A Study of a Formal Model

For View Selection for Aggregate Queries

Zohreh Asgharzadeh Talebi[1], Rada Chirkova[2], and Yahya Fathi[1]

[1] Operations Research Program, NC State University, Raleigh, NC 27695

{zasghar,fathi***}@ncsu.edu

[2] Computer Science Department, NC State University, Raleigh, NC 27695

chirkova@csc.ncsu.edu†

Abstract. We present a formal study of the following view-selection problem: Given a set of queries and a database, return definitions of views that, when materialized in the database, would reduce the evaluation costs of the queries. Optimizing the layout of stored data using view selection has a direct impact on the performance of the entire database system. At the same time, the optimization problem is intractable, even under natural restrictions on the types of queries of interest. We introduce an integer-programming model to obtain optimal solutions for the view-selection problem for aggregate queries on data warehouses. Through a computational experiment we show that this model can be used to solve realistic-size instances of the problem. We also report the results of the post-optimality analysis that we performed to determine the impact of changing certain input characteristics on the optimal solution, and compare our approach to a leading heuristic procedure in the field [21].

1 Introduction

As relational databases and data warehouses keep growing in size, evaluating many common queries — such as aggregate queries — by database-management systems (DBMS) may require significant transformations of large volumes of stored data. As a result, the require-

*** This author's work is partially supported by the National Science Foundation under Grant No. 0321635.

† This author's work is partially supported by the National Science Foundation under Grant No. 0307072.

ment of good overall performance of frequent and important user queries necessitates optimal DBMS choices in choosing and executing query plans.

A significant aspect of query performance is the choice of auxiliary data used in query answering, such as which indexes are used in a query plan to access a given stored relation. In modern commercial database systems, another common type of auxiliary data is *materialized views* — relations that were computed by answering certain queries on the (original) stored data in the database and that can be used to provide, without time-consuming runtime transformations, "precompiled" information that is relevant to the user query. We give an example of using materialized views to answer select-project-join queries with aggregation in a star-schema [23] data warehouse.

Example 1. Consider a star-schema data warehouse with three stored relations: `Sales(CID, DateID,QtySold,Discount)`, `Customer(CID,CustName,Address,City,State)`, and `Time(DateID, Day,Week,Month,Year)`; here, `Sales` is the fact table.

Let the query workload of interest have two aggregate queries, `Q1` and `Q2`, expressed here in SQL. Query `Q1` asks for the total quantity of products sold per customer in the last quarter of the year 2005. `Q2` asks for the total product quantity sold per year for all years after 2000 to customers in North Carolina.

```
Q1: SELECT c.CID, SUM(QtySold)
    FROM Sales s, Time t, Customer c
    WHERE s.DateID = t.DateID AND s.CID = c.CID
    AND Year = 2005 AND Month >= 9 AND Month <= 12
    GROUP BY c.CID;
```

```
Q2: SELECT t.Year, SUM(QtySold)
    FROM Sales s, Time t, Customer c
    WHERE s.DateID = t.DateID AND s.CID = c.CID
    AND Year > 2000 AND State = 'NC'
    GROUP BY t.Year;
```

We can use techniques from [21] to show that the following view V can be used to give exact answers to each of Q1 and Q2.

```
V: SELECT s.CID, Year, Month, State, SUM(QtySold) AS SumQS FROM Sales s, Time t, Customer c
   WHERE s.DateID = t.DateID AND s.CID = c.CID GROUP BY s.CID, Year, Month, State;
```

That is, suppose the view V is materialized in the data warehouse, which means that the answer to the query V on the database is precomputed and stored as a new relation V[3] alongside Sales, Customer, and Time. Then the answer to each of Q1 and Q2 can be computed by accessing just the data in the relation V. For instance, Q1 can be evaluated as

```
Q1: SELECT CID, SUM(SumQS) FROM V WHERE Year = 2005 AND Month >= 9 AND Month <= 12 GROUP BY CID;
```

Note that evaluating the query Q1 using the view V is likely to be more efficient than evaluating Q1 using its original definition, as using V allows the DBMS to avoid taking an expensive join of Sales, Customer, and Time and also — because V is an aggregate view — may save some time in the grouping and aggregation steps.

In this paper we consider the following view-selection problem: Given a set of queries, a database, and a set of constraints on derived data (e.g., a storage limit on the amount of disk space that can be used to store materialized views), return definitions of views that, when materialized in the database, would satisfy the constraints and reduce the evaluation costs of the queries. As automated design of materialized views to answer queries is an important component of query processing in data warehouses [7, 21, 32, 36, 39] and of automated query-performance tuning [22, 29, 31], the problems of selecting views and of answering queries using materialized views have been studied thoroughly in the literature.

[3] We follow the tradition of using the same name for a view query and its answer.

Generally, spending more time on designing views for a query workload tends to pay off, as greater improvement can thereby be achieved in the query performance using the resulting stored derived data. As the number of potentially beneficial views or indexes tends to be prohibitive even for simple query workloads [4, 13, 21], in many cases it is not practical to use exhaustive enumeration to obtain derived data that would *globally minimize* query costs. Several approaches (see, e.g., [4, 19, 21, 32]) have been proposed to design good-quality sets of derived data for SQL queries, without spending an inordinate amount of time on the design. We continue the work of [19, 21] of studying view-selection algorithms that are competitive, that is, provide optimality guarantees on their outputs without necessarily exploring the entire search space of views. In this paper we present a formal model of view selection for queries on star-schema data warehouses and explore competitive techniques for designing and using views in this context. Our techniques and results are applicable to a practically important class of range-aggregate queries on star-schema data warehouses.

Our contributions are as follows:

(1) we model view selection as an integer-programming (IP) problem and give references to similar IP structures in the literature (Section 3),

(2) we use standard IP-solver software (CPLEX [30] with an AMPL interface [16]) to solve optimally realistic-size instances of the problem on the popular TPC-H benchmark [37] and on real data in the Sloan Digital Sky Survey (SDSS) dataset [35] (Sections 4–6),

(3) we study the limits of our approach for two versions of the view-selection problem, determined by whether the table for the view resulting from joining all the base relations (we call this view *raw-data view*) is part of the solution (Sections 3 and 5),

(4) we experimentally compare our method with a leading heuristic approach in the field [21], and theoretically analyze the worst-case performance of that approach (Section 6).

In our experiments, which we report in Sections 5 and 6, we solved hundreds of problem instances, both on TPC-H data [37] and on the SDSS dataset [35]. The problem instances we solved used both randomly generated query workloads and workloads of queries related by an ancestor-descendant relationship[4] in the sense of [21].

After outlining related work, in Section 2 we provide the background and formal definitions. Section 3 introduces our IP model of the view-selection problem. Section 4 describes our framework for analysis and experimentation. We report our experimental results and theoretical analysis in Sections 5 and 6, and conclude in Section 7.

Related Work

Designing and using derived data to improve query performance has long been studied in data-intensive systems. A wealth of theoretical results (see [20] for a survey) and some practical solutions [5, 10, 12] have been accumulated on using views and indexes in query answering. Answering aggregate queries using views was considered in relation to data warehouses and data cubes [3, 9, 17, 38]; results on answering each query using a single view were presented in [18, 34]. Recent work [1, 14] considered rewriting aggregate queries using multiple views.

Considerable work has been done on efficiently selecting views and indexes for general SQL queries [4, 11] and in particular for aggregate queries (e.g., [1, 19, 21, 32]). [39] proposed

[4] A query Q_1 is a *descendant* of a query Q_2 — and Q_2 is Q_1's *ancestor* — if it is possible to obtain an exact answer to Q_1 using Q_2 as the sole data source.

algorithms, including an IP approach, for selecting materialized views to minimize the sum cost of processing the given queries and of maintaining all the views. [4, 5] introduced an end-to-end approach and a system architecture for designing and using materialized views and indexes to answer queries. In this paper we study the problem of selecting views for aggregate queries on star-schema data warehouses. The setting and assumptions we use generalize those in [19, 21] (in contrast to those in [39]) — that is, we seek to minimize the total execution costs of the given set of queries under a storage-limit constraint. At the same time, the novelty of our work is in obtaining efficiently optimal solutions for problem instances of realistic sizes, for two versions of the view-selection problem. In the first version, we assume similarly to [21] that the raw-data view is always part of the solution set of materialized views. In the second version of view selection, we lift this restriction; the resulting problem arises in practice in settings where it is too expensive to maintain efficiently the result of joining all base tables, including data-integration settings where certain views are materialized in the mediator to improve query-processing efficiency [20]. In our current work we are using lower-bound relaxation in developing competitive heuristics for problem instances that are too large to be solved optimally, for both versions of the view-selection problem.

2 Preliminaries and Problem Specification

The setting we use is similar to that in [19, 21]. We consider relational select-project-join queries with equality-based joins and with aggregation functions `sum`, `count`, `max`, or `min`. Our approach is applicable to queries with inequality comparisons of attributes to constant

values, including the important class of range-aggregate queries. We study *parameterized* queries: The parameterized version of a query has placeholders instead of all the constants. In this paper we focus on the special case of *star-schema queries*: We assume that the database schema is a star schema [23], with a fact table and dimension tables. Further, in each star-schema query, each join is a natural join of the fact table with a dimension table.

