline{V_1}$ and $\overline{V_2^\ell}$, for $\ell = 1$ to L, respectively. Similarly, we use the notation $\overline{V_{12}^\ell}$ to denote the reduced search space for views that are materialized at stage 1 and utilized for scenario ℓ at stage 2. We can show that $\overline{V_{12}^\ell} = \overline{V_1} \cap V_2^\ell$. We also use the following notation: For each query $q_j \in Q_1$, let $\overline{V_{1j}} = \{v_i \in \overline{V_1} : v_i \supseteq q_j\}$, and for each query

$q_j \in Q_2^\ell$, let $\overline{V_{12j}^\ell} = \{v_i \in \overline{V_{12}^\ell} : v_i \supseteq q_j\}$, and $\overline{V_{2j}^\ell} = \{v_i \in \overline{V_2^\ell} : v_i \supseteq q_j\}$, for $\ell = 1$ to L.

5.3 Modified integer programming model $IP2$

We now define the model $IP2$ as an integer programming model that is same as model $IP1$, except that we use the reduced search spaces of views in place of the corresponding original search spaces of views in $IP1$. The model $IP2$ can be smaller than model $IP1$. In Section 6, we empirically compare the sizes of the two models, and show that the reduction in the size of the model can be significant. Based on the above observations, an optimal solution for model $IP2$ is guaranteed to be optimal for $IP1$. Hence, it provides an optimal solution for the original problem SVS.

6 Experimental results of solving the problem SVS

Table 1. Comparing the search spaces in the models $IP1$ and $IP2$, for instances over the 13-attribute dataset

inst-ance	query sets $(\lvert Q_1\rvert, \lvert Q_2^1\rvert, \lvert Q_2^2\rvert)$	number of x_i		number of u_i^1		number of y_i^1		number of u_i^2		number of y_i^2	
		IP1 $\lvert V_1\rvert$	IP2 $\lvert\overline{V_1}\rvert$	IP1 $\lvert V_2^1\rvert$	IP2 $\lvert\overline{V_{12}^1}\rvert$	IP1 $\lvert V_2^1\rvert$	IP2 $\lvert\overline{V_2^1}\rvert$	IP1 $\lvert V_2^2\rvert$	IP2 $\lvert\overline{V_{12}^2}\rvert$	IP1 $\lvert V_2^2\rvert$	IP2 $\lvert\overline{V_2^2}\rvert$
1	(20,21,26)	7974	656	5292	641	5292	217	7924	653	7924	381
2	(20,21,23)	7381	587	6144	573	6144	83	6139	570	6139	252
3	(30,26,22)	8168	857	7952	849	7952	350	7254	836	7254	115
4	(30,36,33)	8150	1076	7840	1069	7840	422	7568	1067	7568	279
5	(40,47,41)	8144	2475	7905	2469	7905	1311	7685	2456	7685	965
6	(40,48,46)	8176	2369	7912	2345	7912	1281	7726	2332	7726	807
7	(50,53,56)	8132	2514	7484	2477	7484	733	7880	2506	7880	1048
8	(50,52,56)	8162	2932	7750	2885	7750	1000	7925	2922	7925	1947

In this section, we present the results of a computational experiment with models $IP1$ and $IP2$ in order to examine (i) the effectiveness of the model reductions proposed in Section 5.2, and (ii) the scalability of the model $IP2$. We construct a collection of instances of the problem SVS with varying sizes using a number of datasets generated via the TPC-H benchmark [28]. We then solve each instance using the models $IP1$ and $IP2$. All of our algorithms are implemented in C++; all the experiments were carried out on a 2.66GHz Intel 2 Quad processor with 3.25 GB RAM running Windows XP Professional. We used CPLEX 11 [20] to solve the integer programming models $IP1$ and $IP2$. We observe that the search spaces of views are significantly reduced in $IP2$ compared with $IP1$, which allows us to use $IP2$ to solve several realistic-size instances of the problem SVS. More specifically, our experimental results show that:

