# Table of Contents

## EDITORIAL PREFACE
David Taniar, IJDWM Editor-in-Chief

## REVIEWED PAPERS

### 1 Statistical Sampling to Instantiate Materialized View Selection Problems in Data Warehouses
Mesbah U. Ahmed, University of Toledo, USA
Vikas Agrawal, Fayetteville State University, USA
Udayan Nandkeolyar, University of Toledo, USA
P. S. Sundararaghavan, University of Toledo, USA

In this article, a novel approach is used to instantiate and solve four versions of the MVS problem using three sampling techniques and two databases.

### 29 SeqPAM: A Sequence Clustering Algorithm for Web Personalization
Pradeep Kumar, University of Hyderabad, India
Raju S. Bapi, University of Hyderabad, India
P. Radha Krishna, Institute for Development and Research in Banking Technology, India

In this article, a similarity preserving function is introduced called sequence and set similarity measure $S^2M$ that captures both the order of occurrence of page visits as well as the content of pages.

### 54 A Hyper-Heuristic for Descriptive Rule Induction
Tho Hoan Pham, Hanoi University of Education, Vietnam
Tu Bao Ho, Japan Advanced Institute of Science and Technology, Japan

This article, proposes a hyper-heuristic to construct rule search heuristics for weighted covering algorithms that allows producing rules of desired generality.

### 67 GeoCache: A Cache for GML Geographical Data
Lionel Savary, PRiSM Laboratory, France
Georges Gardarin, PRiSM Laboratory, France
Karine Zeitouni, PRiSM Laboratory, France

To optimize spatial queries over GML encoded data, this article introduces a novel cache-based architecture. A new cache replacement policy is then proposed.
Statistical Sampling to Instantiate Materialized View Selection Problems in Data Warehouses

Mesbah U. Ahmed, University of Toledo, USA
Vikas Agrawal, Fayetteville State University, USA
Udayan Nandkeolyar, University of Toledo, USA
P. S. Sundararaghavan, University of Toledo, USA

ABSTRACT

In any online decision support system, the backbone is a data warehouse. In order to facilitate rapid response to complex business decision support queries, it is a common practice to materialize an appropriate set of the views at the data warehouse. However, it typically requires the solution of the Materialized View Selection (MVS) problem to select the right set of views to materialize in order to achieve a certain level of service given a limited amount of resource such as materialization time, storage space, or view maintenance time. Dynamic changes in the source data and the end users requirement necessitate rapid and repetitive instantiation and solution of the MVS problem. In an online decision support context, time is of the essence in finding acceptable solutions to this problem. In this chapter, we have used a novel approach to instantiate and solve four versions of the MVS problem using three sampling techniques and two databases. We compared these solutions with the optimal solutions corresponding to the actual problems. In our experimentation, we found that the sampling approach resulted in substantial savings in time while producing good solutions.

Keywords: data cube; data mart; data mining; data warehouse; materialized views; sampling

INTRODUCTION

The data warehouse is the heart of the architected environment, and is the foundation of all DSS processing (Inmon, 2002). Atypical data warehouse extracts the relevant information from many different operational databases into one centralized data repository to support business analysis activities and decision-making tasks (Haag, Cummings, & Phillips, 2006; Inmon, 2002). Recent extensions of this concept include building data warehouses to accommodate XML documents and facilitate easy querying of such data warehouses. Recent successful implementations of this concept may be found in Nassis, Rajagopalapillai,
Dillon, and Rahayu (2005) and Rusu, Rahayu, and Taniar (2005). Data warehouses and OLAP tools are based on a multidimensional data model. This model views data in the form of a data cube. The complete discussion on dimensional modeling can be found in Kimball and Ross (2002). There are many important architectural issues concerning the efficient design of a data warehouse. Data cube design is one such important aspect of the data warehouse architecture. In simple terms, a data cube is a multi-dimensional construct of data that lets us explore and analyze a collection of data from many different perspectives (Han & Kamber, 2001; Kimball, Reeves, Ross, & Thornthwaite, 1998). The data in the data cube may be aggregated in one or more dimensions to generate a view (also known as a cuboid). If such a view is stored physically in a storage device, it is called a materialized view.

The problem of quick and easy access to the summarized data at the data warehouse may be alleviated by an efficient selection of a set of materialized views. The general problem of selecting an appropriate set of views to materialize is called the Materialized View Selection (MVS) problem. In these problems, researchers attempt to achieve an objective such as minimizing the query response time given a limited amount of resource such as materialization time, storage space, etc. (Gupta & Mumick, 2005) Literature has reported several variants of the MVS problem obtained by different combinations of objective functions, resource constraints, and problem solving methodologies (Gupta et al., 2005; Harinarayan, Rajaraman, & Ullman, 1996, 1999). In order to apply any problem solving technique, the first step is to know the parameters of the problem instance. To completely specify an MVS problem instance, one must know the number of rows present in each view and the weight associated with each view in the data cube. The evaluation of weights is based on managerial discretion. However, determining the number of rows in a view may be a time consuming process, which may run into hours or even days.

There are many online applications, such as implementing a data cube within a commercial package environment, which may need to determine the views to be materialized before it can interact with the user (Jacobson, 2000). The set of materialized views may have to be altered dynamically to accommodate changes in the frequency and importance of incoming queries as well as the changes in the size of the base cuboid. This demands a quick method to generate an appropriate instance of the MVS problem based on current needs. Typically, this implies running several queries on a data warehouse to count the number of rows in each view. In order to address the need for generating problem instances quickly, we tested three statistical sampling techniques to estimate the actual number of rows present in each view (Figure 1). We then instantiate a pair of MVS problems using the estimated and the actual number of rows in each view. Next, we demonstrate the efficacy of the methods by comparing the solutions obtained by solving the two problem instances. Our experiments reveal that the sampling approaches to instantiate the MVS problems is an effective alternative to complete enumeration.

**Specific Contributions**

The highlight of the contribution of this chapter is to apply the concept of sampling-based row estimation in a relational table to estimate the size of the views in a data cube, and use it to instantiate and
solve several classes of MVS problems and report on the efficacy of the sampling method-solution procedure interface. This process could be used in online systems for speeding up response time in OLAP and other data warehouse applications. Finally, it may be pointed out that research hitherto treated view size estimation and MVS solution as two separate tracks, whereas, we have integrated the two and report on the interaction. Next, we turn to the specifics of our research.

We discuss the motivation for fast instantiation of MVS problems for improving decision support. We used three sampling techniques at three levels (5, 10, and 20%) on four types of MVS problems (weighted sum and weighted bottleneck) using two different lattices derived from real-world inspired databases to instantiate MVS problems. We used algorithms in the literature and solved several companion pairs of problem instances (one generated by sampling and the other generated by complete enumeration) using heuristics and integer programming formulation. A typical solution in this context refers to the views to be materialized. We evaluate both solutions over the actual problem instance generated by complete enumeration. Notice that the solution obtained by solving the instance generated by sampling method should not be evaluated over that instance for comparison purposes, since the solution will only face “real” data while being implemented and not “sampling” data. In most cases, we demonstrate that the heuristic solutions obtained by solving the problem over the instantiations obtained by sampling provided satisfactory results thus obviating the need for complete enumeration. Detailed comparisons across sampling methods and sample sizes are presented under experimentation and results section.