Our cost model is as follows. We consider the costs of answering queries using unindexed materialized views, such that each query can be evaluated using just one view relation and no other data, as in Example 1. (This setting is the same as in [21].) Thus, the cost of evaluating each query is proportional to the *size* of the view chosen for the evaluation. We use two metrics for view sizes: (1) the number of rows in the view relations (this is a common assumption in the literature on view selection), and (2) the number of bytes in the view relations. In Section 5 we will see that these metrics give us different experimental results. Finally, given a query workload $\mathcal{Q}$ and a set of views $\mathcal{V}$ that have been precomputed on a database $\mathcal{D}$, the *total cost* of evaluating $\mathcal{Q}$ using $\mathcal{V}$ is the sum of the costs of evaluating all the queries in $\mathcal{Q}$, such that each query is evaluated using a view in $\mathcal{V}$. The sum can be weighted to reflect the relative frequency or importance of individual queries. We consider two settings: In the first setting, similarly to [21] we assume that a view resulting from joining all the base relations in the star schema — we call it the *raw-data view* — is always part of the available set $\mathcal{V}$ of materialized views. We lift this assumption in the other setting considered in this paper.

We consider the following view-selection problem: Our goal is to minimize the evaluation costs of a given workload of parameterized aggregate queries defined on a star schema, by

selecting and precomputing views that can be used in answering the queries. (We consider only *equivalent* query rewritings, i.e., we require that exact answers to all the queries in $\mathcal{Q}$ can be computed using $\mathcal{V}$.) We consider this minimization problem under a storage-space limit, which is an upper bound on the amount of disk space that can be allocated for the views. Thus, our *problem inputs* are of the form $\mathcal{I} = (\mathcal{D}, \mathcal{Q}, b)$, where $\mathcal{D}$ is a database, $\mathcal{Q}$ is a workload of parameterized queries, and b is the (positive integer) value of the storage limit.

For any parameterized query in the given workload, our goal is to design views that can be used in evaluating *any instance* of the query. Thus, similarly to [19, 21] we consider only views without comparisons with constants. We use the following definitions of solutions and of the optimal viewset problem (OVP):

Definition 1. *For a problem input $\mathcal{I} = (\mathcal{D}, \mathcal{Q}, b)$, a set of views $\mathcal{V}$ is an* admissible viewset *if (1) each query in $\mathcal{Q}$ can be rewritten using $\mathcal{V}$, and (2) $\mathcal{V}$ satisfies the storage limit b.*

Definition 2. *For a problem input $\mathcal{I} = (\mathcal{D}, \mathcal{Q}, b)$, an* optimal viewset *is a set of views $\mathcal{V}$ defined on $\mathcal{D}$, such that (1) $\mathcal{V}$ is an admissible viewset for $\mathcal{I}$, and (2) $\mathcal{V}$ minimizes the cost of evaluating $\mathcal{Q}$ on the database $\mathcal{D}_\mathcal{V}$, among all admissible viewsets for $\mathcal{I}$. Here, $\mathcal{D}_\mathcal{V}$ is the database that results from adding to $\mathcal{D}$ the relations for all the views in $\mathcal{V}$ computed on $\mathcal{D}$.*

Definition 3. *For a given problem input $\mathcal{I} = (\mathcal{D}, \mathcal{Q}, b)$, find an optimal viewset. A* solution *for a given instance of OVP consists of a collection of materialized views $\mathcal{V}$ (which includes the raw-data view on $\mathcal{D}$ and all additional views that we choose to materialize) and an association between each element of $\mathcal{Q}$ and its corresponding element of $\mathcal{V}$.*

As we mentioned earlier in this section, we also consider a variation OVP' of the optimal viewset problem: Unlike OVP, in OVP' we do not require that the raw-data view be materialized as part of the solution. In Section 5 we present an experimental comparison of two versions of our proposed approach, one for OVP and the other for OVP'.

3 The Formal Model

In this section we propose an integer programming (IP) model for the optimal viewset problems OVP and OVP', and discuss methodologies for solving this IP model. We use the following notation to represent the input $\mathcal{I} = (\mathcal{D}, \mathcal{Q}, b)$ in this model:

a_i : Size of the view i, for all $i \in IV$,
where IV is the index set for all possible views;

b : storage limit;

c_{ij} : evaluation cost of answering query j by using view i,
for all $i \in IV$ and $j \in \mathcal{Q}$.

We let $c_{ij} = +\infty$ if view i cannot be used to answer query j. We further define the following decision variables for the IP model.

$$x_i = \begin{cases} 1 & \text{if view i is materialized} \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } i \in IV$$

and

$$y_{ij} = \begin{cases} 1 & \text{if we use view i to answer query j} \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } i \in IV \text{ and } j \in \mathcal{Q}$$

The optimal viewset problem OVP can now be stated as the following IP model $OVIP$.

Minimize	$\sum_{i \in IV} \sum_{j \in Q} c_{ij} y_{ij}$		$(OVIP)$
subject to	$\sum_{i \in IV} a_i x_i \leq b$		(1)
	$\sum_{i \in IV} y_{ij} = 1$	for all j	(2)
	$y_{ij} \leq x_i$	for all i, j such that $c_{ij} \neq +\infty$	(3)
	$x_1 = 1$		(4)
	$x_i, y_{ij} \in \{0, 1\}$	for all i, j	

Constraint (1) limits the size of the views to be no more than the storage space b. Constraint (2) states that each query is answered by exactly one view in the set of views. Constraint (3) guarantees that query j can be answered by view i only if view i is already materialized. Constraint (4) states that the raw-data view is always materialized, and the remaining constraints are simply the binary requirements for x_i and y_{ij}. Note that the binary requirement for y_{ij} can be replaced by a simple non-negativity restriction without affecting the corresponding optimal solution for this model. This relaxation, however, has a significant impact in reducing the overall computational effort required to solve this IP model.

We can also modify this model into an IP model for the problem OVP' (see note after Definition 3 in Section 2) by removing constraint (4). This modification is equivalent to stating that the raw-data view is not required to be materialized. We refer to this modified model, i.e., $OVIP$ without constraint (4), as $OVIP'$.

The structure of this IP model is similar to those for the uncapacitated facility location problem (UFL) and the k-median problem. These two problems are well studied in the literature, and relatively large instances of the corresponding IP models can be solved within reasonable time. Several heuristic approaches for solving these problems have also been reported. See [15] and [25] for the facility location problem and [27] for the k-median problem.

We can also employ the linear programming (LP) or the Lagrangean relaxation of this IP model to develop lower bounds for the optimum value of the objective function. In [33] it is observed that the LP relaxation of the IP model for the facility location problem can provide strong lower bounds for it, and in [28] it is shown that the Lagrangean relaxation of the IP model can provide even better lower bounds. Due to the similarity of the structure of $OVIP$ ($OVIP'$) with these models we expect that similarly strong lower bounds can be obtained for $OVIP$ ($OVIP'$). As these lower bounds are typically obtained with modest computational effort, they can be used to devise exact algorithms (such as branch and bound) for solving this problem. We can also employ the lower bound for each instance to evaluate the solution obtained via an inexact algorithm (i.e., a heuristic procedure), hence providing an upper bound on the performance ratio of the algorithm in that instance.

4 Our Framework

In our experimental results we consider view selection for star-schema queries. To design aggregate views for star-schema queries, we use a data structure, *view lattice*, that was introduced in [21]. A view lattice is a representation of the search space of views for the

workload, where nodes represent views and directed edges between the views denote which view can be evaluated using another view. For any view we choose to materialize from the view lattice, if the view is usable in evaluating some input query, then the answer to the view is the only relation needed in the evaluation. That is, our view-selection procedures determine *joinless* rewritings of queries. In addition to joinless rewritings, we plan to consider rewritings that are computed via joins of aggregate views with other relations [1, 14]; [2] describes our initial results in that direction.

To define an instance of the problem and to construct input data for our IP formalism, we first select a query workload, that is, frequent and important queries whose evaluation costs we want to reduce by materializing views. We then use the approach of [21] to construct a view lattice, by associating each workload query with a node in the lattice. More specifically, we construct from the query workload a set of grouping and aggregated attributes of interest — these are all the attributes mentioned in the queries, except the attributes in the join conditions. We then use the attributes to construct a view lattice as described in [21]. For instance, suppose queries `Q1` and `Q2` in Example 1 are our workload queries. We then use attributes mentioned in `Q1` and `Q2` to construct two sets of attributes of interest — grouping and aggregated attributes. Attributes `CID, Year, Month,` and `State` are our grouping attributes in Example 1, and attribute `QtySold` is the aggregated attribute.

Once we have constructed a view lattice, we calculate the sizes of the answers to all the views in the lattice. We can estimate the sizes by using methods mentioned in [21], for instance sampling. Finally, to complete the input data, we specify a storage limit.