Table 2. Comparing the sizes and computing times in $IP1$ and $IP2$, for instances over the 13-attribute dataset

inst-ance	query sets $(\|Q_1\|, \|Q_2^1\|, \|Q_2^2\|)$	number of variables		number of constraints		time to build model (sec.)		time to solve model (sec.)	
		IP1	IP2	IP1	IP2	IP1	IP2	IP1	IP2
1	(20,21,26)	84,862	11,026	63,744	10,540	0.422	0.328	20.360	1.750
2	(20,21,23)	62,705	7,225	43,110	7,180	0.344	0.313	22.485	6.875
3	(30,26,22)	124,496	18,405	101,205	18,386	0.547	0.391	284.410	8.930
4	(30,36,33)	103,044	18,858	79,590	18,620	0.563	0.438	354.004	34.094
5	(40,47,41)	134,562	53,979	110,961	52,010	0.734	0.719	134.830	19.891
6	(40,48,46)	134,156	51,231	110,481	49,502	0.735	0.735	656.508	31.875
7	(50,53,56)	117,512	47,709	94,180	46,780	0.766	0.797	181.315	36.329
8	(50,52,56)	152,722	71,438	129,048	68,582	0.875	0.891	1084.764	189.890

- The size of the search spaces for the view sets, and the corresponding number of decision variables and constraints are significantly reduced in model $IP2$ compared with $IP1$. This reduction is more significant for instances with a smaller ratio of the number of queries over the total number of possible views.
- The size of the model $IP2$ (i.e., the number of its variables and constraints) is small enough to allow its use to optimally solve a number of realistic-size instances of the problem SVS within reasonable execution time.

In the remainder of this section we present a detailed description and analysis of our experiments.

6.1 Constructing the instances

The input parameters for an instance of the problem SVS are a database $\mathcal{D}$, query sets Q_1 and $\mathcal{Q}_2 = \{Q_2^1, \ldots, Q_2^L\}$, the associated probability vector $\boldsymbol{p} = (p_1, \ldots, p_L)$, and the space limits b_1 and b_2. We used three different datasets based on the TPC-H benchmark [28] – a 7-attribute dataset, a 13-attribute dataset, and a 17-attribute dataset. Each dataset was obtained by using the original stored TPC-H tables (with scale factor one), to generate a single relation, with 7, 13, or 17 grouping attributes, that results from the star join of a star-schema subset of the set of these TPC-H tables. We measure the size of each view relation in bytes.

For all instances in this section, we assume $L = 2$, that is, $\mathcal{Q}_2 = \{Q_2^1, Q_2^2\}$, and $p_1 = p_2 = 0.5$. For each query set, we generated the queries randomly, keeping the total sizes of all the query sets "approximately" the same. (See [19].)

The difficulty of solving a specific instance of the problem SVS depends on the relative magnitude of the storage-space limits, as compared with the size of the queries. In this section, for all instances based on the TPC-H datasets we assume that the storage-space limit b_1 is equal to one-fifth of the sum of the sizes of the queries in $\widehat{Q}$, and b_2 is one-half of b_1. This assumption guarantees that the instances of the problem SVS are nontrivial. (See [19] for the details.)

6.2 Reducing the search space of views

We compare the sizes of the models $IP1$ and $IP2$ for several randomly generated instances of the problem SVS. More specifically, we constructed 8 instances for the 7-attribute TPC-H dataset, and 8 instances for the 13-attribute TPC-H dataset. The number of queries in Q_1 for each instance ranges from 20 to 50. For each instance, we compare the number of views in the view sets V_1 and V_2^ℓ with their corresponding reduced subsets $\overline{V_1}$, $\overline{V_{12}^\ell}$ and $\overline{V_2^\ell}$. For the 8 instances on the 13-attribute dataset, we report these values in Table 1. For each instance, we also report the total number of variables and constraints in the IP models $IP1$ and $IP2$, as well as the time that it takes to build each model and the time that it takes to solve each model by the CPLEX IP solver. The results of the 8 instances over the 13-attribute dataset are shown in Table 2. Similar tables for the 8 instances over the 7-attribute dataset are reported in [19] (we omit them here due to the page limit); those results are similar to those for the 13-attribute dataset.

For these collections of instances, we make the following observations. The number of views in the search space for model $IP1$ is significantly reduced in $IP2$ over all the instances. Also, the size of $IP2$, as expressed by the total number of variables and constraints, is much smaller than that of $IP1$. While there is no significant difference between the build times for the models $IP1$ and $IP2$, the time to solve $IP2$ is significantly smaller than that for $IP1$. This reduction in size tends to be relatively more significant when we have more attributes in the dataset.

