Accommodating Scalability in Warehouse View Selection Problem Using Partial Combined Cube Lattice*

C.I. Ezeife and Anna Meng
School of Computer Science
University of Windsor
Windsor, Ontario
Canada N9B 3P4
cezeife@uwindsor.ca
Tel: (519) 253-3000 ext. 3012; FAX: (519) 973-7093

Abstract

As the number of warehouse dimensions increases, the number of main subviews on a cube lattice increases exponentially. When warehouse dimension hierarchies are taken into consideration in the view selection problem, the total number of subviews on the combined data cube lattice gets even larger. The problem goal is to select an appropriate set of materialized views including dimension views, that minimizes total query response time, given a set of common warehouse queries and some storage constraints.

This paper uses the concept of partial combined cube lattice and warehouse common queries to reduce the number of warehouse views considered in the view-selection problem when the number of dimensions of the data cube increases and dimension hierarchies are included. An algorithm based on a more practical cost model than simply the number of rows in a database table, is also presented for view selection.

Keywords: Data Warehouse Views, Dimension Hierarchies, Partial Combined Cube lattice, Performance benefit.

1. Introduction

Typically, a data warehousing system is designed to support on-line analytical processing (OLAP) and decision support systems, which allow business executives and managers to query huge, integrated data for better and more competitive business decisions. A data warehouse is a collection of subject-oriented, integrated, non-volatile and time-variant data [In96, BS97]. The warehouse data is organized around major subjects of an enterprise and not around its functions as the online transactional processing system (OLTP) does. For example, the OLTP system of an insurance

* This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

company might store customer data on premiums in one database, and customer data on claims in another database, both of which store only current data. An integrated data warehouse for this insurance company which is organized around the major subjects customer, policy, premium and claim, can list the claim made by each customer, the premium paid by each customer for each policy over a period of time. Since data warehouses contain consolidated data, perhaps, from several operational databases over potentially long periods of time, they tend to be orders of magnitude larger than operational databases. Enterprise data warehouses are projected to be hundreds of gigabytes to terabytes in size. The workloads (applications accessing the data warehouse) are quite query intensive with mostly ad hoc, complex queries that can access millions of records and perform many table scans, joins and aggregations. Thus, query throughput and response time are more important than transaction throughput [CD97].

Data warehouse issues include its architecture, algorithms and tools for integrating selected data from multiple databases or other information sources into a single repository [Wi95]. Many queries over data warehouses require aggregate views or summary data, which can be obtained by joining many large tables. The size of the data warehouse and the complexity of queries can cause queries to take very long to execute, and this delay is unacceptable in most decision support systems. One technique for improving warehouse query response time is pre-computation and storing (materialization) of aggregate views. Gray *et al.* [Gretal96] uses the data cube concept to represent all 2^n possible aggregate subviews for an n-dimensional warehouse fact table. However, as changes are made to the data sources, all the warehouse views that depend on this data need to be updated to reflect the changed state of the data sources [MQM97, Zhetal95, Hu97]. When warehouse dimension hierarchies are also considered for materialization, the number of warehouse aggregate views becomes very huge. Storing all these huge warehouse views may not be feasible owing to storage space constraints and increased maintenance cost, as well as query response time.

Since OLAP queries are complex and the volume of data is large, there is need to balance the time-space trade-off in order to make the system usable. Carefully selecting a set of data cube main views, dimension views and indexes to materialize contributes to finding this needed balance between maintenance cost, storage space and query response

time [Ez00]. Thus, given a set of queries and some storage space constraint, the decision on which aggregate views to materialize in a data warehouse to minimize response time and maintenance cost remains a challenging research issue.

1.1 Related Work

Gray *et al*. in [Gretal96] uses the concept of the data cube to present a multidimensional representation of a set of aggregate measures. An *n*-dimensional data cube can be used to represent a data warehouse with *n* dimensional attributes. Each of the 2^n aggregate views of this fact table can be computed by aggregating relevant cells of the data cube and are thus considered the 2^n subviews of the cube. For example, using an example university warehouse with the main fact table *gradepoint* (studentid, course#, term, gpa) which stores the gradepoint average of each course taken by each student for several years for every term, the data cube that stores this fact table information is given in Figure 1.