To illustrate, we present an example adopted from [21]. Figure 2 shows a part of the view lattice for this example that consists of the raw-data view (source node a) and a collection of views $\{b, c, d, e, f, g, h\}$ as indicated. The space requirement for each node is given next to the node, and the edges represent the relationship between views as discussed above. We assume that the query workload consists of all nodes in the lattice, and the problem is to determine a set of at most three additional views to materialize (in addition to the raw-data view a) that would minimize the total cost of answering the workload queries. Note that to be consistent with the example given in [21] we restrict the *number of views*, rather than their corresponding storage space requirement.

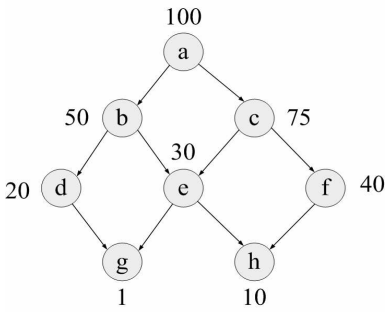

Fig. 1. Lattice example with space costs [21].

The IP model ($OVIP1$) for this example can be written as follows:

$$\begin{array}{lll}
\text{Minimize} & \sum_{i=1}^{8} \sum_{j=1}^{8} c_{ij} y_{ij} & (OVIP1) \\
\text{subject to} & \sum_{i=1}^{8} x_i \leq 4 & \\
& \sum_{i=1}^{8} y_{ij} = 1 & \text{for all } j = 1 \text{ to } 8 \\
& y_{ij} \leq x_i & \text{for all } i, j \text{ such that } c_{ij} \neq +\infty \\
& x_1 = 1 & \\
& x_i \in \{0, 1\},\ y_{ij} \geq 0 & \text{for all } i, j = 1 \text{ to } 8
\end{array}$$

The matrix of objective-function coefficients c_{ij} for the model $OVIP1$ is the following matrix, where nodes a, b, c, d, e, f, g, and h correspond to the rows (and columns) 1 through 8, respectively.

The model $OVIP1$ has 8 binary variables and 64 continuous variables. We solved this problem using the IP solver CPLEX [30] with an AMPL interface [16] and obtained the optimal solution $x_1 = x_2 = x_4 = x_6 = 1$ (corresponding to nodes a, b, d, f in the lattice), $y_{11} = y_{22} = y_{13} = y_{44} = y_{25} = y_{66} = y_{47} = y_{68} = 1$, with all other variables equal to zero. The total cost associated with this solution is 420. Note that this solution is identical to the solution obtained using the heuristic procedure of [21].

$$\begin{bmatrix} 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ \infty & 50 & \infty & 50 & 50 & \infty & 50 & 50 \\ \infty & \infty & 75 & \infty & 75 & 75 & 75 & 75 \\ \infty & \infty & \infty & 20 & \infty & \infty & 20 & \infty \\ \infty & \infty & \infty & \infty & 30 & \infty & 30 & 30 \\ \infty & \infty & \infty & \infty & \infty & 40 & \infty & 40 \\ \infty & \infty & \infty & \infty & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 10 \end{bmatrix}$$

Fig. 2. Cost matrix

To represent the lattice associated with a large data set (for a realistic instance of view selection), we use a table format. Each row in the table corresponds to a lattice node; the two entries in each row represent the *view ID* and the *view size* for that node, respectively. Thus, the table has two columns and as many rows as the number of lattice nodes.

In each row (i.e., node of the lattice) the view size is represented in units that we choose for our analysis (e.g., number of rows in the view, number of bytes of stored data in the view, etc.), and the view ID is a binary (0 and 1) vector of size K. The ith element in this vector is the ith grouping attribute, and K is the total number of grouping attributes. For each view, an entry of 1 in the ith position of its view ID implies that the corresponding attribute is used to group the associated rows in the database (to form this view). In other words, a 1 entry in the ith position of the view ID for a node (view) implies that this node (view) can be used to answer a query that requires the ith attribute, and a 0 entry in the ith position implies otherwise. We give SQL examples in Section 5; see Appendix B for a full

description of the problem input, view lattice, and solution for one instance of view selection on TPC-H data [37]. (In the table in Appendix B, view IDs are represented in decimal.)

It follows that the dependency relationship among views (nodes) can be derived from their corresponding view IDs. A query e in the lattice can be computed from a view f (i.e., f is an ancestor of e in the lattice) if the set of positions with entry 1 in the view ID for e is a subset of the respective set in the view ID for f (e.g., $f = \{1, 1, 0, 0, 1\}$ is an ancestor for $e = \{0, 1, 0, 0, 1\}$, but it is not an ancestor for $e' = \{0, 1, 0, 1, 0\}$). Note that the number of nodes in the lattice is 2^K and increases exponentially as the number of attributes K increases. For this reason, in order to keep the size of the IP model as small as possible, in each instance we only maintain those rows of the table (nodes of the lattice) that are potential ancestors of at least one of the queries in our query workload for that instance.

The evaluation cost of a query e using a view f is the storage cost of the view f if e can be answered by f, and is ∞ otherwise. Following the above criteria, the cost matrix for a query workload can be easily computed and transformed to the input of the IP model.

5 Experimental Evaluation of the Approach

We have conducted extensive experiments to evaluate the IP models and framework presented in Sections 3 and 4. (For each experiment type, we have solved many more instances of the problem than are described in this and next sections. For a report on all the instances, please see [6, 26].) All experiments were run on a machine with a 3GHz Intel P4 processor, 1GB RAM, and a 80GB hard drive running Windows XP SP2 and CPLEX/AMPL 9.0.

In this section we outline an experimental evaluation of our approach; in Section 6, we report experimental comparisons of our approach with the state-of-the-art algorithm of [21]. We did all the experiments on two datasets: (1) on a TPC-H database benchmark [37], and (2) on real data, using a modified Sloan Digital Sky Survey (SDSS) dataset [35]. For the TPC-H dataset, the sizes of the stored tables are shown in Figure 3. Size estimates for the lattice nodes were obtained by running the queries for all views on the TPC-H stored data with scale factor of 0.1 and by extrapolating the sizes of the answers to the sizes of the stored data (scale factor of 1) that were used in the experiments. For the SDSS dataset, whose sizes of the stored tables are shown in Figure 4, the view sizes were measured on the original database. The observations we made in our experiments were consistent across the two datasets; therefore, in reporting our results we will use examples just from the TPC-H dataset.

TPC-H Tables

Name	Size (bytes)
Lineitem	2,147,483,647
Part	1,193,906
Supplier	14,188,544
PartSupp	5,830,541
Customer	244,883,456
Orders	482,877,440
Nation1	2,103
Nation2	2,103
Region	396

Fig. 3. Sizes of TPC-H tables.

The goal of the experiments that we report in this section was to obtain optimal solutions and lower bounds on instances of $OVIP$ and $OVIP'$ of realistic sizes. The experimental results show the following:

SDSS Tables

Name	Size (bytes)
SpecObjAll	200,851,705
PhotoObj	38,206,539
Plate	69,035
Segment	621
Field	919
Neighbors	1,321

Fig. 4. Sizes of SDSS tables.

- relatively large instances of the view-selection problem, including instances of realistic sizes, can be solved optimally within reasonable execution time;
- the linear programming relaxation of the IP model provides a strong lower bound for the corresponding optimal value;

– regardless of whether we use the number of rows or the number of bytes for measuring the storage capacity, the IP model that we propose can be used equally easily to find the corresponding optimal solution of the view-selection problem, and in many instances the respective optimal solutions are not identical;

– in many instances the optimal solution for $OVIP'$ is different from the optimal solution for the corresponding $OVIP$, and the associated value of the objective function is significantly smaller.

The experimental results that we present in Section 6 further demonstrate that the overall computational effort required to solve an instance of the problem using our proposed integer programming model (i.e., the time required to find an optimal solution for a problem instance) is substantially smaller than that required to solve the same instance using the heuristic procedure proposed in [21] (i.e., to find a potentially sub-optimal solution for the same instance).

Ins. ID	No. of attr.	Maximum no. nodes	Query workload	Capacity no. rows	No. of nodes	No. of x_j's	No. of y_{ij}'s
1	7	128	{ 5, 7, 17, 69, 81, 88, 112 }	702,709	60	60	60 × 7
2	13	8,192	{ 88, 112, 593, 912, 2050, 2368, 6656, 7936 }	1,264,194	4,104	4,104	4,104 × 8
3	15	32,768	{ 152, 224, 2848, 3201, 8194, 8832, 26624, 31232 }	1,522,810	17,464	17,464	17,464 × 8

Table 1. Description of three problem instances in the experiments.

5.1 Solving the Problem Instances

For the experiments that we report here we used four TPC-H datasets — raw-data views with 7, 13, 15, and 17 attributes, respectively. (To obtain some raw-data views for the experiments, we used joins of TPC-H tables.) The numbers of nodes in the view lattices for these datasets are 128, 8192, 32768, and 131072, respectively.