We also observe that the reduction in the number of each group of decision variables and the reduction in the size of the model are relatively more significant for instances with a small ratio of the number of queries to the total number of views in the dataset. In Tables 1 and 2, we compare the results for instances over the 13-attribute dataset. When we have a larger number of queries in each query set, the ratio of the number of queries to the total number of views increases (since the total number of views for a 13-attribute dataset is constant and equal to 8192), and the magnitude of associated reductions in the number of variables decreases. We made similar observations (see [19]) for the instances over the 7-attribute dataset and over the 17-attribute dataset.

Finally, for the instances with a relatively large reduction in the number of variables/constraints, the corresponding reduction of the time to solve the model is more significant.

6.3 Scalability of the model $IP2$

To evaluate the scalability of our proposed approach, we attempted to solve larger instances of the problem SVS. First, we note that for the 8 instances over the 13-attribute dataset that we report in Table 2, the execution time to solve the model $IP2$ ranges from 1.75 seconds to slightly over 3 minutes. The time to build the model $IP2$ is relatively insignificant. We also note that the time to solve the model increases as we increase the number of queries in the query

sets Q_1, Q_2^1, and Q_2^2. To further explore the execution time for larger instances of the problem, we constructed instances with even larger numbers of queries over the 13-attribute dataset, please see Table 3. For these instances we did not construct the model $IP1$, since we have already observed that the model $IP2$ is much more effective than $IP1$.

Table 3. $IP2$: scalability over 13-attribute dataset

instance	query sets $(\|Q_1\|, \|Q_2^1\|, \|Q_2^2\|)$	time to build model (sec.)	time to solve model (sec.)
1	(60,60,63)	0.875	41.860
2	(80,83,75)	1.328	153.907
3	(100,103,98)	2.078	217.140
4	(120,122,124)	1.797	584.688
5	(140,140,139)	2.125	451.078
6	(160,161,153)	2.485	$>$ 20min
7	(180,191,192)	2.953	$>$ 20min
8	(200,202,215)	3.203	$>$ 20min

Table 4. Solving $IP2$ over the 17-attribute dataset

instance	query sets $(\|Q_1\|, \|Q_2^1\|, \|Q_2^2\|)$	time to build model (sec.)	time to solve model (sec.)
1	(20,19,21)	20.141	63.297
2	(20,10,15)	18.719	22.844
3	(30,32,28)	21.578	out of memory
4	(30,38,35)	22.219	out of memory
5	(40,48,45)	out of memory	—
6	(40,31,35)	out of memory	—

From Table 3 we observe that we could solve within 10 minutes all the instances whose number of queries in Q_1 is no more than 140. However, when we further increase the size of the query set, the solver fails to provide an optimal solution within our time limit of 20 minutes.

In Table 4, we present a similar result for a few instances of the problem that are constructed based on the 17-attribute TPC-H dataset. We observe that for these instances CPLEX fails to obtain an optimal solution for the instances that have 30 or more queries. Furthermore, for the instance with 40 or more queries, we are not able to even build the model due to insufficient memory. We are presently designing special algorithms to solve larger instances of this problem.

7 Value of the two-stage stochastic model

In this section we assess the value of our proposed two-stage stochastic programming model SP by:

- measuring the gain in solution quality obtained by using the two-stage model versus the one-stage model;
- measuring the benefits resulting from taking into account the stochastic properties in model SP; and by
- discussing the value of obtaining further information about future events (i.e., about second-stage queries).

In Section 7.1, in the context of the problem SVS we compare our two-stage model SP with a corresponding one-stage model, which does not allow for replacement of views in stage 2. We define the *value of two-stage versus one-stage* ($VTVO$), as the difference between the optimal values of the two-stage model and the corresponding one-stage model. $VTVO$ measures the gain obtained from the view-replacement mechanism of the two-stage model SP.

In Section 7.2 we compare our model SP with a two-stage model that is based on the expected value of random events. We introduce the *value of stochastic solution* (VSS) [6], which evaluates the difference between the expected response times achieved by (i) solving the model SP, and by (ii) solving the *expected-value* model. VSS assesses the benefits obtained by explicitly considering the stochastic characteristics of the problem in constructing the model.

In Section 7.3 we show that our model SP is also beneficial in making decisions on whether or not to obtain further information about the occurrence of the second-stage queries. To do so, we define the *expected value of perfect information* ($EVPI$) [6] in the context of the problem SVS, and propose appropriate models to obtain its value.