**ORGANIZATION OF THE CHAPTER**

The rest of the chapter is organized as follows. The next section is on conceptual background on the MVS problem including its variations and different formulations. It is followed by literature review of MVS problem and specific sampling techniques used in this chapter. Next, we present the methodology, describing the two lattices
used, sampling methods used, and a formal presentation of MVS problems and solution procedures. This is followed by a section titled Experiments and Results, which includes specific problems solved, experimental conditions, results, and an analysis. The last section presents conclusions and future research directions.

CONCEPTUAL BACKGROUND

Data Cubes and Materialized Views

Data cube is a multi-dimensional representation of data in which the cells contain measures (i.e., facts) and the edges represent data dimensions by which the data may be reported (sliced and diced). For example, a SALES cube can have measures “PROFIT” and “COMMISSION.” The dimensions can be TIME, PRODUCT, REGION, SALESPERSON, etc. Detailed discussions on data cube can be found in Han et al. (2001) and in Kimball et al. (1998). A data cube contains many views (or cuboids). In this context, a view is a set of aggregated data for a particular set of dimensions. Essentially a view is the result of a “GROUP BY” query. A materialized view is a pre-computed view that is physically stored for fast data retrieval purposes. Figure 2 presents a lattice diagram for a hypothetical data cube. In this lattice diagram, there are nine views. View zero (also called base cuboid or root view) corresponds to the view having the lowest level of data summarization and it is assumed to be always materialized. View eight contains the highest level of aggregated data and it will contain only one row. In between these two extremes, there are seven views, which represent different levels of aggregation. The number in parenthesis represents the number of rows present in that view. The arrows indicate the direction of aggregation. Typical queries posed on the system may be answered by using the root view or any of the appropriate materialized views. For example, in order to answer queries based on View five, one could use View five if it is materialized or any of its materialized ancestral views. The number of rows to be retrieved will be 100, 50, 70, or 40 rows, (respectively for using Views 0, 1, 2, or 5) depending on the materialized view used.

Motivation for MVS Problem

The motivation to materialize a view is to support fast decision making by answering questions posed to the system in an efficient manner using the materialized views rather than generating answers from the base cuboid using Group-By queries, which is typically time intensive. The next natural question is, why not materialize all possible views? Typical online systems have limits on storage and also impose high costs for updating and maintenance. The problem of quick and easy access to the data cube may be alleviated by an efficient selection of a set of views to be materialized. As discussed earlier, not all views in a data cube may be materialized due to constraints imposed on the system. Hence, selecting the right set of views to materialize is an integral part of the design of data cube and its associated views. An efficient design will dramatically reduce the execution time of decision support queries and hence prove pivotal in delivering competitive advantage.

Classical MVS Problem

Many researchers have studied the problem of selecting the “right” set of views to be materialized in a data cube in order to minimize decision support query response time subject to constraints on storage space.
The classical MVS problem, defined over large data cube, involves identifying the set of views to be materialized in order to minimize the total access time subject to constraints on materialization time, the number of views that can be materialized, and/or the amount of storage space available for view materialization (Gupta & Mumic, 2005; Harinarayan et al., 1996, 1999). The total access time is measured by the total number of rows to be retrieved to satisfy the needs of all the users of the system. The classical MVS problem uses a linear cost model with the following assumptions:

1. The cost of constructing a view from its materialized ancestor is a linear function of the number of rows in its materialized ancestor. It may be noted that the number of pages would be a better measure, but the number of rows has been used as a common measure in the literature.

2. If a view is materialized, its storage cost will be a linear function of the number of rows in the view.

3. Whenever a user (or an application) requests a view, the request is always for the entire view and not for any part of it.

4. The views are either stored or created from relational database tables.

**Variations of the MVS Problem**

An interesting variant of the classical MVS problem is to minimize the weighted number of rows to be retrieved, where each view is assigned a weight depending on the importance of the user group and the frequency of use (Gupta, 1999, 1997; Gupta et al., 2005; Harinarayan et al., 1996, 1999). Another interesting variation of the classical MVS problem is called bottleneck MVS problem. The motivation for this formulation is to improve the response time across the entire user spectrum, thus guaranteeing

---

*Figure 2. A simple lattice diagram with nine views (represented by nodes)*

Copyright © 2007, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
a level of quality of service for all users. In the bottleneck formulation, the objective is to minimize the maximum weighted number of rows to be retrieved to meet all user needs. (Agrawal, Nandkeolyar, Sundararaghavan, & Ahmed, 2004). Both variants are subjected to constraints similar to the classical MVS problem.

**Instantiation of MVS Problems**

The first step in solving an MVS problem is to create an instance of the problem. This requires specification of the number of rows in all possible views of the lattice structure corresponding to the data cube in question. A real-life version of the lattice of a data cube may involve a large number of nodes (views or cuboids), and the root view may have millions of rows. The number of rows in all other views will be less than or equal to that of the root view but not necessarily much less. We need to know the number of rows in each of these views in order to instantiate an MVS problem. This could potentially take several hours of computational time. Further, this is not just a one-time operation. For every major update in the source database, the number of rows in all views of a data cube may change. This will require the MVS problem to be re-instantiated and solved. The current online real-time decision support environment demands instantiating and solving MVS problems frequently on a real-time basis. Clearly, large delays will be unacceptable. Hence, we need a reliable and fast method to generate problem instances over which the appropriate type of MVS formulation could be solved. This brings us to the focus of our current research, which is to use the principles of statistical sampling to generate problem instances of the MVS problem on a real-time basis. For specific MVS formulations and sampling methods used for instantiation, please refer to the methodology section.

**LITERATURE REVIEW**

**The MVS Problem**

**Historical Overview**

Research interest in materialized views started in the early eighties. One of the early investigations was to speed up the data retrieval process for running queries on views in very large databases (Adiba & Lindsay, 1980). Subsequently, further research studies were reported in view and index maintenance along with comparative evaluations of materialized views on the performance of queries (Blakeley & Martin, 1990; Qian & Wiederhold, 1991; Segev & Fang, 1991). The MVS problem was formally studied by Harinarayan et al. (1996, 1999), where major features of the MVS problem are discussed elaborately. For a more recent review of literature on this problem, one may turn to Shah, Ramachandran, and Raghavan (2006) and Gupta et al. (2005). Typically, a lattice framework is used to capture the dependencies among views. The size of the views is an important component in the formulation of the MVS problem.

**View Selection**

Kalnis, Mamoulis, and Papadias (2002) have reported on a randomized local search algorithm to generate the “right” set of views to materialize and this approach is particularly useful in large dynamic view selection problems where the execution time for solving the materialized view selection problem is critical. Shah et al. (2006) report on a novel approach to the MVS problem by dichotomizing the set of views
into static and dynamic sets. They have
designed algorithms to select the “right”
set of static views that are most useful
for maintenance and update of potentially
useful dynamic views which are selected
based on query incidents.

Classical MVS Formulation and
Solution

Gupta et al. (2005) have proposed
a theoretical framework for the general
problem of selection of views in a data
warehouse. They have presented competitive
polynomial-time heuristics for a selection
of views to optimize total query response
time under a storage space constraint, for
some important special cases of the problem
that occur in practice. They have also ad-
dressed in detail the view selection problem
under the maintenance cost constraint and
presented provably competitive heuristics.
Theodoratos and Xu (2004) have pointed
out that solving the materialized view
selection problem requires generating a
space of candidate view sets from which an
optimal one is chosen for materialization.
They have suggested a novel approach,
which is based on adding to the alternative
evaluation plans of multiple queries views
called closest common derivators (CCDs),
and on rewriting the queries based on these
CCDs.

