Value-Focused Thinking and its Application in MIS Research
Hong Sheng, University of Missouri-Rolla, USA
Fiona Pui-Hoon Nah, University of Nebraska-Lincoln, USA
Keng Siau, University of Nebraska-Lincoln, USA

Research Papers

1 View Materialization in a Data Cube: Optimization Models and Heuristics
Vikas Agrawal, Fayetteville State University, USA
P. S. Sundararaghavan, The University of Toledo, USA
Meshbah U. Ahmed, The University of Toledo, USA
Udayan Nandkeolyar, The University of Toledo, USA
Data warehouse has become an integral part in developing a DSS in any organization. This article considers a bottleneck objective in designing such a materialization scheme which has the effect of guaranteeing a certain level of performance. The article examines linear integer programming formulations, and develops heuristics and reports on the performance of these heuristics.

A Space-Efficient Protocol for Consistency of External View Maintenance on Data Warehouse Systems: A Proxy Approach
Shi-Ming Huang, National Chung Cheng University, Taiwan
David C. Yen, Miami University, USA
Hsiang-Yuan Hsu, National Chung Cheng University, Taiwan
The materialized view approach is widely adopted in implementations of data warehouse systems in order for efficiency purposes. In this article, a space-efficient protocol for materialized view maintenance with a global data view on data warehouses with embedded proxies is proposed.

Intelligent Search for Experts Using Fuzzy Abstraction Hierarchy in Knowledge Management Systems
Kun-Woo Yang, Keimyung University, South Korea
Soon-Young Huh, KAIST Business School, South Korea
In this article, we propose an intelligent search framework to provide search capabilities for experts who not only match search conditions exactly but also belong to the similar or related subject fields according to the user’s needs.

Information Mediation Using Metamodels: An Approach Using XML and Common Warehouse Metamodel
Luyin Zhao, The State University of New Jersey - Rutgers, USA
Keng Siau, University of Nebraska - Lincoln, USA
This article discusses the concept of information mediation and several related research projects that use a typical mediation model. The problems and issues with existing approaches are discussed. This article then proposes the use of metamodels as an advanced architecture for information mediation, where XML is a main driving force.

The Index to Back Issues is available on the WWW at http://www.igi-pub.com
View Materialization in a Data Cube: Optimization Models and Heuristics

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

ABSTRACT

Data warehouse has become an integral part in developing a DSS in any organization. One of the key architectural issues concerning the efficient design of a data warehouse is to determine the “right” number of views to be materialized in order to minimize the query response time experienced by the decision makers in the organization. We consider a bottleneck objective in designing such a materialization scheme which has the effect of guaranteeing a certain level of performance. We examine linear integer programming formulations, and develop heuristics and report on the performance of these heuristics. We also evaluate heuristics reported in the literature for the view materialization problem with a simpler objective.

Keywords: data cube; data warehouse; view materialization

INTRODUCTION

In today’s fast-paced, ever-changing and wants-driven economy, information is seen as a key business resource to gain competitive advantage (Haag, Cummings & McCubbrey, 2005). Effective use of this information requires good decision support systems. Most decision support systems require reliable and elaborate data backbone which needs to be converted into useful information. With the widespread availability and ever-decreasing cost of computers, telecommunications technologies, and Internet access, most businesses have collected a wealth of data. However, that is only the first and easy step. Many firms are becoming data rich but remain information and knowledge
poor (Gray & Watson, 1998; Grover, 1998; Han & Kamber, 2001; Nemati, Steiger, Iyer & Herschel, 2002). To alleviate this problem, many corporations have built, or are building, unified decision-support databases called data warehouses on which decision makers can carry out their analysis. A data warehouse is a very large data base that integrates information extracted from multiple, independent, heterogeneous data sources into one centralized data repository to support business analysis activities and decision-making tasks.

Business analysts run complex queries over this centralized data repository housed in a data warehouse to gain insights into the vast data and to mine for hidden knowledge. The key to gaining such insight is to design a decision support system which would get the right information to the right person and at the right time which will aid in making quality and often strategic decisions. In order to achieve this objective, design of the data warehouse architecture plays a pivotal role. There are many architectural issues concerning the efficient design of a data warehouse system. Lee, Kim, and Kim (2001) highlighted the importance of metadata for implementing data warehouse. They pointed out that integrating data warehouse with its metadata offers a new opportunity to create a more adaptive information system. Furtado (2006) proposed the concept of node partitioning, a method for parallelism, to improve the performance of a data warehouse system. Huang, Lin, and Deng (2005) proposed an intelligent cache mechanism for a data warehouse system in a mobile environment. They pointed out that because mobile devices can often be disconnected from the host server, and due to the low bandwidth of wireless networks, it is more efficient to store query results from a mobile device in the cache.

Data cube design is one such important aspect of the data warehouse architecture. Data cubes are constructs to store subsets of summarized data by some measures of interest for easy and quick access, and for timely and dynamic updates of these summarized data on an ongoing basis (Chun, Chung & Lee, 2004).

Accessing data from a data cube, if not materialized, can be a time consuming and resource intensive process. A data cube consists of many views with existing interrelated dependencies among themselves (such view is also known as a cuboid or a query). If such a view is stored, it is denoted as a materialized view. 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. Since not all views in a data cube may be materialized due to constraints imposed on the system, 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.

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. The problem is generally described as the materialized view selection (MVS) problem, which has the objective of minimizing the access time subject to constraints on either the number of views that may be materialized or the storage space that may be used for materialization of views (Gupta & Mumick, 2005; Harinarayan, Rajaraman & Ullman, 1996, 1999). In this article we have worked on several variants of the MVS problems and have solved these optimally as well as using heuristics. Our specific contributions may be summarized as follows:

- We have presented a linear integer programming formulation for two versions of the MVS problem with a bottleneck objective, which minimizes the maximum weighted access time experienced by any class of users.
- We have developed heuristics for the above two problems and reported on their performance.
- We have also presented linear integer programming formulations for the general MVS problem reported in Harinarayan et
al. (1999), and also reported on the performance of their greedy heuristics.