For each raw-data view we constructed the IP model $OVIP$ for several instances of the problem, each instance with a different query workload and different storage limit b. For each instance, given the corresponding view lattice, query workload, and storage limit, we constructed the input files for the IP model, as described in Section 4; see Appendix B for an extended example. We solved each instance using the software package CPLEX/AMPL as described earlier, and in each instance on datasets with 7, 13, and 15 attributes we were able to find an optimal solution. For illustrative purposes, in Table 1 we give detailed characteristics of three different problem instances in our experiments. Each row of this table corresponds to one instance and gives the view lattice (raw-data view) and query workload corresponding to that instance. The maximum number of nodes in each instance is 2^K, where K is the number of attributes in the view lattice. Note that in our IP model we only include those nodes that could be used as potential ancestors for one or more queries in the query workload for that instance. Thus, the number of nodes we included in the IP model is in fact smaller than 2^K, as stated in the table. For each instance we also give the number of variables in the corresponding IP model.

We now explain how to read the indexes of queries and views in Table 1 and in the other tables in this and next sections. Recall that by construction of the view lattice, there is a one-to-one relationship between aggregate queries/views on the raw-data view and nodes in the view lattice. We order the grouping attributes of the raw-data view from left to right and index each node in the lattice using the decimal representation of the list of 1/0 bits that is constructed as described in Section 4. As the root node of the view lattice always has exactly the grouping attributes of the raw-data view, the decimal index of the root node is the decimal encoding of a list that has only 1's. To illustrate the assignment of indexes to queries and views in Tables 1 and 2, we give the definition of view 55 and one possible definition of query 55 (see instance 1 in Table 2) on the list of attributes `Orderkey, Suppkey, Partkey, Returnflag, Linestatus, Shipdate`, and `Discount` in the TPC-H [37] table `Lineitem`:

```
View55:  SELECT Suppkey, Partkey, Linestatus, Shipdate, Discount,
         SUM(Extendedprice) FROM Lineitem
         GROUP BY Suppkey, Partkey, Linestatus, Shipdate, Discount;

Query55: SELECT Suppkey, Partkey, Linestatus, SUM(Extendedprice)
         FROM Lineitem WHERE Discount < 0.2 AND Shipdate = '2005-08-01'
         GROUP BY Suppkey, Partkey, Linestatus;
```

Recall that each query is associated with a node that has as *grouping* attributes all attributes mentioned in the `GROUP BY` *and* `WHERE` clauses of the query. Thus, the grouping attributes `Discount` and `Shipdate` of `View55` appear in the `WHERE` clause in the given `Query55`. (The aggregated attribute `Extendedprice` in `View55` and `Query55` is not represented in the view lattice, similarly to [21].)

5.2 Execution time

The execution time for CPLEX/AMPL to solve the three instances in Table 1 was 0.12 seconds, 0.45 seconds, and 3.89 seconds, respectively. The execution time is expected to grow at an exponential rate with the size of the IP model; hence we do not expect it to be practical to solve much larger instances of this IP model using CPLEX/AMPL. At the same time, the instances that we are able to solve are of realistic sizes in practice, as exemplified in the three instances described in Table 1. This demonstrates that we can use a standard IP solver to solve practical instances of our proposed IP model.

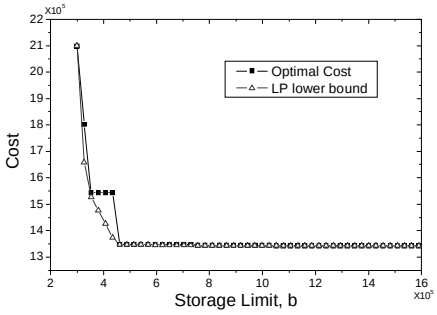

Fig. 5. Sensitivity analysis and LP lower bound for the view-7 instance.

We note that aside from the number of attributes and queries in the workload, the size of the IP model (and hence its execution time) also depends on the number of view-lattice nodes that are actually included in the model, i.e., number of nodes that can be used as potential ancestors for one or more workload queries. This number is usually larger if some workload queries are placed relatively low in the view lattice. Hence, the largest IP model in our experiment does not necessarily pertain to the view lattice with the largest number of attributes. In fact, the largest IP model that we have been able to solve optimally pertains to instance 3 in Table 1, which is on the 15-attribute lattice with 8 queries, and its execution time is 3.89 seconds. The largest instance of the OVP that we were able to solve, however, pertains to the 17-attribute lattice with 15 randomly generated queries, and its execution time is only 2.64 seconds, see [6]. The number of nodes included in the former IP model is

17,464 (see Table 1), while the number of nodes in the latter model is 13,755. Furthermore, the largest IP model does not necessarily pertain to the longest execution time, since the latter also depends on the growth of the search tree in the context of the branch and bound algorithm. In fact the longest execution time reported in our experiment pertains to an instance associated with the 7-attribute view lattice with 50 queries where the corresponding execution time is 13 seconds (instance 14 in Table 4.)

In many instances, while solving the IP model we also observed that the runtime of the CPLEX/AMPL for solving the model is only slightly larger than its runtime for solving the corresponding LP model. Intuitively, the lower bound obtained via the LP model is a relatively strong lower bound, hence it leads to quick fathoming of most branches in the corresponding branch-and-bound tree. This observation is not uncommon among other IP models with similar structures, such as the UFL problem that we mentioned earlier. Throughout our experiment, every time that the CPLEX/AMPL was not able to solve the IP model within our stated time limit, it was also unable to solve its corresponding LP model. We further report on the quality of this lower bound in the next subsection.

Note that the heuristic approach of [21] (see Section 6) was much less effective than the IP model in solving larger instances of the problem (i.e., it took much longer run time).

In our experiments we also studied whether the ancestor-descendant relationships between workload queries influence the runtime or outcome for our approach. In particular, we studied problem instances with "single-path" query workloads, that is, with workloads $\mathcal{Q}$ where for each query $Q \in \mathcal{Q}$, each remaining query in $\mathcal{Q}$ is either a descendant or an

ancestor of Q in the view lattice. It has been observed in [24] that workload queries with such relationships are likely to occur in query sessions by a single user in data warehouses, where the user performs roll-ups and drill-downs on the data of interest. In our study, we did not observe significant differences in the runtime or outcome for our approach depending on the type of relationships among workload queries. Rather, as mentioned earlier, we observed that the *placement* of individual queries in the view lattice has a bigger influence on how efficiently the problem instance can be solved by our approach.

5.3 Post-Optimality Analysis

We have performed a post-optimality analysis to observe the impact of changing the storage limit b on the optimal value of the objective cost function. In realistic view-selection scenarios, the total space available to store the materialized views is usually smaller than the total size of the input query workload; otherwise we can precompute all the queries in advance and store them on disk, which would be a globally optimal solution to the view-selection problem. At the same time, the storage limit has to be at least as large as the size of the raw-data view [21] (unless we relax the requirement that the raw-data view must be materialized, as in $OVIP'$). Hence, to explore the tradeoff between the amount of available storage space and the resulting total query costs, in our experiments we varied the value of storage space b between one and five times the size of the raw-data view. Figures 5 and 6 show the results for two instances of Table 1, with 7 and 13 attributes respectively.

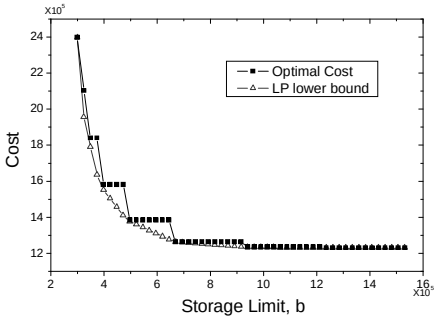

Fig. 6. Sensitivity analysis and LP lower bound for the view-13 instance.

In each instance and for each value of b we also solved the corresponding LP problem to obtain the associated lower bound; the lower bounds are also shown on the graphs. (The step curves in Figures 5 and 6 give the optimal cost value, whereas the smooth curves show the lower bounds.) Intuitively, the optimal cost changes only when the corresponding value of b increases sufficiently to allow for a different combination of materialized views to be the optimal solution. Hence, as we increase the value of b, we observe a stepwise decrease in the optimal cost only at those values of b that would allow sufficient capacity for a different (better) collection of materialized views. Note that if the storage capacity b is so limited that we can only store the raw-data view, then each query would be computed directly from the raw-data view; then as we increase the value of b the corresponding optimal query cost decreases monotonically. Finally, we observe that the LP lower bound is very close to the optimal value of the IP problem most of the time: Linear-programming relaxation provided a good lower bound in all the instances, and the ratio of the lower bound to the optimum varied between 0.92 and 0.99.

5.4 Measuring Table Sizes in Rows and Bytes

In another set of experiments we used bytes, rather than rows, to measure view sizes. In the literature, view sizes and query costs are typically measured in units of rows (see, e.g., [19, 21, 32]). At the same time, the units of bytes are the actual measure of storage requirements and query costs in query processing in database-management systems, because the cost of answering a given query using a given view is proportional to the number of disk blocks occupied by the view. In some of our experiments we expressed in bytes both the storage

requirements for the views and the costs of answering queries using those views. Note that if we state the problem this way for units of bytes, we do not need to change the formulation (equalities and constraints) of our IP model.