Several researchers have proposed a
genetic algorithm-based solution technique
for the materialized view selection problem
and have demonstrated that it is practical
and effective and produces comparable
performance (Horng, Chang, Liu, & Kao,
1999; Lin & Kuo, 2004; Zhang & Yang,
1999).

Shukla, Deshpande, and Naughton et
al. (1998) have proposed a simple and fast
heuristic algorithm called pick by size (PBS)
to solve the MVS problem and explored its
performance. They pointed out that PBS
runs several orders of magnitude faster
then the heuristic algorithm proposed by
Harinarayan et al. (1996) and is fast enough
to make the exploration of the time-space
tradeoff feasible during system configura-
tion. Furthermore, they have examined the
MVS problem when subsets of aggregates
can be computed using chunks (Deshpande,
Ramasamy, Shukla, & Naughton, 1998) and
showed that the benefit of the views selected
by PBS can be greater then the ones selected
without chunk based precomputation.

Baralis, Paraboschi, and Teniente
(1997) have pointed out that the number
of representative queries is extremely
small with respect to the total number of
elements of the complete data cube. Us-
ing such indications (inputs), they have
described the technique to select views and
an algorithm to perform selection that will
reduce the solution space by considering
only the relevant elements of the multi-
dimensional lattice.

Approaches to Improving Efficiency

Park, Kim, and Lee (2002) assume that
the set of materialized views is given and
then ask the question: How do we to rewrite
the given OLAP query to make the best use
of existing materialized views? They have
developed algorithms for the rewrite as well
as identifying the materialized views that
will best answer the query.

Gray et al. (1997) proposed the data
cube as a relational aggregation operator
generalizing group-by, cross-tabs, and sub-
totals. A taxonomy for recent advancements
in data cube queries may be found in Tan,
Tanier, and Lu (2004). Group-by queries
in a very large database take an enormous
amount of time. Dynamic view selection
problems are an important constituent for
supporting fast online queries on such da-
tabases. In order to solve MVS problems, one needs the sizes of the various views which are obtained from running group-by queries. Time required for running such queries can be reduced by an order of magnitude by running parallel group-by queries. For recent advances in parallel group-by queries one may refer to Tan, Taniar, and Lu (2003), Taniar and Rahayu (2006), Taniar and Tan (2002), and Taniar, Rahayu, and Ekononmida (2001), Taniar, Jiang, Liu, and Leung (2002), Taniar, Tan, Leung, and Liu (2004).

Sismanis and Roussopoulos (2004) have demonstrated that by factoring out certain redundancies from the structure of the cube, it is possible to compute, store and index the whole cube with 100% precision using only polynomial (and close to linear) space and time. They have also provided an algorithm for estimating the size of high-dimensional cubes based only on metadata about them. This work may be viewed as an alternative to the sampling based problem instance generation, which is the main topic of our chapter.

**MVS Formulation Based on Quality of Service**

The most common version of the MVS problem has the objective of minimizing the sum of weighted number of rows to be retrieved in responding to a given set of queries (Harinarayan et al., 1996, 1999). An interesting variant has the objective of minimizing the maximum weighted number of rows to be retrieved in responding to any query from the set of queries (Agrawal et al., 2004). This version of the MVS problem may be denoted as the bottleneck MVS problem, which provides a guaranteed quality of service to all users.

**Complexity and Need for Sampling**

Chirkova, Halevy, and Suciu et al. (2001) have pointed out that the complexity of the materialized view selection problem depends crucially on the quality of the estimates that a query optimizer has on the size of the views it is considering to materialize. They have shown that when a query optimizer has accurate size estimates of the views, the cardinality of an optimal view selection may be exponential in the size of the database schema. On the other hand, when optimizer uses standard estimation heuristics, they have shown that the cardinality of an optimal view selection is polynomially bounded. For very large databases, it is very time consuming to generate the actual sizes of the views. Sampling based procedures such as the ones discussed in this article would provide sufficiently accurate size estimates while limiting the time needed to run such queries.

For all versions of the MVS problem, it is necessary to instantiate problem instances using group-by queries, which typically take considerable amount of time while running in very large databases. Our work uses sampling methods to generate problem instances which can then be solved using methods in Harinarayan et al. (1996, 1999) and Agrawal et al. (2004), and hence these are the works closely relevant to our current chapter. In the next section, we briefly review the relevant literature on sampling methods, which helps us to generate problem instances. No published research to our knowledge develops the sampling based comparative solution approach as fully as we have done in this article, though several researchers such as Harinarayan et al. (1999), Shah et al. (2006), and other have recommended using sampling based ap-
proaches for solving MVS problems. Next, we survey the traditional sampling literature relevant to the MVS problem.

DISTINCT VALUE ESTIMATION AND ITS RELATION TO THE MVS PROBLEM

Overview
A sub-problem in instantiating the MVS problem is to find the number of rows in each of the views of a given lattice. This sub-problem is posed in the literature as the problem of estimating the number of distinct values (classes) in a database table. We address this literature first. One of the earliest works in the area of estimation of distinct values of an attribute in a database table (relation) is that of Flajolet and Martin (1985). They designed a fast and efficient algorithm to perform this job in a single pass through the data. For other sampling based approaches for the same problem, please refer to Whang, Vander-Zanden, and Taylor (1990).

Statistical Estimators
Comprehensive overviews of various statistical procedures to estimate the number of distinct values of an attribute in a given relation are available in Haas, Naughton, Seshadri, and Stokes (1995) and Haas and Stokes (1998). They have devised several new estimators including a hybrid estimator that appears to outperform the estimators developed in prior literature, both for real and synthetic data sets. This hybrid estimator first uses a chi-squared test to decide whether the data has low skew or high skew. Accordingly, it applies a smoothed jackknife estimator in the former case and the Shlosser estimator (Shlosser, 1981) in the latter case. Haas and Stokes (1998) have done an extensive study of several generalized jackknife estimators relating them to previously known estimators and proposing new ones.

Charikar, Chaudhuri, Motwani, and Narasayya (2000) have devised an estimator called Guaranteed-Error Estimator (GEE). They pointed out that the GEE performed well for data with high skew or with relatively few low frequency elements. Using the same logic, they pointed out that the GEE would underestimate in cases which has both low skew and a large number of distinct values. They, further pointed out that to ensure accuracy, the estimation procedure needs to take into account characteristics of the input distribution. They devised a new estimator called Adaptive Estimator (AE), which takes the input distribution into account.

The problem of estimating the number of rows present in each view in a data cube has been addressed by Shukla et al. (1996). They have provided an algorithm based on probabilistic counting which provides estimation within provable error bonds. But their approach still requires one full scan of the fact table in order to estimate the storage blowup that will result from a proposed set of precomputations without actually computing them. Our methodology needs a sample of size 10-20% of the fact table in order to estimate the number of rows present in a cube view. Our experimentation in fact points to the fact that even 5% sample size produce a good solution.