RELATED WORK
Today, virtually every major corporation has built, or is building, unified data warehouses to support business analysis activities and decision-making tasks. According to a report from the market research and consulting firm, the Palo Alto Management Group (Mountain View, CA, www.pamg.com), the market for data warehousing and decision support will grow more than 50% a year to pass $113 billion annually by the year 2002. The study, based on forecast modeling and 375 interviews, also predicted that average data warehouse size will balloon from 272 GB to 6.5 TB, and the number of users accessing data warehouses will soar from 2,200 to nearly 100,000 in the next 3 years. This comes to hundreds of billions of dollars of annual investments in these technologies for the current decade (Hillard, Blecher & O’Donnell, 1999; Walton, Goodhue & Wixom, 2002).

In a typical organization, information is spread over many different multiple, independent, heterogeneous, and remote data sources. Acting as a decision support system, a data warehouse extracts, integrates, and stores the “relevant” information from these data sources into one centralized data repository to support the information needs of knowledge workers and decision makers in the form of online analytical processing (OLAP) (Han & Kamber, 2001).

Business analysts run business queries over this centralized data repository to gain insights into the vast data and to mine for the hidden knowledge. Results of such queries are generally precomputed and stored ahead of time at the data warehouse in the form of materialized views. Such materialization of views reduces the query execution time to minutes or seconds which may otherwise take hours or even days to complete.

For an extension of these concepts and a discussion of the appropriate architecture for harnessing knowledge in a broad sense using knowledge warehouse, where knowledge work-
North Carolina, please refer to Watson, Fuller, and Ariyachandra (2004).

The focus of our article is to improve access times across users in an egalitarian or prioritized fashion in a data warehouse. A commonly used technique to improve net access time is to materialize (precompute and store) the results of frequently asked queries. But picking the "right" set of views to materialize is a nontrivial task. For example, one may want to materialize a relatively infrequently asked query if it helps in answering many other frequent queries faster. One cannot materialize all the views in a given data cube as such materialization is constrained by the availability of storage space, view maintenance cost, and computational time. On the other extreme, if one does not materialize any view, then business queries have to be run over the source data, a process that would take considerable time leading to intolerable delays. Between these two extremes, one needs to find the optimum number of views to be materialized that will give a reasonably good query response time while satisfying all the constraints. The MVS problem with the objective of minimizing the sum of access times has been shown to be NP-Complete (Harinarayan et al., 1999). The bottleneck version, which is the focus of this article, is likely to be difficult to solve optimally as it attempts to minimize the maximum weighted number of pages to be retrieved. Next, we address the literature specifically related to the materialized views.

In the 1980s, materialized views were investigated 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).

Gray, Chaudhuri, Bosworth, Layman, Reichart, and Venkatrao (1997) propose the data cube as a relational aggregation operator generalizing group-by, cross-tabs, and subtotals. Harinarayan et al. (1996, 1999) have discussed the major features of the MVS problem elaborately. They have employed a lattice framework to capture the dependencies among views. This lattice framework was then used to develop a greedy algorithm.

Kalnis, Mamoulis, and Papadias (2002) have reported on a randomized local search algorithm to generate the "right" views to materialize; this approach is particularly useful in large dynamic view selection problems where the execution time for solving the materialized view selection problem is critical. Park, Kim, and Lee (2002) assume that the set of materialized views present is given and then ask the question: How do we 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.

One might wonder whether solutions to these problems have impacted software development in the context of data warehouse. We are aware of applications in SQL Server Analysis Services (Jacobson, 2000) where user input includes disk space available for use in a data warehouse application. The response time is inversely related to allocated disk space. We believe that this and similar work will impact software development in the future. Next we define some variants of the MVS problem.

**Problem Definition**

Figure 1 presents a data cube and its associated views organized as a lattice in a hypothetical data warehouse. Each node represents a view (i.e., cuboid) and the numbers inside each node represent the number of pages (p) that must be retrieved to respond to the underlying query and the weight (w) that is associated with each view. We are using pages rather than rows as a surrogate for estimating the time it will take to answer the underlying query. This is consistent with the typical database retrieval process where blocks of rows called pages are retrieved during each physical access of the database. The page is then stored in cache or RAM from where rows can be retrieved quickly. Consequently, the number of pages is a better estimator of
the time needed to obtain a particular view. The weight (w) of each view is a function of frequency of access and/or the importance of the user accessing the view.

View A (at the root) contains the lowest level of aggregated data, and it is assumed to be materialized. The links in the lattice indicate parent-child relationships. Hence View B, for example, can be obtained from View A by processing 100 pages of data. If View B is materialized, it will contain 50 pages of data, and a query on View B will involve retrieving 50 pages. In general, for a given node, an ancestor node is defined as any node from which the given node may be reached by traversing only directed arrows. A query on a view may be answered by materializing the corresponding view or from any of its materialized ancestor views. For example, obtaining View F from View A will require retrieving 100 pages, while it will require retrieving 50 pages to obtain it from View B, or 70 pages to obtain it from View C. View F cannot be obtained from View D. However, we would not have to use both Views B and C even if both were materialized in order to obtain View F.

Based upon this type of configuration, we would like to know the specific set of views that should be materialized to achieve some predetermined objectives. One such objective is termed as a bottleneck objective. For the bottleneck version formulated here, the objective is to minimize the maximum weighted number of pages to be retrieved. Minimizing the maximum weighted number of pages to be retrieved attempts to limit the amount of time it will take to obtain any of the views. This is a bottleneck objective as it tries to minimize the maximum value. This measure also takes into account the relative importance (i.e., weight) of the various views. This objective will help to improve the response time of the system.