Finally, Section 7.4 provides the related numerical results.

7.1 Two-stage versus one-stage

In this subsection, we introduce the *value of two-stage versus one-stage* ($VTVO$) for solving the problem SVS. Recall that in our two-stage stochastic view-selection model, we allow for view replacement at stage 2. The $VTVO$ measures the gain obtained via this replacement mechanism. In other words, for a given instance of the problem SVS, the value of $VTVO$ provides the magnitude of improvement in the optimal query-evaluation costs, as achieved by allowing for partial replacement of views at stage 2.

We begin by discussing the one-stage model, in which we do not allow to drop or replace any materialized views at stage 2. In this model, we must determine a set of views S to materialize at stage 1 within a given space limit b_1. We use the views S to answer the queries in Q_1, as well as all the queries in the query set Q_2^ℓ that may occur at stage 2.

We can formulate the one-stage problem as an integer programming model that we denote by OS. It turns out that the model OS is equivalent to the

integer programming model for solving the one-stage problem introduced in [4]. Further, the optimal value of OS is an upper bound on the optimal value of our model SP. (See [19] for the details.)

We can now define *the value of two-stage versus one-stage* ($VTVO$) as the difference between the optimal values of the one-stage and two-stage models, namely,

$$VTVO = Optv(OS) - Optv(SP), \tag{21}$$

where $Optv(\cdot)$ represent the optimal value of a model $(\cdot)$.

7.2 The value of stochastic solution (VSS)

The literature on stochastic programming (e.g., [6]) defines the *value of stochastic solution* (VSS) as the gain obtained from solving the stochastic programming model as opposed to a model based on the expected values. In our context, VSS is the difference between the optimal value of the stochastic programming model (our model SP) and the corresponding *Expected Value* (EV) model. The EV model is typically defined for the same two-stage environment, under the assumption that there is only one second-stage scenario that represents the *expected value* of the random events.

In the context of the problem SVS, we define the EV problem by considering the query workload at stage 2 as a *deterministic* query set $Q_2 = \cup_{\ell=1}^{L} Q_2^{\ell}$. We define the *weight* associated with each query $q \in Q_2^{\ell}$ as the sum of the probability values of the query sets that contain q. Thus the EV problem could be considered as a special case of the problem SVS where there is only one scenario at stage 2. It follows that the EV problem could be formulated as an integer programming model, which we denote by $IPEV$.

We further denote by $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{z}})$ the optimal values of the first-stage variables in the model $IPEV$. We refer to $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{z}})$ as the *expected value solution* (see [6]). We thus define the *expected result of using the EV solution* as

$$EEV = \sum_{j \in J_1} \sum_{i \in I_{1j}} d_{ij} \bar{z}_{ij} + \boldsymbol{E_{Q_2}} \boldsymbol{\Psi}(\bar{\boldsymbol{x}}, \boldsymbol{Q_2}) \tag{22}$$

where $\boldsymbol{\Psi}(\cdot)$ is defined in (11)-(17). We can show that EEV is an upper bound on the optimal value of model SP.

For EEV and $Optv(SP)$, we define the *value of stochastic solution* (VSS) [6] as the difference between these values:

$$VSS = EEV - Optv(SP). \tag{23}$$

VSS measures the degree to which the decision $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{z}})$ is inferior to the optimal solution of model SP. Again, we refer the reader to [19] for a detailed discussion of this subject.

7.3 The expected value of perfect information

It turns out that our model SP can also help in deciding whether making an effort to obtain further information about the second-stage queries would pay off. To show this, we define the *expected value of perfect information* ($EVPI$) for the problem SVS. In the literature (see, e.g., [6]), $EVPI$ measures the maximum amount of money a decision maker would be ready to pay in return for complete information about the future. In the problem SVS that would be the benefit, in units of the expected query-evaluation costs, from knowing the actual query workload occurring at stage 2. This benefit, $EVPI$, is the maximum amount of query-evaluation time that we could save if we knew the exact query set that will occur at stage 2. In other words, the decision maker can compare the $EVPI$ with the cost of determining the exact query workload to occur at stage 2, and can then make the decision to minimize the expected costs.