Relation to the MVS Problem
The problem addressed in all of the previously mentioned chapters finds direct application in instantiating the MVS problem, which is the primary focus of our work. In order to instantiate an MVS prob-
lem, we need to know the number of rows in each view of a data cube. This problem is identical to the problem of finding the number of distinct values of an attribute or group of attributes in a relation. However, effectiveness of the sampling procedures in instantiating the MVS problem can only be tested by comparing the solution obtained from the sample instance and the instance corresponding to the population (complete enumeration). This is the subject of our discussion in a later section.

**METHODOLOGY**

The objective of this chapter is to develop and test fast routines to instantiate, formulate, and solve appropriate versions of the MVS problem. This will facilitate dynamic and efficient support of large data warehouse based decision support systems. Three components are needed in this context: Data warehouse on which the MVS is applied, instantiation of the problem using sampling techniques, and appropriate solution procedure for the MVS problem.

**Data Warehouses Used in this Research**

For generating problem instances, we need a lattice structure of the corresponding data cube. In this research, we have constructed two 27-node lattices denoted as Lattice 1 and Lattice 2. Lattice 1 was derived from a 1-GB TPCH benchmark database and Lattice 2 was derived from a second synthetic database referred to as AAMS database. These lattices are discussed briefly below.

**Lattice 1**

The TPCH is a well-known transaction processing benchmark database generated by Transaction Processing Performance Council (http://www.tpc.org/tpch/spec/h130). Using their template, we populated a 1-GB TPCH Benchmark database. Then we populated the root node (base cuboid) of the data cube from this database using three dimensions (i.e., the Customer (C), the Part (P), and the Time (T) dimensions), and one measure of interest (i.e., the “Sale.”). Each dimension in this data cube has two levels, for example, the Part dimension has individual part and part type as two levels. For a given number of dimensions and levels in those dimensions, the number of views \( T \) is given by the formula

\[ T = \prod_{i=1}^{n} (L_i + 1) \]

where \( L_i \) is the number of levels associated with dimension \( i \) and \( n \) is the number of dimensions. This results in 27 views as shown in Figure 3 along with the actual number of rows in each view. While Lattice 1 is derived from a well-known transaction processing benchmark database, it had some distinct characteristics probably due to the size of the database created. For example, there are 12 nodes with over one million rows, and 10 nodes with less than a few thousand rows, and very few nodes with number of rows in between. In order to create another representative lattice, we designed and populated another synthetic database (i.e., AAMS database).

**Lattice 2**

In AAMS database, the daily transactions were created randomly from certain predefined probability distributions with the additional requirement that the number of rows in a descendent view would in general be significantly less than the lowest number of rows in all the immediate
ancestral views. The dimensions and levels used while populating the data cube were kept the same as those in Lattice 1, which ensured an identical structure. This lattice along with row counts is shown in Figure 4. It may be noted that the number of rows in different nodes of Lattice 2 is more evenly distributed as compared to Lattice 1.

Sampling Methods Used in this Research

The literature has reported several sampling-based techniques for estimating the number of distinct values in a relation. As noted earlier, this approach may be used to estimate the number of rows in a GROUP BY query from the root view. Further, only one random sample from the root view is needed in order to estimate the number of rows in all nodes of a lattice. The ability of a sampling technique in estimating the population size depends on the characteristics of the underlying population. Factors such as skewness and number of distinct values in the population may affect accuracy. Perusing the literature with this criterion, we selected three sampling techniques for our research (i.e., the Shlosser’s Estimator (SE))(Haas et al., 1995; Shlosser, 1981), the Guaranteed-Error Estimator (GEE) (Charikar et al., 2000), and the AE (Charikar et al., 2000). The details of these estimators are available in the respective referenced papers. In this section, we will
simply include the computational expressions that have been used to estimate these three statistics.

Let,

- \( n \) = number of rows in a relation,
- \( r \) = number of rows in a sample drawn from a relation,
- \( f_i \) = the number of attribute values that appear exactly \( i \) times in the sample of size \( r \).
- \( d \) = number of distinct values of an attribute or of a composite attribute that appears in the sample. It may be noted that \( d = \sum_{i=1}^{n} f_i \) and \( r = \sum_{i=1}^{n} f_i \) for \( 1 \leq i \leq r \).

\( \hat{D} \) = estimate for the number of distinct values of an attribute or a composite attribute that exists in the population.

Sanity bounds were incorporated into the estimators to ensure that \( d \leq \hat{D} \leq n \), i.e., if \( \hat{D} > n \), we set \( \hat{D} \) to \( n \) and if \( \hat{D} < d \), we set \( \hat{D} \) to \( d \) (Charikar et al., 2000).
Shlosser's Estimator (SE) (Haas et al., 1995; Shlosser, 1981)

\[
D_{\text{shlosser}} = d + \frac{\sum f_i (1-q)^{f_i}}{\sum iq(1-q)^{i-1}} f_i
\]

(1)

where, \( q = r/n \) (is the probability with which each tuple is included in the sample, independently of all other tuples.)

The Guaranteed-Error Estimator (GEE) (Charikar et al., 2000)

\[
D_{\text{GEE}} = (n \sqrt{r}) f_1 + \sum_{i=2}^{\infty} f_i
\]

(2)

The Adaptive Estimator (AE) (Charikar et al., 2000)

\[
m - f_1 - f_2 = f_1 \frac{\sum f_i (1+e^{c(1-\beta)})}{\sum f_i (1+2e^{c(1-\beta)})}
\]

(3)

\[
D_{\text{AE}} = d + m - f_1 - f_2
\]

(4)

where \( m \) is the number of low frequency values.

Please recall that our focus is on the efficacy and accuracy of the solution to the MVS problem as opposed to the accuracy of the sampling procedure itself (Figure 1).

MVS Problems Considered

As noted in Figure 1, the thrust of this research is to compare the solutions to the MVS problem instantiated using the actual number of rows and the estimated number of rows. Clearly, one expects to know the actual problem instance solved in order to appreciate and understand the impact of the sampling procedure on the final solution. Hence, we describe the MVS problems and solution procedures used to solve the problems addressed in this chapter.

General MVS Problems

- **Problem 1**: Given a data cube, the maximum number of views that can be materialized, and the weight associated with each view, determine the set of views to be materialized so as to minimize the total weighted number of rows to be retrieved in order to obtain each view in the data cube.

- **Heuristic 1**: Problem 1 was solved using a greedy approach presented in Harinarayan et al. (1999).

- **Problem 2**: It is the same as Problem 1, except that the constraint on the number of views to be materialized is replaced by the constraint on the number of rows that may be stored.

- **Heuristic 2**: This heuristic is an extension of Heuristic 1.

Bottleneck MVS Problems

In this section, we look at a bottleneck version of the MVS problem. The objective is to minimize the maximum weighted number of rows to be retrieved. This is a QOS-type measure, which minimizes the maximum time, thus guaranteeing certain level of service quality for all users. This measure also takes into account the relative importance of the various views. We consider two types of constraints—the total number of views that may be materialized, which leads to Problem 3 and the total amount of storage space (measured in rows) available to store the materialized views, leading to Problem 4.
Figure 5. Sampling size used to generate problem instances

<table>
<thead>
<tr>
<th>Lattice</th>
<th>SE</th>
<th>GEE</th>
<th>AE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lattice 1</td>
<td>20%  10%  5%</td>
<td>20%  10%  5%</td>
<td>20%  10%  5%</td>
</tr>
<tr>
<td>Lattice 2</td>
<td>20%  10%  5%</td>
<td>20%  10%  5%</td>
<td>20%  10%  5%</td>
</tr>
</tbody>
</table>

- **Problem 3**: Given a data cube, the maximum number of views that can be materialized, and the weight associated with each view, determine the set of views to be materialized so as to minimize the maximum weighted number of rows to be retrieved in order to obtain each view in the data cube.