We consider two types of constraints—the total number of views that may be materialized and the total amount of storage space (measured in pages) available to store the materialized views—leading to two formulations. Limiting the number of views that may be materialized is a commonly used constraint and attempts to limit the complexity of the data warehouse design architecture (Harinarayan et al., 1996, 1999). The storage of too many views will make the data warehouse design more complex, and increase the amount of time and effort required to both compute and maintain the various views. In addition, there may be a limit on the amount of space available to store the materialized views. Hence, from a practical point of view, a more realistic constraint might be to compute the storage requirement of the views and limit this to the amount of space available.
General Assumptions Used in MVS Problems
In this article, we have used the following assumptions to define and solve the MVS problems:

a. The cost of constructing a view from its materialized ancestor is a linear function of the number of pages in its materialized ancestor.

b. If view $i$ is materialized, its storage cost will be $p_i$, where $p_i$ is the number of pages in view $i$.

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

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

Next, we define bottleneck variants of the MVS problem (i.e., Problem 1 and 2) and the traditional MVS problems discussed in the literature (i.e., Problems 3 and 4).

Problem 1
Given a data cube, the list of associated views, weight associated with each view, and the maximum number of views that may be materialized, determine the set of views to be materialized so as to minimize the maximum weighted number of pages to be retrieved.

$$\sum_{i} x_{ij} = 1, \forall j$$

$$x_{ij} \leq x_{ij}, \forall j$$

$$\sum_{i} x_{ij} \leq T$$

$$w_{ij} x_{ij} = Z_j, \forall j$$

$$Z \geq Z_j, \forall j$$

$$x_{ij} = 1 \text{ or } 0$$

Where

$$N = \{1, 2, 3, ..., T_{max}\}$$

$$i, j \in N$$

$$x_{ij} = 1 \text{ implies that view } j \text{ is obtained from } i \text{ and } 0 \text{ otherwise.}$$

Problem 4
Given a data cube, the list of associated views, weight associated with each view, and the maximum number of pages that can be stored, determine the set of views to be materialized so as to minimize the total weighted number of pages to be retrieved.

In the next section, we present the 0-1 Integer Programming models for these problems and how it may be adopted for the conventional MVS problems.

Integer Programming Models for the MVS Problem
In this section, we have developed the integer programming formulations for the bottleneck MVS problem. We present the linear integer programming models LIP 1 and LIP 2, for Problems 1 and 2 respectively.

LIP 1

$$\text{Min: } Z \quad (1)$$

Such that:

$$\sum_{j} x_{ij} = 1, \forall j \quad (2)$$

$$x_{ij} \leq x_{ij}, \forall i \text{ and } i \neq j \quad (3)$$

$$\sum_{i} x_{ij} \leq T \quad (4)$$

$$w_{ij} \sum_{i} x_{ij} = Z_j, \forall j \quad (5)$$

$$Z \geq Z_j, \forall j \quad (6)$$

$$x_{ij} = 1 \text{ or } 0 \quad (7)$$

Copyright © 2007, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
Specifically, \( x_i = 1 \) implies that view \( i \) is obtained from \( i \) itself meaning that view \( i \) is materialized.

\[ w_j = \text{Weight assigned to view } j \]
\[ p_{ij} = \text{Number of pages associated with view } i, \]
if view \( i \) is an ancestor of \( j \)
= \( M \) otherwise (where \( M \) assumes a very big value)
\[ T = \text{Maximum number of views that can be materialized} \]
\[ T_{\text{tot}} = \text{Total number of views present in a given data cube} \]
\[ w_j = \text{Weight of view } j \]
\[ Z_j = \text{Weighted number of pages that must be retrieved to obtain view } j \]
\[ Z = \text{Maximum weighted number of pages that must be retrieved to obtain any of the views in a given data cube} \]

**Explanation**

We will illustrate the above formulation with the example shown in Figure 1 with the caveat that at most five views may be materialized. The weights and pages associated with the nodes are also given in Figure 1. Our formulation uses the “Big M” approach. We assign an arbitrary high number of pages (for computational purposes, it is assumed to be 10,000 pages for this specific illustration) for obtaining view \( j \) from view \( i \) when view \( i \) is not an ancestor of view \( j \). Hence in Table 1, which presents the number of pages, one would find many 10,000s which correspond to view \( j \) that cannot be obtained from view \( i \).

The objective function in (1) minimizes the maximum weighted number of pages to be retrieved to obtain all the views in the given data cube. Equations in (2) ensure that every view in the data cube can be obtained, and each is obtained from exactly one source. This will in general be the most economical view among the views materialized. However, if there are alternate solutions, this view may be any of the views materialized by the solution. Equations in (3) ensure that view \( j \) can be obtained from view \( i \) if and only if view \( i \) is materialized. For example, the cell corresponding to column \( G \) and row \( C \) in Table 2 can be 1 only if cell corresponding to column \( C \) and row \( C \) is equal to 1. Equation (4) ensures that a maximum of ‘\( T \)’ views will be materialized that is the sum of the main diagonal elements of Table 2 must be less than or equal to \( T \) (for this illustration, \( T = 5 \)). Equations in (5) compute the weighted number of pages to be retrieved to obtain each view \( j \). Equations in (6) ensure that the optimal \( Z \) is equal to the largest weighted number of pages retrieved. The constraints in (7) define the \( x_{ij} \) as binary. If \( x_{ij} = 1 \), it implies that queries on view...