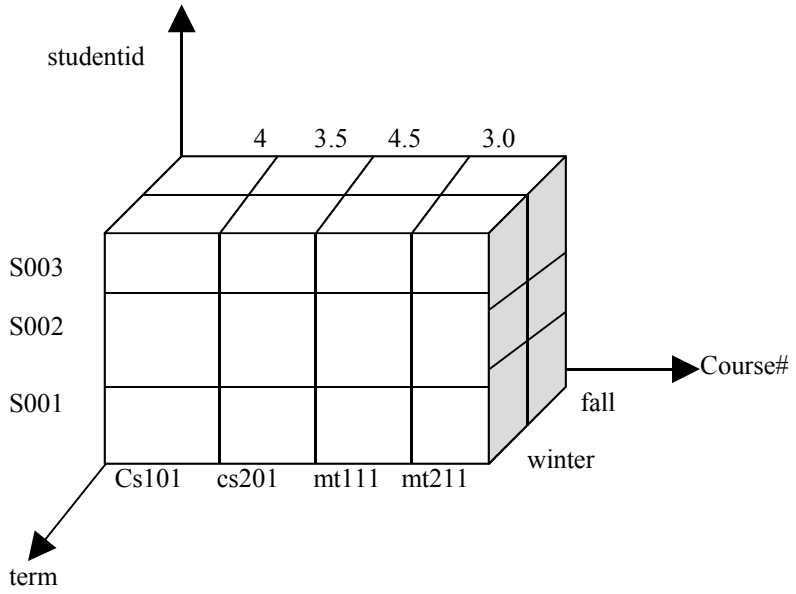


Figure 1: A 3- dimensional datacube for Student, Course and Term

The cube cell with aggregate grade point average (gpa) of 4 represents the tuple of the fact table with student id s003, course# cs101 and for fall term. With this 3-dimensional data cube, the 8 possible subviews are SCT (standing for the view which computes the gpa for each studentid for each course and for each term). This view is the same as represented by the cells of the cube above. The other views are SC, ST, CT, S, C, T and

ALL. Each of these remaining seven views can be computed by adding up relevant cells of the cube above. For example, to compute the average GPA of each studentid for each term (that is view ST), the column filled with data is added up and divided by 4 to obtain the one for S003 for the fall term and other similar columns are also computed the same way. In relational OLAP, all the 2^n subviews of a data cube are stored in a relational table. Since these views are sometimes huge, reducing query response time and maintenance cost is an important research issue.

A lot of algorithms have been proposed for selecting views and indexes of cube views [Ez97, Guetal97, HRU96, LQA97]. Harinayaran *et al.* in [HRU96] defines the relationship between subviews using a lattice framework and defines a greedy algorithm for selecting a set of views of the data cube. The greedy algorithm starts by selecting the top level view into the set *S*. Then, each of the cube views is checked for the one that yields the maximum benefit considering the views that are already in the set *S*. The benefit of a view, *v* not in the set *S* is computed as the (number of rows in the smallest parent *u* of view already in *S* minus the number of rows in *v*) plus the benefit of each descendant view of v in the cube which can be created using view *v*. The benefit of all remaining views in the lattice are computed each time and the view with the highest benefit is included in the set *S*. The direct product of the dimension hierarchies and the fact table yields the combined cube lattice, which includes dimension views (e.g., the view that computes the cumulative gpa for each student for each year (Sy). Gupta *et al*. [Guetal97] extends the greedy algorithm to select both views and indexes. Ezeife in [Ez00] defines a scheme based on the greedy algorithm for selecting views but which fragments every selected view horizontally using application access pattern and frequency. The percentage of all rows of the view accessed on the average by queries and the frequency of their access is used to re-calculate a new size for the partitioned view. The new size is used when making future selection. Most of these view selection techniques work with full cube lattice, a technique that increases computation time drastically if not only main level views are considered but dimension level views as well.