Inst-ance ID	No. of grouping attrib.	Root node index	Query workload (query indexes)	Units of rows		Units of bytes	
				storage limit	optimal viewset	storage limit	optimal viewset
1	7	127	{ 55, 59, 125, 126 }	899,418	{ 55, 126, 127 }	4,487,825	{ 55, 127 }
2	7	127	{ 1, 7, 53, 76, 111, 115 }	1,084,770	{ 1, 53, 76, 127 }	5,412,766	{ 1, 7, 76, 127 }
3	13	8,191	{ 1792, 3013, 5392, 6096, 7063 }	900,541	{ 1792, 5392, 8191 }	6,889,391	{ 5392, 6096, 8191 }
4	13	8,191	{ 1185, 5224, 6401, 6672 }	836,835	{ 6401, 6672, 8191 }	6,402,022	{ 6929, 8191 }

Table 2. Solving the problem for units of rows and bytes.

In our experiments, for some problem instances we obtain identical optimal solutions when view sizes and query costs are measured in rows *and* bytes; for other instances, however, we obtained different results. In Table 2 we report some results where we obtained different optimal solutions when we used units of rows versus units of bytes. The table shows experimental results for two problem instances on the view lattice with seven grouping attributes (instance IDs 1 and 2), and for two problem instances on the view lattice with thirteen grouping attributes (instance IDs 3 and 4); the raw-data view for each lattice comes from the TPC-H dataset [37]. For each instance we report the index of the root node of the

lattice; the root node is the raw-data view, which is (similarly to [21]) always required to be part of the viewsets we output as solutions for the problem instances of $OVIP$.

For each of the four problem instances we did two experiments — one with the storage limit and all view sizes measured in rows, and the other with the storage limit and view sizes measured in bytes. For example, the second row of Table 2 gives results for two experiments for instance ID 2 — that is, for query workload $\{1, 7, 53, 76, 111, 115\}$. One experiment was done for the value of storage limit b = 1,084,770 rows, and the other was done for a storage limit b = 5,412,766 bytes. (We set the value of b in units of rows and in units of bytes in such a manner that the instances are comparable.)

Our main observation on the results reported in Table 2 is that regardless of the units of measurement employed (rows or bytes), the IP model that we propose can be used to obtain an optimal solution for the problem within a reasonable amount of execution time (less than 20 seconds for the instances reported earlier), and using the units of bytes in this context does not impose any additional computational burden for solving the IP model. Further, the two optimal solutions obtained when we use these units of measurement are not necessarily identical. As units of bytes (instead of rows) is a more realistic measure in the context of view selection, we posit that these units should be employed as the primary units of measurement in problem inputs.

5.5 Solving the $OVIP'$ problem

We now outline our experimental results for the $OVIP'$ version of the problem, where the top raw-data view in a view lattice is not required to be materialized. This version of the

problem can arise in, for instance, data warehouses with form interfaces, where the total number of *parameterized* queries that a user can ask is finite and fixed by the interface. Another application is situations where it may be too expensive to maintain the raw-data view relation, and it is desirable to explore solutions which do not force that relation to be materialized; these situations include the data-integration settings where certain views are materialized in the mediator to improve query-processing efficiency [20]. (Recall that in many cases, the top raw-data view in the view lattices of [21] is the result of joining several base relations in the star schema.) Note that even though in this paper we compare our approach to the heuristic approach $HHRU$ of [21] (see Section 6), we could not do the comparison in this setting as the $HHRU$ approach requires that the raw-data view be always materialized. Clearly the optimal cost for $OVIP'$ can be smaller than the optimal cost for the corresponding $OVIP$ since we are no longer forced to materialize the raw-data view.

We report here some representative problem instances and solutions for a 7-attribute TPC-H dataset, with relation sizes measured in rows. In Table 3, the value of the storage limit for each problem instance is the sum of the size of the raw-data view with .67 the sum of the sizes of the answers to the input queries. By "optimal cost" (columns 6 and 9 in Table 3) we denote the total cost of evaluating the input queries using the optimal solutions we obtained for these problem instances.

For example, in instance 2 in Table 3 we solved the problem for six queries, with IDs $\{40, 50, 52, 88, 104, 112\}$. The storage limit for both the $OVIP$ and $OVIP'$ versions of the problem was set to $370,977$ tuples. The $OVIP$ version of the problem output six views

Instance ID	No. of workload queries	Storage limit	$OVIP$			$OVIP'$		
			no. of views	space used	optimal cost	no. of views	space used	optimal cost
1	5	1,112,258	4	914,119	1,213,067	4	913,664	1,212,612
2	6	370,977	6	364,837	364,837	6	106,220	106,220
3	8	405,781	5	385,411	171,804	6	113,156	158,166
4	8	362,002	8	360,985	360,985	8	92,823	92,823

Table 3. Solving the $OVIP'$ problem.

with IDs $\{40, 50, 52, 127, 104, 112\}$; the first view (ID 40) is used to answer the first query (ID 40), the second view (ID 50) is used to answer the second query (ID 50) and so on. Note that the raw-data view 127 is used to answer query 88. The resulting storage-space requirement for $OVIP$ is $364,837$ tuples, and the solution cost is also $364,837$. (The storage-space requirement for the solution is the same as the solution cost because each view is used to answer a distinct workload query.) In the $OVIP'$ version of the problem, we obtained six views with IDs $\{40, 50, 52, 120, 104, 112\}$. The resulting storage-space requirement and solution cost for $OVIP'$ are both $106,220$. (This solution for $OVIP'$, while not identical to the input query workload, is still globally optimal for this problem instance, because the query-evaluation costs of both solutions are identical on the problem inputs.)

In Table 3 we see that the solution cost is consistently lower for the $OVIP'$ version of the problem (column 9) than for the $OVIP$ version (column 6) — even when the number of materialized views is greater in the $OVIP'$ version (see instance 3 in the Table, columns 4 and 7). The storage space actually used by the solutions (columns 5 and 8) gives an intuition

for why the costs are the way they are — recall that the cost of answering each workload query is the size, in rows, of the smallest-size view that can be used to answer the query.

6 Comparing $OVIP$ to the Leading Heuristic Procedure

In order to further evaluate our proposed approach for solving the view-selection problem, we have experimentally compared the effectiveness of the approach with that of a leading heuristic procedure in the field. This heuristic procedure, which is based on the principle of greedy construction, is proposed in Harinarayan et al. [21] and throughout this section we refer to it as procedure $HHRU$; see Appendix A for the details of $HHRU$. In this section we discuss our experimental study and present the findings. Briefly stated, our empirical observations are as follows:

- The execution time required for solving $OVIP$ is generally smaller than the execution time required by $HHRU$; this implies that we can solve larger instances of the problem via $OVIP$ than we can via $HHRU$.
- In some instances the quality of the solution obtained via $HHRU$ is comparable to (either close to or identical with) the optimal solution obtained via $OVIP$. However, there are also some instances in which the quality of the solution obtained via $HHRU$ is far from the optimal solution.

In this section we also demonstrate that in general the value of the solution obtained via $HHRU$ can be significantly larger than the optimal value. To this end we present a family of instances of the view selection problem for which the performance ratio for the heuristic

procedure (defined as the value of the solution obtained via $HHRU$ divided by the optimal value) can be arbitrarily large.

6.1 Computational results and empirical observations

In order to carry out this comparative study we developed an independent computer program for the heuristic procedure $HHRU$. This program is written in C and run on the same machine that we used for solving the $OVIP$ models, as described in Section 5. (The pseudocode for $HHRU$ is reproduced from [21] in Appendix A.)

The data sets that we used to carry out the experiments were the same as those described in the previous section. Initially we used the number of rows to measure the view sizes, and all observations that we report in this section pertain to these instances. Later we ran a similar experiment in which we used the number of bytes to measure the view sizes. We noticed that the pattern of our observations did not change in any significant manner, except that when we used the number of bytes to measure the view sizes, in those instances where $HHRU$ did not find the optimal solution, the difference between the value obtained by $HHRU$ and the optimal value was slightly larger (at the order of 1% or 2%). In all instances that we report, unless specifically mentioned otherwise, we set the value of the storage limit to the size of the top raw-data view plus half the sum of sizes of the answers to the input queries.

On the TPC-H dataset, we solved instances with 7, 13, 15, and 17 attribute lattices. The results that we present below primarily pertain to the 7-attribute lattice. Our observations with the 13-attribute and 15-attribute lattices were similar to those for the 7-attribute lattice, except that the execution time was generally larger. In the case of 17-attribute lattice,

however, for most instances that we attempted to solve via the IP model the corresponding execution time was larger than the 15 minute time limit that we imposed, hence we terminated the algorithm prior to its successful completion. Only in a few such instances the algorithm terminated successfully within the imposed time limit, and in all these instances it actually terminated very quickly (within 3 seconds.) In comparison, when we attempted to solve a similar set of instances of the problem on the 17-attribute lattice via $HHRU$ we did not succeed to solve even one such instance, due to its excessive computation time.

Within each lattice we constructed and solved several problem instances using both the $OVIP$ model and the $HHRU$ procedure. Each instance has a different number of input queries, and we select the queries either *randomly* or *systemically* from among the set of all nodes in the lattice (e.g., among all 128 nodes in the 7-attribute lattice). The set of instances where we selected the input queries systemically includes the following special cases:

- the input queries have an ancestor-descendant relationship with each other;
- the input queries form a single path, i.e., for each query Q in the list of input queries, each remaining query in the list is either a descendant or an ancestor of Q;
- the list of input queries consists of the entire collection of views (nodes) in the lattice.