---

**Table 1. The \( p_{ij} \) matrix for Problem 1 defined in Figure 1**

<table>
<thead>
<tr>
<th>( j )</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>A</td>
<td>10000</td>
<td>50</td>
<td>10000</td>
<td>10000</td>
<td>50</td>
<td>50</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>B</td>
<td>10000</td>
<td>70</td>
<td>10000</td>
<td>10000</td>
<td>70</td>
<td>70</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>C</td>
<td>10000</td>
<td>60</td>
<td>10000</td>
<td>10000</td>
<td>60</td>
<td>60</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>D</td>
<td>10000</td>
<td>40</td>
<td>10000</td>
<td>10000</td>
<td>40</td>
<td>40</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>E</td>
<td>10000</td>
<td>30</td>
<td>10000</td>
<td>10000</td>
<td>30</td>
<td>30</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>F</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>G</td>
<td>10000</td>
<td>50</td>
<td>10000</td>
<td>10000</td>
<td>50</td>
<td>50</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>H</td>
<td>10000</td>
<td>40</td>
<td>10000</td>
<td>10000</td>
<td>40</td>
<td>40</td>
<td>10000</td>
<td>10000</td>
</tr>
</tbody>
</table>
Table 2. The $x_{ij}$ solution for Problem 1 defined in Figure 1

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$j$ is answered using view $i$. Otherwise, $x_{ij} = 0$. $x_{ij} = 1$ implies that view $i$ is materialized and $x_{ij} = 0$ implies that it is not materialized.

The optimal solution indicates that Views A, B, E, F, and G should be chosen for materialization to minimize the maximum weighted number of pages to be retrieved. The objective function value for this solution is 600. In addition it tells us that Views C, D, and H should be obtained from View A. This accounts for all the “T” views (i.e., 5 views) to be materialized.

LIP 2

Min $Z$  \hspace{1cm} (8)

Such that:

$\sum_{i} x_{ij} = 1, \forall j$  \hspace{1cm} (9)

$x_{ij} \leq x_{ik}, \forall i \text{ and } i \neq j$  \hspace{1cm} (10)

$\sum_{i} x_{ij} p_{ij} \leq S$  \hspace{1cm} (11)

$w_{j} \sum_{i} x_{ij} p_{ij} = z_{j}, \forall j$  \hspace{1cm} (12)

$Z \geq Z_{j}, \forall j$  \hspace{1cm} (13)

$x_{ij} = 1 \text{ or } 0$  \hspace{1cm} (14)

Where:

$N = \{1, 2, 3, \ldots, T_{\text{total}}\}$

$i, j \in N$

$x_{ij} = 1$ implies that view $j$ is obtained from $i$ and 0 otherwise.

Specifically, $x_{ij} = 1$ implies that view $i$ is obtained from $i$ itself meaning that view $i$ is materialized.

$p_{ij}$ = Number of pages associated with view $i$, if view $i$ is an ancestor of $j$

$= M$ otherwise (where $M$ assumes a very big value)

$S$ = Maximum number of pages that can be stored

$w_{j}$ = Weight of view $j$

$Z_{j}$ = Weighted number of pages that must be retrieved to obtain view $j$

$Z$ = Maximum weighted number of pages that must be retrieved to obtain any of the views in a given data cube

$T_{\text{total}}$ = Total number of views present in a given data cube

Explanation

The objective function in (8) minimizes the maximum weighted number of pages to be retrieved to obtain all the views in a given data cube. The constraints are the same as Problem
1, except that equations in (11) ensure that the number of pages stored does not exceed “S,” whereas the corresponding constraint (4) in Problem 1 was limiting the number of views that can be materialized.

LIP 3
This denotes the integer programming formulation of the traditional MVS problem which minimizes the total weighted number of pages to be retrieved when the number of views to be materialized is constrained. This objective function is given as follows.

\[
\text{Min. } \sum_i \sum_j w_{ij} \cdot p_{ij} \tag{15}
\]

Constraints are same as (2), (3), and (4) and (7).

LIP 4
This denotes the integer programming formulation of the traditional MVS problem which minimizes the total weighted number of pages to be retrieved when disk space (measured in number of pages) for views to be materialized is constrained. The objective function is same as (15) above and the constraints are given by (9), (10), (11) and (14).

In the next section, we present the heuristic procedures developed for various versions of the MVS problem.

Heuristic Procedures for the MVS Problem

Heuristic When Number of Views to be Materialized is Constrained

Next, we present a heuristic procedure (Heuristic 1) for Problem 1.

Heuristic 1.

- Step 1: Let \( k \) be the maximum number of views that may be materialized. Let \( |N| \) denote the cardinality of the set \( N \), where \( N \) is the set of all views under consideration, which will initially be all views except the root View \( A \). Let \( M \) be the set of views to be materialized. Initially, let \( M = \{ A \} \) where View \( A \) is the root view which is required to be materialized. Let \( w_j \) be the weight associated with view \( j \) and let \( p_j \) be the number of pages required to represent materialization of view \( j \). For each view \( j \), calculate \( f_j = w_j \cdot p_j \) (minimum number of pages to be retrieved to answer queries based on view \( j \) using only views in the current solution given by set \( M \), which has all materialized views). Let \( Z = \max \{ f_j \text{ for } j \in N \cup M \} \).

- Step 2: For each View \( j \in N \), calculate \( Z_j \) for \( j \in N \), where \( Z_j \) is the objective function value if view \( j \) were to be materialized in addition to all the views in \( M \). Let \( j' \in N \) be such that it maximizes \( Z_j - Z \) for \( j \in N \) and \( Z_j > 0 \) breaking ties arbitrarily. If there is no such \( j' \), go to Step 3. Otherwise, let \( M = M \cup j', N = N - j' \). Set \( Z = Z_{j'} \). If \( |M| = k \) go to Step 3, else go to Step 2.

- Step 3: Views to be materialized are given by the set \( M \). \( Z \) gives the objective function.

Example Demonstrating Heuristic 1

Applying Heuristic 1 to the problem whose data cube and its associated views are given in Figure 1, we get the following:

- Step 1: Let \( M = \{ A \} \) and \( N = \{ B, C, D, E, F, G, H \} \). Let the set of weights associated with Views \( A \) through \( H \) be \( \{ 1, 10, 2, 3, 20, 15, 10, 5 \} \). Initial value of the objective function \( Z = \max \{ 1 \cdot 100, 10 \cdot 100, 2 \cdot 100, 3 \cdot 100, 20 \cdot 100, 15 \cdot 100, 10 \cdot 100, 5 \cdot 100 \} = 2000 \).