Suppose we know at stage 1 which exact query set will occur at stage 2. In this case, we only need to study the model SP that results from that particular realization of the second-stage scenarios. That is, for each scenario ℓ we make the associated first-stage and second-stage decisions exclusively for it, and obtain the minimum costs of answering the queries at stage 1 and stage 2 under that scenario. We refer to this problem as the ℓ^{th} *scenario* problem.

We formulate an integer programming model for the ℓ^{th} scenario, for $\ell = 1$ to L, refer to it as IP^{ℓ}, and denote its optimal value as $Optv(IP^{\ell})$. We then define the *expected optimal value under perfect information* as the expected value of this optimal value. Following a common practice in the literature [6], we refer to this value as the *wait-and-see solution value*, denoted by WS. We have that

$$WS = \sum_{\ell=1}^{L} p_{\ell} \cdot Optv(IP^{\ell}). \tag{24}$$

WS is a lower bound on the optimal value of our model SP.

In the literature [6], the difference between the optimal value of the stochastic programming model and its wait-and-see solution is referred to as "the expected value of perfect information," denoted by $EVPI$. We have that

$$EVPI = Optv(SP) - WS. \tag{25}$$

See [19] for further details.

7.4 Numerical results

We now discuss the computational experiments that we performed to assess the values of $VTVO$, VSS, and $EVPI$, as associated with our model SP. These results show that the magnitude of the values of $VTVO$, VSS, and $EVPI$ varies among instances, and that it depends on the structure and properties of both the database and the query workloads, as well as on the space limits. Specifically, these values are influenced significantly by (i) the ratio $\beta = b_2/b_1$, and by (ii)

the structure of the database and the relative size of each view as compared with its descendant views in the view lattice.

To examine the impact of the ratio $\beta = b_2/b_1$ on these entities, we conduct experiments over several instances, as follows. Given an instance of the problem SVS – that is, given a database $\mathcal{D}$, query sets Q_1 and $\mathcal{Q}_2$, probabilities $\boldsymbol{p}$ and the total space limit b_1 – we examine the impact of the value of β on the solutions by changing this value over $[0, 1]$.

We use three different types of datasets to test the impact of datasets: the TPC-H datasets [28] (see Section 6.1), our symmetric synthetic datasets [18], and our type I non-symmetric synthetic datasets [18]. Readers are referred to [18] for detailed descriptions of the synthetic datasets.

Numerical results on *VTVO* We evaluated $VTVO$ over a number of instances of the problem SVS with varying sizes over different datasets. We observed that the relative magnitude of $VTVO$ varies among these instances, depending on the relative value of the storage limits and on the structure of the dataset.

We examine first the impact of the space limit b_2 on the $VTVO$. We constructed an instance, referred to as Example 2, with $|Q_1| = 20$, $|Q_2^1| = 30$, and $|Q_2^2| = 22$, over the 13-attribute TPC-H dataset denoted by D_{T-13}. The specific query sets for this instance (as well as for the other examples here) are presented in [19]. We set $b_1 = 28,777,009$ bytes and $\boldsymbol{p} = (0.5, 0.5)$. Figure 2 shows the value of $VTVO$ for this instance as we change the value of the ratio $\beta = b_2/b_1$. As the plot shows, $VTVO$ is a non-decreasing function of β (or of b_2), and at $b_2 = 0$, we have $VTVO = 0$ as expected.

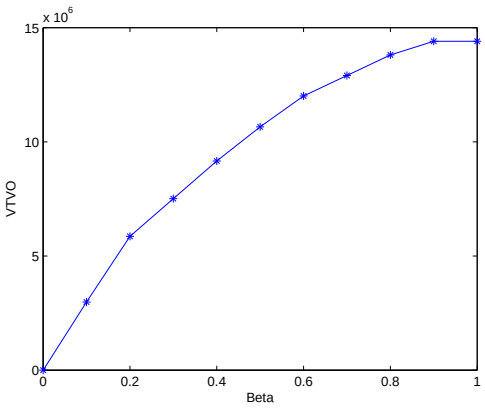

Fig. 2. The impact of β on $VTVO$ for Example 2

Next, we examine the impact of different datasets on the $VTVO$ by comparing the ratio of $VTVO$ over the optimal value of SP for the instances based on