Results for 18 representative instances within the 7-attribute lattice are reported in Table 4. Results for other instances within the lattice of size 7, as well as for the instances in other lattices, are similar to those presented here. For each instance (row) in Table 4 we report the number of queries in the input query workload, the execution time required for solving the corresponding $OVIP$ model, the execution time required by the $HHRU$ proce-

dure, the optimal value obtained via the $OVIP$ model, and the value of the solution obtained via the $HHRU$ procedure. (By "value" we mean the total cost of evaluating the workload queries using the solution output by either procedure.)

Based on this data we make two observations. First, the execution time of $HHRU$ is generally larger than the time required for solving the corresponding $OVIP$ model. With the exception of the fourteenth instance, the execution time for solving the $OVIP$ model is always less than one second, but the execution time for $HHRU$ is usually larger, and it varies in an unpredictable manner: For some instances the execution time is relatively small, but in others it is quite large. Following is a possible explanation for this behavior of the heuristic procedure which, we believe, is consistent with its features. $HHRU$ selects one materialized view in each iteration in a greedy manner, and terminates as soon as the total size of all materialized views becomes close to the storage capacity b (so that no other view would fit). If at an early iteration a relatively large-size view is selected, the procedure would terminate sooner than the case where all selected views in the early iterations are relatively small in size. As a result, the number of input queries has a relatively low impact on the execution time of the procedure, while the relative size of the storage capacity and the choice (i.e., size) of the materialized view could play a more significant role.

Our second observation is that for this collection of instances, the quality of the solutions obtained via the $HHRU$ procedure is relatively good. In 13 of the 18 instances $HHRU$ finds an optimal solution, while in the remaining 5 instances the value of the solution found by $HHRU$ is within a fraction of percentage of the optimal value. But this characteristic is

not shared among all instances that we solved. In some instances the quality of the solution obtained via $HHRU$ was in fact far from optimal, as we discuss later in this section.

With respect to the execution time, we observed a similar behavior when we solved numerous other instances of the problem, either on the same (7-attribute) lattice or on 13- or 15-attribute lattices. For the instances on the 13-attribute lattice, the execution time for solving the corresponding $OVIP$ model ranged from 0.05 seconds (for an instance with 2 input queries) to 0.59 seconds (for an instance with 20 input queries), while the execution time for $HHRU$ on the same collection of instances ranged from 2 seconds (for an instance with 2 input queries) to 265 seconds (for an instance with only 10 input queries). Comparable numbers when we used $OVIP$ to solve randomly generated instances on the 15-attribute lattice are from 0.05 seconds (for an instance with 2 input queries) to 3.52 seconds (for an instance with 20 input queries). On the same lattice, when we employed $HHRU$, in took 597 seconds to solve an instance with only 2 input queries. For obvious reasons we did not attempt to solve larger instances with $HHRU$. The largest instance of the problem that we solved using $OVIP$ is on a 17-attibute lattice (with 15 input queries), and it took about 2.64 seconds. In general, among all instances in our experiment, the overall computational requirements for solving the $OVIP$ model is significantly less than those for the $HHRU$ procedure. As a result we are able to solve larger instances of the problem using our proposed integer programming approach.

With respect to the quality of solution obtained via $HHRU$ we had mixed observations, while, of course, the solution obtained via $OVIP$ was consistently optimal in every instance.

Similar to the results that we presented above, we observed in many other instances that the value of the solution obtained via $HHRU$ is either identical to the optimal value or very close to it. But this behavior was not consistent among all instances. Indeed in one instance on the 7-attribute lattice the value of the solution obtained via $HHRU$ (i.e., $v(HHRU)$) was as much as 584 folds larger than the corresponding optimal value (i.e., $v(opt)$) obtained via $OVIP$. See Table 5 for detailed information regarding several instances for the 7-attribute lattice where the $HHRU$ procedure performed poorly.

Upon further scrutiny we noticed that all instances in our experiment in which $HHRU$ performed poorly had a common feature in their input data. In all these instances there were two or more input queries that shared a common parent. This led us to construct a general family of instances for which we are able to demonstrate that the $HHRU$ procedure always performs poorly relative to the globally optimal solution returned by $OVIP$. We present this family of instances in the next subsection.

6.2 A family of instances with special structure

We present a family of instances of the view-selection problem for which we can demonstrate that the quality of the solution returned by the $HHRU$ algorithm would always be highly suboptimal. Consider a k-attribute lattice and suppose the list of input queries includes two queries (nodes) where each query is grouped by one attribute only; further let us assume that these two nodes have a common parent node. We refer to these two queries as queries $q1$ and $q2$, respectively, and to their common parent as query p. Let c_1 and c_2 denote the sizes of $q1$ and $q2$, respectively, and let c_p denote the size of their common parent. Also let c_{top} denote

the size of the top raw-data view. For convenience of presentation we assume that queries $q1$ and $q2$ are the only required (input) queries, although this assumption can be easily lifted with minor adjustments (i.e., the list of input queries can include other queries as well.)

We assume that the values c_1, c_2, c_p, and c_{top} satisfy the following relationships:

$$2(c_{top} - c_p) > c_{top} - c_1 \quad (5)$$

$$2(c_{top} - c_p) > c_{top} - c_2 \quad (6)$$

$$c_p > c_1 + c_2 \quad (7)$$

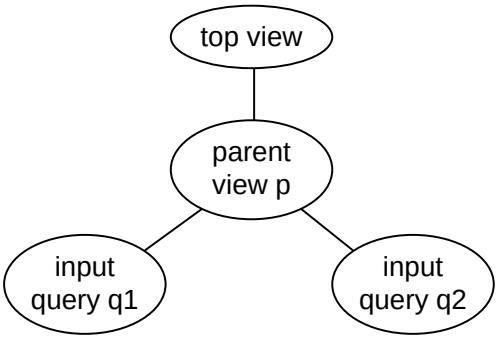

Fig. 7. Queries with special structure.

The storage capacity (limit) is $b = c_{top} + c_p$.

Under these restrictions, $HHRU$ would select to materialize the view (node) p before selecting the node for either $q1$ or $q2$, due to relationships (5) and (6). As soon as node p is selected, the storage limit ($c_{top} + c_p$) is reached, hence the procedure terminates. The corresponding value of this solution is $v(HHRU) = 2c_p$, as both queries $q1$ and $q2$ would be evaluated using the materialized view p.

From the relationship (7) it follows that the optimal solution in this case is to select nodes $q1$ and $q2$, with a total value of $v(opt) = (c_1 + c_2)$. The corresponding value of the performance ratio (i.e., the value of the solution obtained via the heuristic procedure divided by the optimal value) is

$$\rho = v(HHRU)/v(opt) = 2c_p/(c_1 + c_2).$$

Clearly we can select the values of c_1, c_2, and c_p in such a manner that this ratio is arbitrarily large. Note that relations (5) and (6) are equivalent to $c_{top} > 2c_p - c_1$ and $c_{top} > 2c_p - c_2$, respectively. This implies that we may select the values of c_1, c_2, c_p, and c_{top} in such a manner that they satisfy all above requirements while making the value of ρ as large as desired.

This family of instances demonstrates that the value of the solution obtained via $HHRU$ may be far from the optimal value, while the solution obtained via the $OVIP$ model is always guaranteed to be optimal.

7 Conclusions and Future Work

In this paper we considered the following view-selection problem: Given a set of queries, a database, and an upper bound on the amount of disk space that can be used to store materialized views, return definitions of views that, when materialized in the database, would satisfy the constraints and reduce the evaluation costs of the queries. We focused on practically important range-aggregate queries on star-schema data warehouses. We described our approach to obtaining *globally optimal* sets of views. The approach is an IP model that allows us to obtain optimal solutions without having to exhaustively enumerate all possible candidate solutions. We presented the formulation of the IP model and introduced an LP relaxation. We reported the results of our extensive experiments; the results show the practicality of our approach for problem instances of realistic sizes. We experimentally compared

our method with a leading heuristic approach in the field [21], and provided a theoretical analysis of the worst-case performance of that approach.

Our experiments show that the computational requirements of solving the $OVIP$ ($OVIP'$) problem (see Section 3) become prohibitive once the size of the problem exceeds certain limits. Hence, to solve larger instances, we are investigating techniques for designing and developing an algorithm (and the corresponding software) that takes advantage of the special structure of the problem. Further, solving even larger instances of the problem using exact methods might prove to be altogether too time consuming; thus, we may have to employ an appropriate heuristic procedure that exploits the structure of $OVIP$ ($OVIP'$), such as a Lagrangean heuristic. Such heuristic procedures have been developed for the facility location problem [8] and for the k-median problem [27], and the computational results show that these procedures obtain good solutions with a modest amount of computational effort. We expect that similar heuristic procedures can be developed for solving the view-selection problem as well. We are going to study how our current and planned procedures compare to approaches in the literature, including that of [32].