- Step 2: For each node \( j \) calculate \( Z_j \). For example, details of the calculation of \( Z_j \) corresponding to materializing View \( B \) is given by \( Z_j = \max \{ 1 \cdot 100, 10 \cdot 50, 2 \cdot 100, 3 \cdot 100, 20 \cdot 50, 15 \cdot 50, 10 \cdot 100, 5 \cdot 100 \} = 1000 \). Similarly, \( Z \) for nodes \( C \) through \( H \) is given by \( \{ 200, 2000, 1300, 2000, 2000, 2000 \} \). So \( j' \) will be \( B \). So \( M = \{ A, B \} \), \( N = \{ C, D, E, F, G, H \} \). \( Z = 1000 \).
• **Repeat Step 2:** Calculate $Z_j$ for nodes $C$ through $H = \{1000, 1000, 1000, 1000, 1000, 1000\}$. Since there is no such node $j$ that satisfies $(Z - Z_j) > 0$ for $j \in N$, we go to Step 3.

• **Step 3:** The solution is: $M = \{A, B\}$, and $Z = 1000$.

**Heuristic When Storage Spaces Is Constrained**

Below, we present the heuristic procedure (Heuristic 2) for Problem 2.

**Heuristic 2**

- **Step 1:** Let $S$ be the total storage space available before materializing the root view. Let $N$ be the set of all views under consideration, which will initially be all views except the root View $A$. Let $M$ be the set of views to be materialized. Initially, let $M = \{A\}$ where View $A$ is the root view which is required to be materialized and this will be denoted as the current solution. Let $Q$ be the set of views that may not be considered. Initially $Q$ will be empty. Let $w_j$ be the weight associated with view $j$ and let $p_j$ be respectively the number of pages required, and space required to materialize view $j$. Let $S_j$ be the space required to materialize all the views in the set $M$. For each node $j$, calculate $f_j^* = \min_{j \in N - Q} L_j$ (minimum number of pages to be retrieved to answer queries based on view $j$ using only views in the current solution given by set $M$, which has all materialized views). Let the objective function value $Z = \max \{f_j^* \mid j \in N - Q \}$.

- **Step 2:** For each view $j \in N - Q$, calculate $Z_j$, where $Z_j$ is the objective function value if view $j$ were to be materialized in addition to all other views in $M$. Let view $j' \in N - Q$ be such that it maximizes $(Z - Z_j)$ for $j \in N - Q$ and $(Z - Z_j) > 0$. If there is no such $j'$, go to Step 4 else go to Step 3.

- **Step 3:** If there is such a $j'$ and if $S - S_{M'} - S < 0$, that implies that there is not enough space left to accommodate view $j'$. Reset $Q = Q \cup j'$. Go to Step 2.

- **Step 4:** Views to be materialized are given by the set $M$. $Z$ gives the objective function.

**Heuristic 3 and Heuristic 4 for Problems 3 and 4**

Effective greedy heuristics have been reported by Harinarayan et al. (1999) for Problems 3 and 4. Comparative analysis of these heuristics against the optimal solutions has not been presented in their article. Since we have developed linear integer programming models for these two problems, we decided to test these heuristics against the optimal solutions (obtained using LIP 3 and LIP 4 respectively). For testing purposes, we label the greedy heuristics presented in Harinarayan et al. (1999) as Heuristic 3 (for Problem 3) and Heuristic 4 (for Problem 4).

In the next section, we discuss and compare the solutions obtained using Heuristics 1 through 4 with the solutions obtained using LIP 1 through 4 for Problems 1 through 4 respectively.

**EXPERIMENTAL RESULTS**

**Problem Generation Scheme**

We randomly generated three sets of 10 representative instances of data cubes; one with 32 views (i.e., 5 dimensions and one level), one with 64 views (i.e., 6 dimensions and one level), and the other with 128 views (i.e., 7 dimensions and one level). For a given number of dimensions with a unitary level present in each of these dimensions, the number of views present in a given data cube will be given by $2^n$, where $n$ is the number of dimensions. For example, if $n = 3$, the number of views present in a data cube will be $2^3 = 8$ views (or cuboids). The procedure of generating problem instances in the case of three dimensions (n = 3) is described below in Figure 2.

The root view is labeled as View 1 and it has data summarized by three dimensions. The notation 123 in the node corresponding to
View 1 may be interpreted as if the data have summarized values by its three dimensions, say customer (corresponding to Dimension 1), product (corresponding to Dimension 2), and time of sales (corresponding to Dimension 3). The measure that goes with it may be the dollar value of sales. In the node corresponding to View 2, the notation 12 implies that the data in this node have summarized values by the customer and the product dimensions, but has not been aggregated by the time dimension. The measure corresponding to this view would be total sales for all time periods for a given customer and product.

Next we will find all the possible combinations of two dimensions out of these three dimensions. So the possible combinations are View 2 having data aggregated by Dimensions 1 and 2, View 3 having data aggregated by Dimensions 2 and 3, and View 4 having data aggregated by Dimensions 1 and 3 and all of them are answered from the root view View 1. The numbers in the nodes correspond to the number of pages in the respective aggregations. For example, the number 6,000 in the node corresponding to View 3 represents the number of aggregated pages by Dimensions 2 and 3 in that view.

The next level of nodes corresponds to all possible combinations of one dimension out of the three, which results in the View 5, View 6, and View 7. The last view has just one row which will have data summarized across all the dimensions. In order to create numerous problem instances for this study, the following simulation regiment was used. In general, for any child node, the number of pages was obtained by using the formula, $UN(0,6,1)^*(\text{Min}\{p_i\} i \in K)$, where $K$ is the set of all immediate ancestors of $i$. The weight associated with root view was always set to 1. The weights corresponding to all other nodes were generated from $UN(1,100)$. Using this scheme, problems with any number of dimensions can be generated.