different datasets. We compare the results of Example 2 (based on the dataset D_{T-13}), with the results of Example 3, which is based on a 13-attribute type I non-symmetric synthetic dataset [18] denoted by D_I. In Example 3, the number of queries in query sets Q_1 and $\mathcal{Q}_2 = \{Q_2^1, Q_2^2\}$ are 20, 15 and 21, respectively. We set $b_1 = 1,632,334$ bytes and $\boldsymbol{p} = (0.5, 0.5)$. The results are shown in Figure 3 for different values of the ratio β. We observe that for the same value of β, the instance based on D_I has a significantly higher ratio of $VTVO$ over $Optv(SP)$ than the instance based on D_{T-13}. For example, the benefit for the instance based on D_I can be 501.76% of the optimal value of the associated model SP, while the benefit for the instance based on D_{T-13} is no more than 13.48% of the optimal value of the associated model SP. This indicates that while we can obtain some benefit by applying the two-stage model instead of the one-stage model, the magnitude of the benefit depends on the structure and size of the queries and views involved.

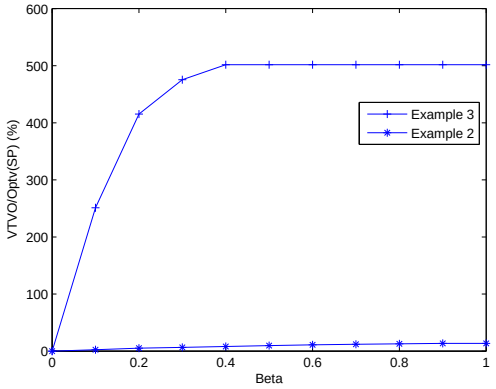

Fig. 3. The impact of β on $VTVO/Optv(SP)$

Numerical results for VSS In another computational experiment, we evaluated VSS over a collection of instances of varying sizes based on different types of datasets. The results show that the relative value of VSS varies among these instances, and that its magnitude depends on both the space limits and on the structure of the underlying database.

First we observe the impact of b_2 on VSS. Figure 4 shows the value of VSS for different values of β over Example 4. This example is based on the 7-attribute TPC-H dataset [28] denoted as D_{T-7}. The number of queries in query sets Q_1 and $\mathcal{Q}_2 = \{Q_2^1, Q_2^2\}$ are 20, 22, and 22, respectively. We set $b_1 = 21605174$ bytes and $\boldsymbol{p} = (0.5, 0.5)$. As the plot shows, when $b_2 = 0$ or $b_2 = b_1$ ($\beta = 0$ or $\beta = 1$), we have $VSS = 0$. When $0 < b_2 < b_1$, the impact of b_2 on VSS varies.

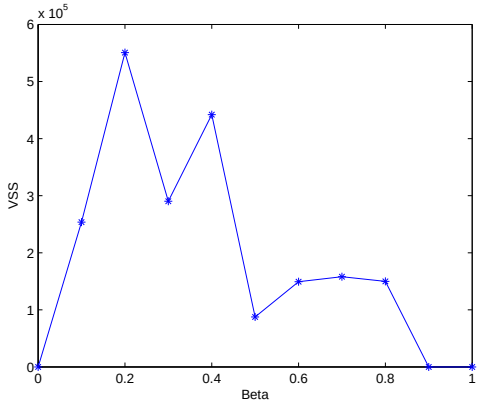

Fig. 4. The impact of β on VSS for Example 4

In general, given an instance of problem SVS, when $b_2 = 0$, the models SP and EV are equivalent to the one-stage model OS, see Section 7.1. Thus, $EEV = Optv(SP) = Optv(EV)$ and $VSS = 0$. When $b_2 = b_1$, the problems for the first and for the second stage can be solved separately. Thus, $EEV = Optv(SP)$ and $VSS = 0$. In both cases, no benefit can be obtained by taking into account the stochastic properties in solving the problem SVS. However, when $0 < b_2 < b_1$, the impact varies for different values of b_2.

To examine the impact of the underlying dataset, we compare the results for Example 4 (based on the dataset D_{T-7}) with the results for Example 5, which is based on the symmetric synthetic dataset D_S [18]. In Example 5, the number of queries in query sets Q_1, $\mathcal{Q}_2 = \{Q_2^1, Q_2^2\}$ are 20, 18, and 15, respectively. We set $b_1 = 15764$ bytes and $\boldsymbol{p} = (0.5, 0.5)$. The results are shown in Figure 5. We observe that the maximum value of the ratio $VSS/Optv(SP)$ among all the instances in Example 4 is 1.23%. In Example 5, the ratio has a maximum value of 50.14%, and has an average value of 19.95% for the instances with non-zero VSS. As we observe by comparing the results for Examples 4 and 5, the benefits obtained by taking into account the stochastic properties in solving the problem SVS depends on the specific instance and on the structure of its underlying database.