In addition to designing competitive heuristics for selecting views, we are extending our approach to selecting indexes alongside views (see, e.g., the setting of [19]). We plan to apply and extend our results to generalizations of range-aggregate queries, where queries can be answered using *joins* of views [1, 14]. We are also interested in studying the view- and index-selection problems under the maintenance-cost constraint on materialized views and indexes.

Acknowledgments

We are grateful to Anastassia Ailamaki's team for providing an adapted version of the SDSS dataset [35]. Many thanks to Shalu Gupta and Jingni Li for their help in preparing the datasets and assisting in the experiments.

References

1. F. Afrati and R. Chirkova. Selecting and using views to compute aggregate queries. In *Proc. ICDT*, 2005.
2. F. Afrati, R. Chirkova, S. Gupta, and C. Loftis. Designing and using views to improve performance of aggregate queries. In *Proc. 10th Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, 2005.
3. S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. VLDB*, pages 506–521, 1996.
4. S. Agrawal, S. Chaudhuri, and V.R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *Proc. VLDB*, pages 496–505, 2000.
5. S. Agrawal, S. Chaudhuri, and V.R. Narasayya. Materialized view and index selection tool for Microsoft SQL Server 2000. In *Proc. ACM SIGMOD*, 2001.
6. Z. Asgharzadeh Talebi, R. Chirkova, and Y. Fathi. Experimental study of an IP model for the view selection problem. Technical Report TR-2005-34, NC State University, July 2005. Available from `http://www4.ncsu.edu/~rychirko/Papers/techReport072005.pdf`.
7. Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized view selection in a multidimensional database. In *Proc. VLDB*, pages 156–165, 1997.
8. J. Barcelo and J. Casanovas. A heuristic Lagrangean algorithm for the capacitated plant location problem. *European J. Operations Research*, 15:212–226, 1984.
9. S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, 1997.
10. S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proc. ICDE*, pages 190–200, 1995.

11. S. Chaudhuri and V.R. Narasayya. An efficient cost-driven index selection tool for Microsoft SQL server. In *Proc. VLDB*, pages 146–155, 1997.

12. S. Chaudhuri and V.R. Narasayya. AutoAdmin 'What-if' index analysis utility. In *Proc. ACM SIGMOD*, 1998.

13. R. Chirkova, A.Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. *VLDB Journal*, 11(3):216–237, 2002.

14. S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. PODS*, 1999.

15. G. Cornuejols, G.L. Nemhauser, and L.A. Wolsey. The uncapacitated facility location problem. Technical Report 605, Operations Research and Industrial Engineering, Cornell University, 1984.

16. Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Boyd and Fraser, Danvers, Mass., 2002.

17. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, and M. Venkatrao. Data cube: A relational aggregation operator generalizing Group-by, Cross-Tab, and Sub Totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

18. A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *Proc. VLDB*, pages 358–369, 1995.

19. H. Gupta, V. Harinarayan, A. Rajaraman, and J.D. Ullman. Index selection for OLAP. In *Proc. ICDE*, 1997.

20. Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

21. V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. In *Proc. ACM SIGMOD*, pages 205–216, 1996.

22. IBM. Autonomic Computing. `http://www.research.ibm.com/autonomic/`.

23. R. Kimball and M. Ross. *The Data Warehouse Toolkit (second edition).* Wiley Computer Publishing, 2002.

24. Yannis Kotidis and Nick Roussopoulos. Dynamat: A dynamic view management system for data warehouses. In *Proc. ACM SIGMOD*, pages 371–382, 1999.

25. J. Krarup and P.M. Pruzan. The simple plant location problem: Survey and synthesis. *European Journal of Operations Research*, 12:36–81, 1983.

26. J. Li, R. Chirkova, and Y. Fathi. An IP model for the view selection problem. Technical report, NC State University, February 2005.

27. John M. Mulvey and Harlan P. Crowder. Cluster analysis: An application of lagrangian relaxation. *Management Science*, 25:329–340, 1979.

28. R. G. Parker and R.L. Rardin. *Discrete Optimization*. Academic Press, 1988.

29. Microsoft Research AutoAdmin Project. Self-Tuning and Self-Administering Databases. `http://research.microsoft.com/dmx/autoadmin/default.asp`.

30. ILOG S.A. CPLEX 7.0 software package. `http://www.ilog.com`, 2000.

31. Dennis Shasha and Philippe Bonnet. *Database Tuning: Principles, Experiments, and Troubleshooting Techniques.* Morgan Kaufmann, 2002.

32. A. Shukla, P. Deshpande, and J.F. Naughton. Materialized view selection for multidimensional datasets. In *Proc. VLDB*, pages 488–499, 1998.

33. K. Spielberg. Algorithms for the simple plant location problem with some side constraints. *Operations Research*, 17:85–111, 1969.

34. D. Srivastava, S. Dar, H.V. Jagadish, and A.Y. Levy. Answering queries with aggregation using views. In *Proc. VLDB*, pages 318–329, 1996.

35. A.S. Szalay, J. Gray, A.R. Thakar, P.Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg. The SDSS SkyServer - Public access to the Sloan Digital Sky Server data. Technical Report MSR-TR-2001-104, Microsoft Research, Microsoft Corporation, 2002. Available from `ftp://ftp.research.microsoft.com/pub/tr/tr-2001-104.pdf`; also see `www.sdss.org`.

36. Dimitri Theodoratos and Timos Sellis. Data warehouse configuration. In *Proc. VLDB*, 1997.

37. TPC-H:. TPC Benchmark H (Decision Support). `http://www.tpc. org/tpch/spec/tpch2.1.0.pdf`.

38. Jennifer Widom. Research problems in data warehousing. In *Proc. CIKM*, 1995.

39. J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proc. VLDB*, pages 136–145, 1997.

A The Greedy Algorithm of [21]

In this section we give an outline of the greedy algorithm proposed in [21], using mostly the wording of that paper. Recall that the algorithm has been developed for the version of the view-selection problem where the input query workload always includes all possible parameterized queries on a data cube — that is, all nodes in the view lattice of interest. In

this paper we use a straightforward extension of the approach to problem inputs whose query workload may be a proper subset of the set of all nodes of the view lattice, see Section 2 for the details. Note also that, similarly to our OVP problem but unlike our OVP' problem, the view-selection problem of [21] requires the raw-data view to always be part of the solution.

Suppose we are given a view lattice with space costs associated with each view; the space cost is defined as the number of rows in the view. Let $C(v)$ be the cost of view v. Suppose also that there is a limit S_{max} on the total size of views, in addition to the top view (which is the raw-data view, i.e., the view that has all the grouping attributes that define the lattice), that we may select. After selecting some set V of views (which surely includes the top view), the *benefit* of view v relative to V, which is denoted $B(v, V)$, is defined as follows. (The partial-order relation $\preccurlyeq$ denotes the descendant-ancestor relationship, namely which views in the lattice can be used to evaluate other views; thus, $x \preccurlyeq y$ stands for " view x can be evaluated using view y.")

1. For each $w \preccurlyeq v$, define the quantity B_w by:
   (a) Let u be the view of least cost in V such that $w \preccurlyeq u$. Note that since the top (raw-data) view is in V, there must be at least one such view in V.
   (b) If $C(v) < C(u)$, then $B_w = C(v) - C(u)$. Otherwise, $B_w = 0$.
2. Define $B(v, V) = \Sigma_{w \preccurlyeq v} B_w$.

In perhaps simpler terms, the benefit of v is computed by considering how it can improve the cost of evaluating views, including itself. For each view w that v covers, we compare the cost of evaluating w using v and using whatever view from S offered the cheapest way

of evaluating w. If v helps, i.e., the cost of v is less than the cost of its competitor, then the difference represents part of the benefit of selecting v as a materialized view. The total benefit $B(v, S)$ is the sum over all views w of the benefit of using v to evaluate w, provided that benefit is positive.

Now, we can define the *Greedy Algorithm* for selecting a set of views to materialize, such that the total size of materialized views, in addition to the top view, does not exceed S_{max}. The algorithm is as follows.

```
V = {top view};
S_{total} = 0;
while S_{total} < S_{max} begin
    select that view v not in V such that
        size(v) + S_{total} < S_{max}
        and B(v,V) is maximized;
    V = V union {v};
end;
resulting V is the greedy selection;
```

B Example of a Problem Instance and Solution

In this section we give an extended example of our experimental setting and results, using a problem instance $\mathcal{I}$ on the TPC-H dataset [37]; the instance is typical for our experimental data. The problem input in $\mathcal{I}$ uses seven attributes of the TPC-H tables, which are described

in Table 6; column `AttributeID` gives the encoding of each attribute in the IDs of queries and views. For instance, input query with ID 7 in $\mathcal{I}$ has attributes `OK` (ID 1), `PK` (ID 2), and `SK` (ID 4), because $1 + 2 + 4 = 7$.

The five input queries in the problem instance $\mathcal{I}$ have IDs 7, 23, 71, 87, and 119; the storage-space constraint is 7,870,005 bytes. We require the raw-data view to be materialized in the solution — that is, $\mathcal{I}$ is an instance of the OVP problem.