- **Heuristic 3**: Details of the heuristic are given in (2004).

- **Problem 4**: It is the same as Problem 3, except that the constraint on the number of views to be materialized is replaced by the constraint on the number of rows that may be stored.

- **Heuristic 4**: Details of the heuristic are given in Agrawal et al. (2004)

### Optimal Solutions

We also instantiated Problems 1-4 with the actual number of rows. We then solved these instances optimally using integer linear programming formulations presented in Agrawal et al. (2004). Finally, we report on the efficacy of using the heuristics with the sampled data as well as the actual data to solve Problems 1-4 by comparing these solutions to the optimal solutions.

In the next section, we describe the details of the problems solved and the performance comparisons of sampling and complete enumeration.

### EXPERIMENTS AND RESULTS

There are six factors considered in this study: lattice, sample size, estimation method, problem type, problem parameters, and solution method. There are two types of lattice (Lattice 1, Lattice 2), three sample sizes (5%, 10%, 20%), three estimation methods (SE, GEE, AE), four problem types (Problems 1-4), five problem parameter sets (Problems 1 & 3—three levels, Problems 2 & 4—two levels), and three solution approaches (ILP with actual number of rows, Heuristics 1-4 with actual number of rows, Heuristics 1-4 with estimated number of rows). Figure 5 defines part of the experimental setup used in this work.

For each cell in Figure 5, we define three instances of Problems 1 and 3 with the number of views to be materialized held at 5, 10, and 20. Similarly, we define two instances of Problems 2 and 4, with the number of rows to be materialized held at 60% and 70% of the maximum number of rows required to materialize all views.

To estimate the number of rows in each view, we first sampled the root view using random sampling without replacement and the appropriate sample size. Next, we applied the corresponding estimators ((SE (Eq. 1), GEE (Eq. 2), and AE (Eq. 3 & 4)). This process generated an estimate of the number of rows for all the views.
corresponding to a given combination of lattice, sampling method, and sample size. This is the basic kernel to be used by executing appropriate changes in the constraints and/or objective function to create different instances of Problems 1-4. These problem instances were solved using appropriate heuristics.

Another set of instances of Problems 1-4 were generated using the actual number of rows. These problems instances were then solved using the optimal ILP approach, as well as the appropriate heuristic approach. Finally, the objective function values of all solutions were evaluated using the actual number of rows.

**Specific Problems Solved**

There are 18 individual combinations obtained by using two lattice types, three sampling techniques, and three sample sizes (Figure 5). For each of these combinations, three versions of Problem 1 were created corresponding to the number of views to be materialized set to 5, 10, and 20. This procedure was repeated for Problem 3.

It may be recalled that Problems 2 and 4 use storage space constraint in place of the number of views to be materialized constraint. In case of Problem 2, we started with the 18 instances generated earlier, and obtained two versions of the problem by setting the available total space to 60% and 70% of the total space required to materialize all the views of the lattice. It may be noted that typically, the space required to materialize the root view is about 45% of the total space required to materialize all views. This procedure was repeated for Problem 4. In the next section, we present the results of our investigation.

**COMPUTER SYSTEM SETUP USED FOR EXPERIMENTATION**

All computations were performed using Windows 2000 Server with dual 3.066 GHz processors and 3 GB of RAM, and SQL Server 2002. We have performed all experimentations on a stand-alone server by disconnecting the server from the network in order to remove the network effect. The same machine doubled as the client machine. We have created datasets for our experimentation using Lattice 1 and Lattice 2, discussed earlier.

**Results of Sampling**

The first step in problem generation is to use sampling procedures to generate problem instances. The actual and estimated numbers of rows corresponding to Lattices 1 & 2 for 20% sample size for different estimators are shown in Figures 6 & 7. For other sample sizes, summaries are presented in Figure 8. Here, we use the absolute deviation proportion (ADP) as a measure of how close the sampling estimate is to the population. The sum of all these proportions across the 27 views is used as a summary measure for the estimator.

From Figures 6 & 7, it appears that all three estimators tend to estimate the number of rows fairly accurately for the views that contain more aggregated data. These methods, in general, tend to lose their accuracy as the number of dimensions and levels in each dimension in the aggregation increases. In Figure 8, we notice an increase in the sum of ADP as sample size decreases. Also, notice that in both lattice instances, GEE and SE are more sensitive to sample size as compared to AE. One reason for this could be that AE adapts itself to the input data distribution, but the
Figure 6. Actual and estimated number of rows for Lattice 1 for sample size of 20%

<table>
<thead>
<tr>
<th>Node</th>
<th>Actual Rows (A)</th>
<th>SE (SS)</th>
<th>ADP</th>
<th>GEE (G)</th>
<th>ADP</th>
<th>AE</th>
<th>ADP</th>
<th>A-E/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td>6,001,192</td>
<td>6,001,192</td>
<td>0.000</td>
<td>6,001,192</td>
<td>0.000</td>
<td>6,001,192</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6,001,028</td>
<td>6,001,107</td>
<td>0.000</td>
<td>2,683,791</td>
<td>0.553</td>
<td>6,001,192</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5,710,004</td>
<td>5,858,214</td>
<td>0.026</td>
<td>2,642,931</td>
<td>0.537</td>
<td>6,001,192</td>
<td>0.051</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5,873,146</td>
<td>5,937,209</td>
<td>0.011</td>
<td>2,665,567</td>
<td>0.546</td>
<td>6,001,192</td>
<td>0.022</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5,510,833</td>
<td>5,753,234</td>
<td>0.044</td>
<td>2,612,666</td>
<td>0.526</td>
<td>6,001,192</td>
<td>0.089</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5,746,207</td>
<td>5,873,933</td>
<td>0.022</td>
<td>2,647,446</td>
<td>0.539</td>
<td>6,001,192</td>
<td>0.044</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>45,000</td>
<td>45,000</td>
<td>0.000</td>
<td>45,000</td>
<td>0.000</td>
<td>45,000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>26,250</td>
<td>26,250</td>
<td>0.000</td>
<td>26,250</td>
<td>0.000</td>
<td>26,250</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>6,000,127</td>
<td>6,000,570</td>
<td>0.000</td>
<td>2,683,638</td>
<td>0.553</td>
<td>6,001,192</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2,199,959</td>
<td>3,337,699</td>
<td>0.517</td>
<td>1,841,899</td>
<td>0.163</td>
<td>2,723,921</td>
<td>0.238</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1,126,591</td>
<td>1,680,661</td>
<td>0.492</td>
<td>1,164,430</td>
<td>0.034</td>
<td>1,224,216</td>
<td>0.087</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3,494,158</td>
<td>4,510,793</td>
<td>0.291</td>
<td>2,236,494</td>
<td>0.360</td>
<td>5,473,028</td>
<td>0.566</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1,378,602</td>
<td>2,266,261</td>
<td>0.644</td>
<td>1,429,477</td>
<td>0.037</td>
<td>1,616,643</td>
<td>0.173</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>617,446</td>
<td>793,585</td>
<td>0.285</td>
<td>675,558</td>
<td>0.094</td>
<td>642,939</td>
<td>0.041</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>300</td>
<td>300</td>
<td>0.000</td>
<td>300</td>
<td>0.000</td>
<td>300</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>4,804,917</td>
<td>5,555,939</td>
<td>0.115</td>
<td>2,496,101</td>
<td>0.481</td>
<td>6,001,192</td>
<td>0.249</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1,800</td>
<td>1,800</td>
<td>0.000</td>
<td>1,800</td>
<td>0.000</td>
<td>1,800</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>175</td>
<td>175</td>
<td>0.000</td>
<td>175</td>
<td>0.000</td>
<td>175</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>3,750</td>
<td>3,750</td>
<td>0.000</td>
<td>3,750</td>
<td>0.000</td>
<td>3,750</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1,050</td>
<td>1,050</td>
<td>0.000</td>
<td>1,050</td>
<td>0.000</td>
<td>1,050</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>99,996</td>
<td>100,250</td>
<td>0.003</td>
<td>100,723</td>
<td>0.007</td>
<td>100,136</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>199,996</td>
<td>201,990</td>
<td>0.010</td>
<td>203,213</td>
<td>0.016</td>
<td>200,754</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>12</td>
<td>12</td>
<td>0.000</td>
<td>12</td>
<td>0.000</td>
<td>12</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>25</td>
<td>25</td>
<td>0.000</td>
<td>25</td>
<td>0.000</td>
<td>25</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>150</td>
<td>150</td>
<td>0.000</td>
<td>150</td>
<td>0.000</td>
<td>150</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>7</td>
<td>7</td>
<td>0.000</td>
<td>7</td>
<td>0.000</td>
<td>7</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>26*</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>1</td>
<td>0.000</td>
<td>1</td>
<td>0.000</td>
<td></td>
</tr>
</tbody>
</table>