Numerical results for *EVPI* As in the previous experiments, we observe that the value of $EVPI$ could vary significantly, depending on both the space limits and the specifics of the database and query sets.

We first consider the impact of b_2 on the $EVPI$. Table 5 shows the $EVPI$ and the ratio of $EVPI/Optv(SP)$ for different values of β $(= b_2/b_1)$ over Examples 6 and 7. Example 6 is based on the TPC-H dataset D_{T-7}, and the values of its input parameters are the same as in Example 4. Example 7 is based on the symmetric synthetic dataset D_S, with $|Q_1| = 20$, $|Q_2^1| = 18$, and $|Q_2^2| = 15$.

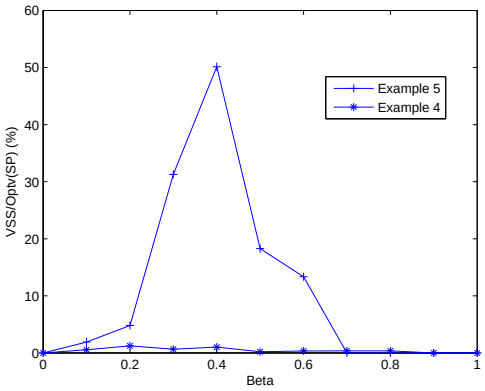

Fig. 5. The impact of β on the $VSS/Optv(SP)$

We set $b_1 = 7882$ bytes and $\boldsymbol{p} = (0.5, 0.5)$. As Table 5 shows, when $\beta \geq 0.7$ in Example 6 and $\beta \geq 0.8$ in Example 7, the $EVPI$ is 0. It indicates that the knowledge of the exact stage-2 query workload provides little benefit when b_2 is relatively large. For each example and for each value of β, the value of $EVPI$ is non-negative, and it varies as we change the value of β.

Table 5. Results for $EVPI$ over Examples 6 and 7

β	$EVPI$		$EVPI/Optv(SP)$	
	Example 6	Example 7	Example 6	Example 7
0	1,522,229	5734	3.25%	10.18%
0.1	1,072,046	2354	2.34%	4.53%
0.2	625,183	542	1.40%	1.16%
0.3	263,424	8416	0.60%	18.29%
0.4	123,707	9760	0.28%	22.24%
0.5	101,214	6477	0.24%	16.54%
0.6	964	2807	0.00%	9.23%
0.7	0	576	0.00%	2.16%
0.8	0	0	0.00%	0.00%
0.9	0	0	0.00%	0.00%
1.0	0	0	0.00%	0.00%

In general, if $b_2 = b_1$, the first-stage problem and the second-stage problem can be solved separately. As a result, there is no benefit obtained from perfect information, i.e., $EVPI = 0$. When $0 \leq b_2 < b_1$, the impact on $EVPI$ varies.

Finally, we observe the impact of the datasets on $EVPI$. We compare the results for Example 6 over the dataset D_{T-7}, and the results for Example 7 over the dataset D_S. From Table 5, observe that, in general, Example 6 has significantly

higher values of $EVPI$ than Example 7, and the ratios of $EVPI/Optv(SP)$ for Example 7 are generally higher than those for Example 6. This implies that both the relative magnitude and the absolute magnitude of $EVPI$ depends on the structure of the underlying dataset and on the specific instance of the problem.

8 Concluding remarks

In this paper we presented the two-stage stochastic view-selection problem SVS, and undertook a systematic study of the problem. We introduced a stochastic programming model for the problem, and showed that it is equivalent to an integer programming (IP) model. We studied the structure of this model, and proposed procedures to efficiently prune the search space of views, hence reducing the size of the IP model. We showed experimentally that the reduction in the size of the IP model is significant, and the resulting IP model can be solved by commercial IP solvers for small to medium realistic-size instances of the problem, within reasonable execution time. We also developed measures ($VTVO$, VSS, and $EVPI$) to evaluate the effectiveness of our proposed model and approach. Presently, to improve the efficiency and scalability of our approach, we are further investigating the structure of the IP models so as to design more effective exact and inexact methods for solving larger instances of the problem. This will allow for a wider potential applicability of our problem and models in the future.