In our experiments, we have solved all three randomly generated problem sets (as mentioned above) for two different view materialization constraints (i.e., number of views to be materialized and storage space), each by applying Heuristics 1 through 4 on LIP 1 through 4 respectively. Please note that Heuristics 3 and 4 assume equal weight of 1.0 for all views in the data cube.

In addition to the synthetic problem instances generated above, we wanted to test the algorithms on real-world inspired databases. TPCH (http://www.tpc.org/tpch/) is a well-known database used for benchmarking commercial database and data warehouse products. We decided to use this as one of our problem generation platforms. We populated a 1-GB TPCH Benchmark database (referred as TPCH database) to test our heuristics. In this case, the root node of the data cube generated using TPCH database has three dimensions, that is, customer, part, and time dimensions and one measure of interest that is sales. We have considered two levels in each of these dimensions. For example, customer data could be grouped by individual customers (C) and by nation (N). Similarly, part data could be grouped by individual part (P) and by part type (T), and time dimension data could be grouped by month (M) and by year (Y). Number of views ($T$) in a data cube with given
dimensions and levels in those dimensions are
given by the formula $T = \prod_{i=1}^{l} (l_i + 1)$, where $l_i$ is
the number of levels associated with dimension $i$, and $n$ is the number of dimensions. This results in
27 views for our TPCH database, as shown in Figure 3 in Appendix I along with the actual
number of pages in each view.

We also wanted to test our algorithm on
another type of data cube with specific characteristics, such as wider variation in the number
pages corresponding to the associated views. We
decided to generate another database to accom-
modate this need. This is referred to as AANS
database. The size of this database is roughly
1.5 GB. The daily transactions in this database
were created randomly from certain predefined
probability distributions. We imposed an ad-
ditional requirement that the number of pages
in a descendent view be significantly less than
the minimum number of pages in the immediate
ancestral views. The dimensions and levels in
each dimension were kept the same as the TPCH
data cube, which ensured an identical structure.
The corresponding data cube with page counts
is shown in Figure 4 in Appendix I.

We then solved one set of five problem
instances each using LIP 1 and 2 and cor-
responding Heuristics 1 and 2. The weights
corresponding to all other views except the root
views, which has a weight equal to one, were
generated from UN(1, 100) as mentioned earlier.
In the next section, we discuss the results of the
experimentation.

**Results**

We solved the linear integer programming
models for Problems 1 through 4 using LINGO
software. Heuristic solutions were obtained by
implementing the heuristics using VB.NET.

Table 3 shows the cost comparison between
the optimal solutions and the solutions obtained
using heuristics for Problems 1 through 4 re-
spectively. For Problems 1 and 3, the number
of views to be materialized was set to 10 for
32-node data cube, 20 for 64-node data cube,
and 50 for 128-node data cube. For Problems
2 and 4, the amount of storage space available
was set to 50% of the total space (which is the
space required if all the views including the
root view were materialized).

As observed in Table 3, Heuristic 1 reached
the optimal solution in 21 out of 30 instances.
In the remaining nine instances, the deviation
of the heuristic solution from the respective
optimal solution varied between 0% - 8.99% for
32-node problem instances and 0% - 52.07% for
128-node problem instances. The performance
of the heuristic may improve with tighter
stopping condition, but at a cost of increase in
processing time.

Heuristic 2 found the optimal solution in
17 out of 30 instances. In the remaining 13
instances, the deviation of the heuristic solution
from the respective optimal solution varied be-
tween 0% - 29.41% for 32-node problem instances
and between 0% - 52.07% for 128-node problem
instances. Furthermore, our experimental eval-
uation of Problem 1 through 4 points to the fact
that the space constrained environment is more
demanding on the heuristic, that is, it seems to
find the optimal solution in fewer instances. One
must note that a space constrained problem is
inherently harder to solve since many combi-
nations of views with different cardinality may
satisfy the space constraint, and all of these
have to be examined as a part of heuristic or
optimal solution process.

In case of Heuristic 3, we set the weights
equal to 1.0 for all views. We found that Heuristic
3 produces solutions within 1% of the optimal
solution in all of the 30 problem instances.
Furthermore, Heuristic 3 found the optimal
solution for 15 of the 30 problem instances.
Based on the size and complexity of problems
tested here, Heuristic 3 seems to be a good
method for solving Problem 3.