Sum 2.459 4.445 1.566

* Values are already known

Copyright © 2007, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
Figure 7. Actual and estimated number of rows for Lattice 2 for sample size of 20%

<table>
<thead>
<tr>
<th>Node</th>
<th>Actual Rows (A)</th>
<th>SE (S)</th>
<th>ADP</th>
<th>GEE</th>
<th>ADP</th>
<th>AE</th>
<th>ADP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td>9,275,377</td>
<td>9,275,377</td>
<td>0.000</td>
<td>9,275,377</td>
<td>0.000</td>
<td>9,275,377</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>3,722,050</td>
<td>5,050,036</td>
<td>0.357</td>
<td>2,801,702</td>
<td>0.247</td>
<td>4,289,960</td>
<td>0.153</td>
</tr>
<tr>
<td>2</td>
<td>1,208,341</td>
<td>1,570,251</td>
<td>0.300</td>
<td>1,246,307</td>
<td>0.031</td>
<td>1,238,163</td>
<td>0.025</td>
</tr>
<tr>
<td>3</td>
<td>406,941</td>
<td>451,995</td>
<td>0.111</td>
<td>438,316</td>
<td>0.077</td>
<td>416,636</td>
<td>0.024</td>
</tr>
<tr>
<td>4</td>
<td>1,003,445</td>
<td>1,246,332</td>
<td>0.242</td>
<td>1,047,691</td>
<td>0.044</td>
<td>1,016,862</td>
<td>0.013</td>
</tr>
<tr>
<td>5</td>
<td>326,971</td>
<td>352,443</td>
<td>0.078</td>
<td>344,281</td>
<td>0.053</td>
<td>329,426</td>
<td>0.008</td>
</tr>
<tr>
<td>6</td>
<td>73,697</td>
<td>74,295</td>
<td>0.008</td>
<td>75,125</td>
<td>0.009</td>
<td>73,939</td>
<td>0.003</td>
</tr>
<tr>
<td>7</td>
<td>61,055</td>
<td>61,436</td>
<td>0.006</td>
<td>62,052</td>
<td>0.016</td>
<td>61,178</td>
<td>0.002</td>
</tr>
<tr>
<td>8</td>
<td>609,926</td>
<td>729,032</td>
<td>0.195</td>
<td>676,665</td>
<td>0.109</td>
<td>639,527</td>
<td>0.049</td>
</tr>
<tr>
<td>9</td>
<td>21,576</td>
<td>21,576</td>
<td>0.000</td>
<td>21,576</td>
<td>0.000</td>
<td>21,576</td>
<td>0.000</td>
</tr>
<tr>
<td>10</td>
<td>53,002</td>
<td>53,005</td>
<td>0.000</td>
<td>53,066</td>
<td>0.001</td>
<td>53,002</td>
<td>0.000</td>
</tr>
<tr>
<td>11</td>
<td>110,228</td>
<td>111,299</td>
<td>0.010</td>
<td>112,822</td>
<td>0.024</td>
<td>110,749</td>
<td>0.005</td>
</tr>
<tr>
<td>12</td>
<td>17,980</td>
<td>17,980</td>
<td>0.000</td>
<td>17,980</td>
<td>0.000</td>
<td>17,980</td>
<td>0.000</td>
</tr>
<tr>
<td>13</td>
<td>43,495</td>
<td>43,549</td>
<td>0.001</td>
<td>43,841</td>
<td>0.008</td>
<td>43,535</td>
<td>0.001</td>
</tr>
<tr>
<td>14</td>
<td>840</td>
<td>841</td>
<td>0.001</td>
<td>841</td>
<td>0.001</td>
<td>863</td>
<td>0.027</td>
</tr>
<tr>
<td>15</td>
<td>34,223</td>
<td>34,238</td>
<td>0.000</td>
<td>34,264</td>
<td>0.001</td>
<td>34,237</td>
<td>0.000</td>
</tr>
<tr>
<td>16</td>
<td>1,200</td>
<td>1,200</td>
<td>0.000</td>
<td>1,200</td>
<td>0.000</td>
<td>1,200</td>
<td>0.000</td>
</tr>
<tr>
<td>17</td>
<td>699</td>
<td>699</td>
<td>0.000</td>
<td>699</td>
<td>0.000</td>
<td>699</td>
<td>0.000</td>
</tr>
<tr>
<td>18</td>
<td>6,150</td>
<td>6,151</td>
<td>0.000</td>
<td>6,152</td>
<td>0.000</td>
<td>6,153</td>
<td>0.000</td>
</tr>
<tr>
<td>19</td>
<td>1,000</td>
<td>1,000</td>
<td>0.000</td>
<td>1,000</td>
<td>0.000</td>
<td>1,000</td>
<td>0.000</td>
</tr>
<tr>
<td>20</td>
<td>4,417</td>
<td>4,417</td>
<td>0.000</td>
<td>4,417</td>
<td>0.000</td>
<td>4,417</td>
<td>0.000</td>
</tr>
<tr>
<td>21</td>
<td>1,798</td>
<td>1,798</td>
<td>0.000</td>
<td>1,798</td>
<td>0.000</td>
<td>1,798</td>
<td>0.000</td>
</tr>
<tr>
<td>22</td>
<td>12</td>
<td>12</td>
<td>0.000</td>
<td>12</td>
<td>0.000</td>
<td>12</td>
<td>0.000</td>
</tr>
<tr>
<td>23</td>
<td>70</td>
<td>70</td>
<td>0.000</td>
<td>70</td>
<td>0.000</td>
<td>70</td>
<td>0.000</td>
</tr>
<tr>
<td>24</td>
<td>100</td>
<td>100</td>
<td>0.000</td>
<td>100</td>
<td>0.000</td>
<td>100</td>
<td>0.000</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>10</td>
<td>0.000</td>
<td>10</td>
<td>0.000</td>
<td>10</td>
<td>0.000</td>
</tr>
<tr>
<td>26*</td>
<td>1</td>
<td>1</td>
<td>0.000</td>
<td>1</td>
<td>0.000</td>
<td>1</td>
<td>0.000</td>
</tr>
</tbody>
</table>