Table 7 gives information on all the views in the view lattice for the problem instance $\mathcal{I}$, one row per view. The lattice contains a view for each subset of the set of seven attributes `OK, PK, SK, DS, RF, LS, SD` in $\mathcal{I}$. Each view relation is computed as the result of grouping the join of all the TPC-H tables on the grouping attributes of the view. For example, the third row in Table 7 gives information on a view with ID 3; the view has two grouping attributes `OK` and `PK`, and the size of the view relation on the TPC-H dataset is 899,295 bytes.

An optimal solution to this problem is to materialize views $\{7, 23, 87, 127\}$:

- view 7 `(OK, PK, SK)` to answer query 7;
- view 23 `(OK, PK, SK, RF)` to answer query 23 `(OK, PK, SK, RF)`,
- view 87 `(OK, PK, SK, RF, SD)` to answer queries 87 and 71 `(OK, PK, SK, SD)`.

The last query, query 119 `(OK, PK, SK, RF, LS, SD)`, should be answered by the raw-data view 127 `(OK, PK, SK, DS, RF, LS, SD)`.

Instance ID	No. of workload queries	Time (sec.) $OVIP$	Time (sec.) $HHRU$	Optimal value $v(OVIP)$	Heuristic value $v(HHRU)$	Ratio $v(HHRU)/v(OVIP)$
1	2	0.03	40	300,314	300,314	1
2	3	0.02	< 1	565,830	565,830	1
3	4	0.02	4	405,052	405,052	1
4	5	0.04	1	925,292	925,292	1
5	6	0.03	1	1,469,930	1,469,934	1.000003
6	7	0.04	1	1,596,748	1,596,748	1
7	8	0.03	2	942,652	942,652	1
8	9	0.09	11	1,987,860	1,987,864	1.000002
9	10	0.07	2	2,043,735	2,043,735	1
10	12	0.06	1	3,242,040	3,242,040	1
11	15	0.03	1	3,875,900	3,875,903	1.000001
12	20	0.07	1	4,994,736	4,994,736	1
13	30	0.11	3	6,850,848	6,850,848	1
14	50	13	117	11,242,300	11,242,304	1
15	60	0.1	5	13,001,276	13,001,276	1
16	70	0.16	5	15,267,900	15,267,919	1.000001
17	100	0.22	5363	21,006,900	21,006,947	1.000002
18	127	0.23	59188	27,191,700	27,191,700	1

Table 4. Instances on a 7-attribute lattice on the TPC-H dataset.

Ins	query		Heuristic solution		Optimal solution		
tan	work-	Capa-	Mtrl'zed	$v(HHRU)$	Mtrl'zed	$v(opt)$	Ratio
ce	load	city	views	value	views	value	ρ
1	{4,8,16}	300328	{8,16,24}	299828	{4,8,16}	514	583.3
2	{4,8,32}	300327	{8,32,40}	299827	{4,8,32}	513	584.5
3	{8,16,32}	299858	{56}	132	{8,16,32}	16	8.25

Table 5. Some instances on a 7-attribute TPC-H lattice in which $HHRU$ performed poorly; root node 127 is included in all solutions; "ratio" $\rho = v(HHRU)/v(opt)$.

Attribute	AttributeID
orderkey (OK)	1
partkey (PK)	2
suppkey (SK)	4
discount (DS)	8
returnflag (RF)	16
linestatus (LS)	32
shipdate (SD)	64

Table 6. Seven attributes of TPC-H data used in the problem instance.

Views	ViewID	Size (bytes)
OK	1	150000
PK,	2	20000
OK, PK	3	899295
SK,	4	1000
OK, SK	5	895923
PK, SK	6	119922
OK, PK, SK	7	1199180
DS,	8	22
OK, DS	9	753180
PK, DS	10	308130
OK, PK, DS	11	1199236
SK, DS,	12	16500
OK, SK, DS	13	1198848
PK, SK, DS	14	869864
OK, PK, SK, DS	15	1499070
RF,	16	6
OK, RF	17	310647
PK, RF	18	89967
OK, PK, RF	19	1199088
SK, RF	20	4500
OK, SK, RF	21	1195772
PK, SK, RF	22	426200
OK, PK, SK, RF	23	1498995
DS, RF,	24	99
OK, DS, RF	25	1052284
PK, DS, RF	26	759440

Table 7. View sizes (in bytes) for the view lattice { `OK, PK, SK, DS, RF, LS, SD` }.

Views	ViewID	Size (bytes)
OK, PK, DS, RF	27	1499055
SK, DS, RF	28	66000
OK, SK, DS, RF	29	1498660
PK, SK, DS, RF	30	1323890
OK, PK, SK, DS, RF	31	1798884
LS	32	4
OK, LS	33	230781
PK, LS	34	60000
OK, PK, LS	35	1199060
SK, LS	36	3000
OK, SK, LS	37	1194660
PK, SK, LS	38	312496
OK, PK, SK, LS	39	1498975
DS, LS	40	66
OK, DS, LS	41	1007308
PK, DS, LS	42	652932
OK, PK, DS, LS	43	1499045
SK, DS, LS	44	44000
OK, SK, DS, LS	45	1498585
PK, SK, DS, LS	46	1270860
OK, PK, SK, DS, LS	47	1798884
RF, LS	48	12
OK, RF, LS	49	418168
PK, RF, LS	50	126608
OK, PK, RF, LS	51	1498860
SK, RF, LS	52	7908

Table 8. View sizes (in bytes) for the view lattice { `OK, PK, SK, DS, RF, LS, SD` }.

Views	ViewID	Size (bytes)
OK, SK, RF, LS	53	1494740
PK, SK, RF, LS	54	541390
OK, PK, SK, RF, LS	55	1798794
DS, RF, LS	56	176
OK, DS, RF, LS	57	1316620
PK, DS, RF, LS	58	956110
OK, PK, DS, RF, LS	59	1798866
SK, DS, RF, LS	60	90315
OK, SK, DS, RF, LS	61	1798404
PK, SK, DS, RF, LS	62	1591704
OK, PK, SK, DS, RF, LS	63	2098698
SD	64	5048
OK, SD	65	884745
PK, SD	66	893928
OK, PK, SD	67	1199252
SK, SD	68	797982
OK, SK, SD	69	1199224
PK, SK, SD	70	1197436
OK, PK, SK, SD	71	1499070
DS, SD	72	82677
OK, DS, SD	73	1197456
PK, DS, SD	74	1198672
OK, PK, DS, SD	75	1499070
SK, DS, SD	76	1186168
OK, SK, DS, SD	77	1499060
PK, SK, DS, SD	78	1498875

Table 9. View sizes (in bytes) for the view lattice { `OK, PK, SK, DS, RF, LS, SD` }.

Views	ViewID	Size (bytes)
OK, PK, SK, DS, SD	79	1798884
RF, SD	80	11439
OK, RF, SD	81	1184740
PK, RF, SD	82	1193600
OK, PK, RF, SD	83	1499065
SK, RF, SD	84	1096640
OK, SK, RF, SD	85	1499040
PK, SK, RF, SD	86	1497305
OK, PK, SK, RF, SD	87	1798884
DS, RF, SD	88	164788
OK, DS, RF, SD	89	1497430
PK, DS, RF, SD	90	1498500
OK, PK, DS, RF, SD	91	1798884
SK, DS, RF, SD	92	1486695
OK, SK, DS, RF, SD	93	1798872
PK, SK, DS, RF, SD	94	1798716
OK, PK, SK, DS, RF, SD	95	2098698
LS, SD	96	7572
OK, LS, SD	97	1179660
PK, LS, SD	98	1191904
OK, PK, LS, SD	99	1499065
SK, LS, SD	100	1063976
OK, SK, LS, SD	101	1499030
PK, SK, LS, SD	102	1496795
OK, PK, SK, LS, SD	103	1798884
DS, LS, SD	104	110236

Table 10. View sizes (in bytes) for the view lattice { `OK, PK, SK, DS, RF, LS, SD` }.

Views	ViewID	Size (bytes)
OK, DS, LS, SD	105	1496820
PK, DS, LS, SD	106	1498340
OK, PK, DS, LS, SD	107	1798884
SK, DS, LS, SD	108	1482710
OK, SK, DS, LS, SD	109	1798872
PK, SK, DS, LS, SD	110	1798650
OK, PK, SK, DS, LS, SD	111	2098698
RF, LS, SD	112	15252
OK, RF, LS, SD	113	1480925
PK, RF, LS, SD	114	1492000
OK, PK, RF, LS, SD	115	1798878
SK, RF, LS, SD	116	1370800
OK, SK, RF, LS, SD	117	1798848
PK, SK, RF, LS, SD	118	1796766
OK, PK, SK, RF, LS, SD	119	2098698
DS, RF, LS, SD	120	205985
OK, DS, RF, LS, SD	121	1796916
PK, DS, RF, LS, SD	122	1798200
OK, PK, DS, RF, LS, SD	123	2098698
SK, DS, RF, LS, SD	124	1784034
OK, SK, DS, RF, LS, SD	125	2098684
PK, SK, DS, RF, LS, SD	126	2098502
OK, PK, SK, DS, RF, LS, SD	127	2398512

Table 11. View sizes (in bytes) for the view lattice { `OK, PK, SK, DS, RF, LS, SD` }.