References

1. S. Agrawal, N. Bruno, S. Chaudhuri, and V. R. Narasayya. AutoAdmin: Self-tuning database systems technology. *IEEE Data Eng. Bull.*, 29(3):7–15, 2006.
2. S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *VLDB*, pages 496–505, 2000.
3. Z. T. Asgharzadeh. *Exact and inexact methods for solving the view and index selection problem for OLAP performance improvement.* Phd dissertation, NCSU, 2010.
4. Z. T. Asgharzadeh, R. Chirkova, and Y. Fathi. Exact and inexact methods for solving the problem of view selection for aggregate queries. *IJBIDM*, 4(3/4):391–415, 2009.
5. Z. T. Asgharzadeh, R. Chirkova, Y. Fathi, and M. Stallmann. Exact and inexact methods for selecting views and indexes for OLAP performance improvement. In *EDBT*, pages 311–322, 2008.
6. J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming.* Springer, 1997.
7. N. Bruno and S. Chaudhuri. Constrained physical design tuning. *VLDB J.*, 19(1):21–44, 2010.
8. N. Bruno and S. Chaudhuri. Interactive physical design tuning. In *ICDE*, pages 1161–1164, 2010.
9. N. Bruno, S. Chaudhuri, and G. Weikum. Database tuning using online algorithms. In *Encyclopedia of Database Systems*, pages 741–744. Springer US, 2009.
10. S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, 1997.

11. S. Chaudhuri, U. Dayal, and V. R. Narasayya. An overview of business intelligence technology. *Commun. ACM*, 54(8):88–98, 2011.
12. S. Chaudhuri, V. R. Narasayya, and G. Weikum. Database tuning using combinatorial search. In *Encyclopedia of Database Systems*, pages 738–741. Springer US, 2009.
13. S. Chaudhuri and G. Weikum. Self-management technology in databases. In *Encyclopedia of Database Systems*, pages 2550–2555. Springer US, 2009.
14. S. Duan, P. Franklin, V. Thummala, D. Zhao, and S. Babu. Shaman: A self-healing database system. In *ICDE*, 2009.
15. H. Gupta. Self-maintenance of views. In *Encyclopedia of Database Systems*, pages 2548–2550. Springer US, 2009.
16. V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, 1996.
17. H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *CIDR*, pages 261–272, 2011.
18. R. Huang, R. Chirkova, and Y. Fathi. Synthetic datasets. Technical Report TR-2011-10, NC State University, 2011. `ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc_anon/tech/2011/TR-2011-10.pdf`.
19. R. Huang, R. Chirkova, and Y. Fathi. A two-stage stochastic view selection problem in database management systems. Technical Report TR-2011-8, NC State University, 2011. `ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc_anon/tech/2011/TR-2011-8.pdf`.
20. ILOG. CPLEX 11.0 software package, 2007. `http://www.ilog.com/products/cplex/`.
21. P. Kalnis, N. Mamoulis, and D. Papadias. View selection using randomized search. *DKE*, 42:89–111, 2002.
22. Y. Kotidis and N. Roussopoulos. A case for dynamic view management. *ACM TODS*, 26(4):388–423, 2001.
23. A. Labrinidis and Y. Sismanis. View maintenance. In *Encyclopedia of Database Systems*, pages 3326–3328. Springer US, 2009.
24. S. Lightstone. Physical database design for relational databases. In *Encyclopedia of Database Systems*, pages 2108–2114. Springer US, 2009.
25. A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. In *VLDB-98*.
26. D. Theodoratos, S. Ligoudistianos, and T. K. Sellis. View selection for designing the global data warehouse. *Data Knowl. Eng.*, 39(3):219–240, 2001.
27. D. Theodoratos and T. K. Sellis. Incremental design of a data warehouse. *J. Intell. Inf. Syst.*, 15(1):7–27, 2000.
28. TPC-H Revision 2.1.0. TPC Benchmark H (Decision Support). `http://www.tpc.org/tpch/spec/tpch2.1.0.pdf`.
29. L. A. Wolsey. *Integer Programming*. Wiley, 1998.
30. J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *VLDB*, pages 136–145, 1997.