| Sum  | 1.309 | 0.633 | 0.31 |

* Values are already known
GEE performs well only for the data with high skew. In our case, data in the root view for both lattices have low skewness with the root view of Lattice 1 having large number of low frequency elements while that of Lattice 2 having small number of low frequency elements.

Also, one can notice that for all sample sizes, variations were less in the case of Lattice 2 compared to Lattice 1. One reason could be that the root view of Lattice 2 is comparatively less skewed than that of Lattice 1 or it could be the presence of small number of low frequency elements in the root view of Lattice 2. Also, as discussed earlier, in the TPC-H database a size of 1 GB is not adequate to get a fully representative picture. For example, in seven years of data, no customer has ordered the same product more than once. Because of this type of anomaly, some of the derived views have the same number of rows as their immediate parent.

Results for the General MVS Problems

We instantiated and solved Problem 1 using the estimates obtained from three sample sizes and applying different solution procedures described earlier. Figure 9 gives the results for the two extreme cases, namely 5% and 20% for the three estimators. We found that problem instances created using AE produced the best results even for 5% sampling. The maximum deviation from
### Figure 9. Percent deviation of the cost of solution for Problem 1 obtained by Heuristic 1

<table>
<thead>
<tr>
<th>Lattice</th>
<th>Number of Views Materialized</th>
<th>Cost of Optimal Solution ($Z^*$) with Actual Number of Rows</th>
<th>Percent Deviation of Heuristic 1 Solution with Actual Number of Rows from $Z^*$</th>
<th>Percent Deviation Of Heuristic 1 Solution With Estimated Number Of Rows From $Z^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SE</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>62,023,340</td>
<td>0.00</td>
<td>18.46</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>56,396,780</td>
<td>0.00</td>
<td>1.78</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>54,844,770</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>25,421,580</td>
<td>0.00</td>
<td>7.25</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>17,638,620</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>16,989,310</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

### Figure 10. Percent deviation of the cost of solution for Problem 2 obtained by Heuristic 2

<table>
<thead>
<tr>
<th>Lattice</th>
<th>Number of Views Materialized</th>
<th>Cost of Optimal Solution ($Z^*$) with Actual Number of Rows</th>
<th>Percent Deviation of Heuristic 2 Solution with Actual Number of Rows from $Z^*$</th>
<th>Percent Deviation Of Heuristic 2 Solution With Estimated Number Of Rows From $Z^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SE</td>
</tr>
<tr>
<td>1</td>
<td>60</td>
<td>55,170,060</td>
<td>0.00</td>
<td>1.06</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>54,970,890</td>
<td>0.00</td>
<td>0.53</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>47,622,320</td>
<td>0.00</td>
<td>19.28</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>22,832,300</td>
<td>0.01</td>
<td>0.07</td>
</tr>
</tbody>
</table>
### Figure 11. Percent deviation of the cost of solution for Problem 3 obtained by Heuristic 3

<table>
<thead>
<tr>
<th>Lattice</th>
<th>Number of Views Materialized</th>
<th>Cost of Optimal Solution (Z*) with Actual Number of Rows</th>
<th>Percent Deviation of Heuristic 3 Solution with Actual Number of Rows from Z*</th>
<th>Percent Deviation of Heuristic 3 Solution With Estimated Number Of Rows From Z*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SE</td>
<td>GEE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5% Sample Size</td>
<td>20% Sample Size</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>22,531,820</td>
<td>0.00</td>
<td>113.07</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>11,265,910</td>
<td>0.00</td>
<td>166.34</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>11,265,910</td>
<td>0.00</td>
<td>166.34</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>9,275,377</td>
<td>322.73</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>9,275,377</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>9,275,377</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

### Figure 12. Percent deviation of the cost of solution for Problem 4 obtained by Heuristic 4

<table>
<thead>
<tr>
<th>Lattice</th>
<th>Number of Views Materialized</th>
<th>Cost of Optimal Solution (Z*) with Actual Number of Rows</th>
<th>Percent Deviation of Heuristic 4 Solution with Actual Number of Rows from Z*</th>
<th>Percent Deviation of Heuristic 4 Solution With Estimated Number Of Rows From Z*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SE</td>
<td>GEE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5% Sample Size</td>
<td>20% Sample Size</td>
</tr>
<tr>
<td>1</td>
<td>60</td>
<td>11,365,910</td>
<td>0.00</td>
<td>166.34</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>11,265,910</td>
<td>0.00</td>
<td>166.34</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>9,275,377</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>9,275,377</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
the optimal cost was 4.3%, and it produced the optimal solution in 50% of the cases. The results for the problem instances using AE with 10% and 20% sampling were not significantly different. Results for problem instances using SE and GEE were more sensitive to the sample size used, and were generally inferior to those of AE.

Appropriate comparisons call for calculating the objective function values (i.e., cost values) of any solution using the actual number of rows. Hence, even when a problem has been instantiated using estimates for the number of rows in each view, in all comparisons the objective function is re-calculated using the actual number of rows.

We instantiated and solved Problem 2 using the estimates obtained from three sample sizes and applying different solution procedures described earlier. Figure 10 gives the results for the two extreme cases, namely 5% and 20% for the three estimators. In this case, we have found that SE estimator produced the best results across all sample sizes. The maximum deviation was 0.46% at a level 20% sampling and it was 19.28% for 5% sampling. Results for all three estimators were more sensitive to sample sizes and were generally inferior to that of SE.

**Results for the Bottleneck MVS Problems**

We generated instances of Problem 3 with arbitrary weights, and the rest of the data generated by using the estimates obtained by applying three estimators each with three sample sizes. These were solved using Heuristic 3, and by the integer linear programming procedure referenced earlier. Figure 11 presents partial results. We found that SE produced consistently good results at the level of 20% sampling. Results of AE were not sensitive to sample size. It produced good results for many cases but was not effective for Lattice 2 when the number of views allowed is highly constrained. GEE was more sensitive to sample size and it produced good results only at 20% level in all cases except when the number of views to be materialized are restricted to five.