However, it is not possible to generalize
this observation for situations where there
are more views present in a data cube and
perhaps more complex dependencies among
those views. Harinarayan et al. (1999) have
presented an upper bound to the extent of the
error. For situations requiring three views to
be materialized (including the base view), this
upper bound is 25%.
<table>
<thead>
<tr>
<th>Heuristics and Problems</th>
<th>Number of Problem Instance Solved</th>
<th>Number of Dimensions</th>
<th>Total Number of Views in the Data Cube</th>
<th>Number of Views to be Materialized</th>
<th>Allowed % of storage space</th>
<th>Range of % Deviation</th>
<th>Avg. of % Deviation</th>
<th>Number of Instances Solved Optimally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heuristics 1 to Problem 1</td>
<td>10</td>
<td>5</td>
<td>32</td>
<td>10</td>
<td>n.a.</td>
<td>0 – 8.99</td>
<td>0.89</td>
<td>9</td>
</tr>
<tr>
<td>Heuristics 2 to Problem 2</td>
<td>10</td>
<td>6</td>
<td>64</td>
<td>20</td>
<td>n.a.</td>
<td>0 – 47.47</td>
<td>8.43</td>
<td>7</td>
</tr>
<tr>
<td>Heuristics 2 to Problem 2</td>
<td>10</td>
<td>7</td>
<td>128</td>
<td>50</td>
<td>n.a.</td>
<td>0 – 52.07</td>
<td>18.08</td>
<td>5</td>
</tr>
<tr>
<td>Heuristics 3 to Problem 3</td>
<td>10</td>
<td>5</td>
<td>32</td>
<td>n.a.</td>
<td>50</td>
<td>0 – 29.41</td>
<td>2.94</td>
<td>9</td>
</tr>
<tr>
<td>Heuristics 3 to Problem 3</td>
<td>10</td>
<td>6</td>
<td>64</td>
<td>n.a.</td>
<td>50</td>
<td>0 – 66.67</td>
<td>16.66</td>
<td>5</td>
</tr>
<tr>
<td>Heuristics 3 to Problem 3</td>
<td>10</td>
<td>7</td>
<td>128</td>
<td>n.a.</td>
<td>50</td>
<td>0 – 52.07</td>
<td>24.75</td>
<td>3</td>
</tr>
<tr>
<td>Heuristics 4 to Problem 4</td>
<td>10</td>
<td>5</td>
<td>32</td>
<td>10</td>
<td>n.a.</td>
<td>0 – 0.93</td>
<td>0.18</td>
<td>5</td>
</tr>
<tr>
<td>Heuristics 4 to Problem 4</td>
<td>10</td>
<td>6</td>
<td>64</td>
<td>20</td>
<td>n.a.</td>
<td>0 – 0.56</td>
<td>0.15</td>
<td>5</td>
</tr>
<tr>
<td>Heuristics 4 to Problem 4</td>
<td>10</td>
<td>7</td>
<td>128</td>
<td>50</td>
<td>n.a.</td>
<td>0 – 0.16</td>
<td>0.05</td>
<td>5</td>
</tr>
<tr>
<td>Heuristics 4 to Problem 4</td>
<td>10</td>
<td>5</td>
<td>32</td>
<td>n.a.</td>
<td>50</td>
<td>0 – 0.70</td>
<td>0.18</td>
<td>3</td>
</tr>
<tr>
<td>Heuristics 4 to Problem 4</td>
<td>10</td>
<td>6</td>
<td>64</td>
<td>n.a.</td>
<td>50</td>
<td>0 – 0.36</td>
<td>0.15</td>
<td>1</td>
</tr>
<tr>
<td>Heuristics 4 to Problem 4</td>
<td>10</td>
<td>7</td>
<td>128</td>
<td>n.a.</td>
<td>50</td>
<td>0.02 – 0.22</td>
<td>0.08</td>
<td>0</td>
</tr>
<tr>
<td>Heuristics and Problems</td>
<td>Database</td>
<td>Number of Problem Instance Solved</td>
<td>Number of Dimensions</td>
<td>Total Number of Views in the Data Cube</td>
<td>Number of Views to be Materialized</td>
<td>Allowed % of storage space</td>
<td>Range of % Deviation</td>
<td>Avg. of % Deviation</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
<td>----------------------------------</td>
<td>----------------------</td>
<td>---------------------------------------</td>
<td>-----------------------------------</td>
<td>---------------------------</td>
<td>----------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Heuristics 1 to Problem 1</td>
<td>TPCH</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>27</td>
<td>5</td>
<td>n.a.</td>
<td>0 – 24.89</td>
</tr>
<tr>
<td>Heuristics 2 to Problem 2</td>
<td>TPCH</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>27</td>
<td>n.a</td>
<td>55</td>
<td>0 – 63.42</td>
</tr>
<tr>
<td>Heuristics 1 to Problem 2</td>
<td>AANS</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>27</td>
<td>5</td>
<td>n.a.</td>
<td>0 – 89.68</td>
</tr>
<tr>
<td>Heuristics 2 to Problem 2</td>
<td>AANS</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>27</td>
<td>n.a</td>
<td>55</td>
<td>0 – 37.50</td>
</tr>
</tbody>
</table>
In case of Heuristic 4, (with weights set to 1.0) again the heuristic solutions are within 1% of the optimal solutions for all 30 problem instances. However, in this case the Heuristic 4 found the optimal solution only in 10% of the number of problem instances as compared to 50% of the number of problem instances solved in case of Problem 3 using Heuristic 3.

Next, we discuss the performance of Heuristics 1 and 2 on problems formulated using real-word inspired databases denoted as TPCH and AANS. Using each database, we created a data cube with 27 nodes using three dimensions and two levels in each dimension. The weights (w) associated with views were generated randomly from UN(1,100). The number of views to be materialized was set to five for Problem 1, and the amount of storage space to 55% of the total space for Problem 2. Then we solved five problem instances for each problem type and data cube combination.

Table 4 details these results. In case of Problem 1 using TPCH data cube, Heuristic 1 reached the optimal solution in two out of five problem instances. The deviation between the costs varied between 0% and 24.89% and the average deviation is noted to be 13.42%. In case of Problem 2 using TPCH data cube, Heuristic 2 reached the optimal solution in two out of five problem instances and the deviation between the costs varied between 0% and 63.42%. The average deviation is noted to be 17.67%.

In the case Problem 1 using AANS data cube, Heuristic1 reached the optimal solution in two out of five problem instances and the deviation between the costs varied between 0% and 89.68%. The average deviation is noted to be 45.29%. In case of Problem 2 using AANS data cube, Heuristic 2 reached the optimal solution in two out of five problem instances and the deviation between the costs varied between 0% and 37.5%. The average deviation is noted to be 16.85%.

Next, we look at the computation times required for finding the optimal and the heuristic solutions. All problems were solved using laptop computer with Intel Pentium III processor, 996 MHz, and 256 of RAM. Table 5 summarizes the average computation time in seconds for finding the optimal and heuristic solutions for Problems 1 through 4. The instances are same as those described in Table 3. The average time to optimally solve a 32-node problem corresponding to Problem 1 was 3.5 seconds, while average for Heuristic 1 was only 0.1 second. For Problem 1 over a 64-node cube, the average computation time was 22 seconds for the optimizing algorithm, while it just took 0.2 seconds for Heuristic 1. Problem 1 over a 128-Node data cube had an average computation time of 215 seconds for the optimizing algorithm, while it just took 0.1 seconds in case of the Heuristic 1. For 32-node and 64-node data cubes, the computation time averages were of the same order of magnitude for Problems 1 through 4. However, for the 128-node data cube, the computation times for the optimizing algorithm for Problems 2 through 4 were relatively lower than that of Problem 1. However, the average time for Heuristics 2 was about 0.1 second, and for Heuristics 3 and 4 it was 2.5 and 4 seconds respectively.