Some of these larger deviations may be attributed to the nature of the underlying lattices as well as the fact that the bottleneck objective is not a regular measure. As mentioned earlier, Lattice 2 has a good degree of aggregation, as we move from the root view to more aggregated views, reducing the number of dimensions in the process. That is, each level of aggregation may reduce the number of rows by about 50-70% and these reductions are evenly distributed as a function of the number dimensions in a view. On the other hand, Lattice 1, the number of rows reduces rather slowly as we aggregate by different dimensions. All lattices have a fully aggregated view with only one row, and hence eventually the total reduction due to aggregation will be the same in any lattice, but the reduction is rather sudden as opposed to being smooth. The bottleneck objective itself may also contribute to the deterioration, since one bad estimate may be a cause for a poor solution.

As in the case of Problem 3, we generated instances of Problem 4 with arbitrary weights, and the number of rows estimated by sampling methods studied. These problem instances were solved using Heuristic 4, and by the integer linear programming procedure referenced earlier. Figure 12 presents partial results. We found that AE produced consistently good results and was not sensitive to sample size. GEE produced good results for many cases but was not
Figure 13. Computational times in seconds

effective for Lattice 1 at 5% sampling. SE was very sensitive to sample size. It produced good results for 20% sampling, and it produced good results for Lattice 2 at even 5%. But for Lattice 1, it produced poor results at 5% sampling level. The overall reasons may be a combination of the difficulty in estimating the number of rows for Lattice 1 at 5% sample size and the bottleneck objective.

Computational Times to Estimate the Number of Rows

The main purpose of sampling is to reduce the time needed to count the number of rows in each view in a given lattice, which in turn will be used to generate problem instances. It is hence appropriate to examine the efficiency of this process. In this section, we examine the relationship between sample size and the time for obtaining the estimates. Figure 13 presents the computation times required to determine the actual number of rows in each view in a given lattice by complete enumeration and the times required to obtain the estimates by using the three different sampling estimators at three levels of sampling. We have also shown the percent time-savings by using the sampling and the estimating methods in Figure 14.

There are two steps to generating the estimate: the first step is to draw samples from the root view and the second step is to use the appropriate sampling method to calculate the estimate. The times shown in Figure 13 are the sum of the two. One may notice that computational time is almost directly proportional to the sample size. This implies that the bulk of the time is very sensitive to sample size.
In Figure 13, bars titled Lattice 1 (Actual) and Lattice 2 (Actual) show the time needed for counting the actual number of rows present in all views in a given lattice. Even though Lattice 2 has more records in the root view (more than nine million) as compared to Lattice 1 (about six million), the time taken in the case of Lattice 2 was much less. The main reason for this could be that in the database corresponding to Lattice 2, most key fields are numeric which enables database engine to process the records faster.

Figure 14 shows the details about percentage savings in computation times. In the case of Lattice 1, even with a 20% sample size, time savings ranged from 65% to 68%. For a sample size of 5%, the savings were as high as 91%, with the GEE giving the best results (59.62 seconds—Figure 13). But as discussed earlier, the overall performance of AE was the best with regard to quality of the solutions. In Figure 13, one can notice that the time taken by the AE is very much comparable with the time taken by the GEE.

In the case of Lattice 2 (Figure 14), GEE took the least time to estimate the number rows. However, its performance in terms of prediction accuracy was not the best. For the sample size of 5%, the savings were as high as 87% with the GEE and 85% with the AE estimator. As seen earlier, the performance of the AE estimator was best when it comes to prediction accuracy. Based on the above results, the AE has given the best performance in our experimental evaluation both in terms of prediction accuracy and savings in time.

The previous statistics clearly indicate the worth of using statistical sampling techniques in generating problem instances for
a data warehouse as it results in significant savings in terms of time without compromising the accuracy in terms of deviation of the cost of solution from the optimal solution. Even a sample size of as low as 5% produced satisfactory results.

CONCLUSION AND FUTURE RESEARCH

Time is of the essence in supporting dynamic online decision support systems. One of the key features of a dynamic online DSS is its ability to quickly and accurately respond to the queries by managers, employees, and customers. The dynamism of the system comes from rapidly changing data and end users' priorities. One of the methods used for addressing effective access to the large amount of aggregated data is to materialize an appropriate set of views at the data warehouse. There are several versions of the MVS problem, which are of interest in this context.

In this research, we addressed the problem of generating instances of the MVS Problem using sampling methods and the basic information provided by the root view (base cuboid). We employed three sampling methods at three levels of sampling using two lattices in order to formulate the MVS problems. Next, we evaluated the efficacy of the sampling procedure in solving different types of the MVS problems.

Our evaluation of the sampling methods revealed that even for a sample size of 5%, the solutions to the MVS problems competed very effectively against the optimal solutions. In about 70% of the instances, the 5% sampling level produced solutions close to the optimal solutions.

The time saved in generating problem instances can be as high as 90% with little loss in solution quality, making the sampling procedure a prime candidate for dynamic online problem solving tool for the MVS type problems. This procedure can be used to change the set of materialized views, which will in turn lead to faster response time. Closeness of the sampling based solution to the optimal solution did vary much based on the type of problem and the underlying lattice. One line of future research may be to explore this relationship further and classify the problem situations where sampling will be most effective.

We expect the size of database and data warehouse applications used in the MVS type problems will expand exponentially over time, which will make sampling approaches to problem instantiation a necessity. Hence, we expect this research to spawn further research in the area of developing sophisticated and new sampling methods and expanding application of current methods to different database related problems. This general approach may also be of interest to researchers in large-scale model building where current methods emphasize accurate problem instantiation as opposed to using sampling to instantiate.

REFERENCES

Adiba, M. E., & Lindsay, B. (1980). Database snapshots. In Proceedings of the 6th International Conference on Very Large Databases (pp. 86-91), Montreal, Canada.


a multidimensional database. In *Proceedings of the 23rd VLDB Conference* (pp. 156-165), Athens, Greece.


warehouse environment. *Intelligent Data Engineering and Automated Learning, Lecture Notes in Computer Science*, 2690, 709-716.


Mesbah Ahmed is a professor of information operations and technology management at the University of Toledo, Toledo, Ohio, USA. He received his BS in chemical engineering from the Bangladesh University of Engineering and Technology, Dhaka (1968), MS and PhD in industrial engineering from Texas Tech University, Lubbock, Texas, USA (1973 and 1976, respectively). His major research interests are in the area of physical and logical database design, application development, and enterprise resource planning systems.

Vikas Agrawal is an assistant professor of information systems at Fayetteville State University, Fayetteville, NC, USA. He received his BE in mechanical engineering (1996) and MBA in marketing and finance (1999) from Nagpur University, Nagpur, India. He completed MS in industrial engineering (2000) and PhD in manufacturing management and engineering (2005) from the University of Toledo, Toledo, Ohio, USA. His main research interests include databases, data warehousing and OLAP.

Udayan Nandkeolyar is an associate professor at The University of Toledo, Ohio. He received a Bachelor of Technology from The Indian Institute of Technology, Delhi, India in 1975, a Master of Business Management from the Asian Institute of Management, Manila, Philippines in 1980 and a Doctor of Philosophy degree from Penn State University, USA in 1988. He has published in the areas of manufacturing flexibility, group technology, scheduling and logistics.

P. S. Sundararaghavan is a professor in the information operations and technology management department at the University of Toledo, Toledo, Ohio, USA. He received his BT in Chemical Engineering from the University of Madras, MBA from the Indian Institute of Management-Calcutta and PhD in management science from The University of Tennessee, Knoxville. His current research interests are in supply chain management and modeling of problems in information systems. He has published in many scholarly journals.