As discussed earlier, the computation time for the optimizing procedure is expected to increase exponentially with problem size and we have the evidence based in Table 5. However, the computation time for the heuristics is only increasing linearly with the problem size for all problem instances and is expected to behave similarly over the wide ranges of problem sizes. Hence, very large problem instances can be effectively solved using these heuristics. In the next section, we conclude this article and suggest future research directions.

CONCLUSION AND FUTURE RESEARCH

Data warehouses are seen as a strategic weapon to gain competitive advantage for businesses. A data warehouse extracts, integrates, and stores the "relevant" information from multiple, independent, and heterogeneous data sources into one centralized data repository to support the information needs of decision makers and knowledge workers in the form of online analytical processing (OLAP). Business analysts
Table 5. Average computation time in seconds to solve the MVS problem instances

<table>
<thead>
<tr>
<th>Total Number of Views in the Data Cube</th>
<th>Storage Space Constraint</th>
<th>Number of Views to be Materialized Constraint</th>
<th>Number of Views to be Materialized Constraint</th>
<th>Storage Space Constraint</th>
<th>Number of Views to be Materialized Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Heuristic 1</td>
<td>Optimal</td>
<td>Heuristic 1</td>
<td>Optimal</td>
<td>Heuristic 1</td>
</tr>
<tr>
<td></td>
<td>Heuristic 2</td>
<td>Optimal</td>
<td>Heuristic 2</td>
<td>Optimal</td>
<td>Heuristic 2</td>
</tr>
<tr>
<td></td>
<td>Heuristic 3</td>
<td>Optimal</td>
<td>Heuristic 3</td>
<td>Optimal</td>
<td>Heuristic 3</td>
</tr>
<tr>
<td></td>
<td>Heuristic 4</td>
<td>Optimal</td>
<td>Heuristic 4</td>
<td>Optimal</td>
<td>Heuristic 4</td>
</tr>
<tr>
<td>32</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>30</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>64</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>128</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Copyright © 2007, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
run complex business queries over the data stored at the data warehouse to mine the valuable information and identify hidden business trends. Results of such queries are generally precomputed and stored ahead of time at the data warehouse in the form of materialized views. This drastically reduces the query execution time to minutes or seconds which otherwise may take hours or even days to complete.

There are many architectural issues involved in the design of a data warehouse. In this context, selecting the “right” set of views to be materialized in a data cube is a major concern. In this article, we have presented two detailed heuristic procedures for solving two bottleneck versions of the materialized view selection problem. In Heuristic 1, the constraint is the maximum number of views that can be materialized, and in Heuristic 2 the constraint is the total storage space available for materialization of views. We also outlined two more heuristics (i.e., Heuristics 3 and 4) from literature for two more versions of the MVS problem. We presented two novel optimization formulations for the bottleneck problems denoted as Problems 1 and 2. We used the standard optimization formulations for Problems 3 and 4. We used these formulations to solve three sets of 10 randomly generated problem instances each with 32-node, 64-node, and 128-node data cubes. We then compared the costs of optimal solutions with the corresponding costs of heuristic solutions.

In order to use these formulations on a wider set of data cubes, we generated a quasicommercial database called TPCH, designed by the Transactions Processing Counsel. Based on this benchmark database, we formulated and solved Problems 1 and 2 over a 27-node data cube derived from this database. The heuristics performed well, thus reiterating its role in such problems. We have also used it on another data cube derived from our own database, denoted as AANS. The results were similar, though the range of deviation between heuristic and optimal solutions for Problem 1 was little higher.

Our findings generally indicate that the heuristics used to solve the problem instances of Problems 1 and 3 are effective and find solutions close to optimal very often. Heuristics for Problems 2 and 4 are less effective, and it reflects the inherent additional complexities of the storage space constrained problems. Even though the Problems 1 and 2 are NP-complete, we were able to solve optimally 128-node problem instances of this type within and average of 4 minutes using our formulation. We also found that the computation time for the heuristics is nearly linear with problem size and considerably less than that of the optimal procedures. Hence, even very large problems can be effectively solved using these heuristics. We observe that in general it takes more time to solve problems with the storage space constraint as it puts higher burden on the system to check each possible combination to find the best set of views to materialize for obtaining optimal solution.

A critical application of these models and heuristics may be found in the design of large data warehouse systems, where one could use these models in an online environment to dynamically change the materialization scheme to suit users’ objectives at a point in time. For example, during peak times, the objective may be to provide a guaranteed level of weighted service for all users, which may be well served by the bottleneck formulations. During times when there is a significant demand on storage space, the storage space constrained models may find an application.

The heuristics may be improved upon as well as validated against real-world data warehouse systems. One could also analyze the relationship between weights and optimal solutions using sensitivity analysis. One can also focus on how to derive these weights more systematically and assign them to respective views. The optimization models and heuristics presented in this article do not consider the view maintenance cost. Further research can focus on incorporating view maintenance cost in the optimization models and the heuristics presented here and validating these heuristics against the real world data warehouse systems.
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.


APPENDIX A.

Figure 3. Data cube lattice with actual number of pages present in each associated cuboid (Source: TPCH Benchmark Database)*

*To keep the diagram simple, some dependencies have not been shown
Figure 4. Data cube lattice with actual number of pages present in each associated cuboid  
(Source: AANS Database)*

*To keep the diagram simple, some dependencies have not been shown