1.2 Contributions

The paper aims at making solutions to view selection problem more scalable as the number of dimensions of the data cube increases using only the partial combined cube lattice which reduces the number of views on the full lattice to be considered for selection. An algorithm is presented which constructs the partial combined cube lattice from a given set of common warehouse queries and the full combined cube lattice. The partial combined cube lattice eliminates views that are not relevant to the warehouse problem using some practical cost/benefit model. The cost/benefit model used overcomes the limitations of the simple cost model adopted by [HaRaUl96] because it incorporates factors such as common warehouse queries, access frequencies, dimension subviews with degrees of table joins needed to compute them if not materialized.

1.3 Outline of the Paper

The rest of the paper is organized as follows. Section 2 presents the discussion of the proposed technique with an example based on a university academic warehouse system. This example shows selected views of the system and some queries as well as how decisions are made regarding the best execution path. Section 3 gives formal presentation of the partial combined lattice and the selection schemes, section 4 discusses some performance justification while section 5 presents conclusions.

2. An Example - Discussion of the Proposed Algorithm

This section gives an example to show how a combined cube lattice is defined from the cube lattice and the dimension hierarchies and it further shows how a partial combined cube lattice can be generated from the combined cube lattice and common warehouse queries. Finally, it presents the working of the proposed view selection scheme, which runs using only partial combined cube lattice. A formal presentation of both the algorithm for constructing the partial combined cube lattice and selecting the views is presented in section 3.

Example 2.1 A simple university database keeps track of the grade point average (gpa) information for students in all the academic terms over a number of years. It has information on students, the courses they have taken and the term in which they took the

courses. Following the star schema, the data warehouse has the following fact and dimension tables:

gradepointaverage (studentid, course#, term, gpa)
student (studentid, name, gender)
course(course#, coursename)
terminfo (term, year, season)

Studentid	Course#	Term	gpa
C0001	CS212	1996W	12
C0001	CS255	1997W	12
C0001	CS255	1998F	8
C0001	CS315	1997F	13
C0001	CS330	1997F	11
C0002	CS254	1996F	12
C0002	CS254	1996W	5
C0002	CS315	1997F	13
C0002	CS330	1997F	11
C0002	CS367	1998W	12
C0003	CS212	1996W	12
C0003	CS254	1996W	11
C0003	CS255	1997W	11
C0003	CS315	1996F	10
C0003	CS315	1997F	13
C0003	CS330	1997F	12
C0004	CS212	1996W	11
C0004	CS254	1996W	12
C0004	CS255	1997W	11
C0004	CS330	1996F	10
C0004	CS330	1997F	13
C0005	CS212	1996W	10
C0005	CS212	1997W	13
C0005	CS254	1996W	12
C0005	CS255	1996F	11

Figure 2: Sample Warehouse Fact table data

The domain of studentid is c0001, c0002, .., c9999. The domain of course# is cs100, cs101, …, cs799. The term in which the courses are offered is recorded as yyyyW, yyyyF or yyyyS to indicate Winter, Fall and Summer terms in each year respectively. The sample fact table is given as Figure 2, while the dimension tables are given in Figure 3 (excluding the dimension table *term*).

Studentid	Name	Gender
C0001	Linda Sharon	F
C0002	Scott Johnson	M

C0003	Edward Green	M
C0004	John Smith	M
C0005	Mark Clark	F

Dimension table student

Course#	Course Name
CS104	Computer Concepts
CS212	C++ Programming
CS254	Data Structures
CS255	File Structures
CS315	Database Management Systems
CS330	Operating Systems
CS367	Computer Networks

Dimension table course.

Figure 3: Warehouse Dimension Tables

In referring to the warehouse fact table attributes, a single upper case letter is used to denote each foreign key attribute as follows: studentid (S), course# (C), term (T). Each attribute of the dimension tables is also denoted with a single lower case letter as follows: name in student (n), gender (g), coursename (d), year (y) and season (o).

2.1 Cube Lattice and Dimension Hierarchies

Assume the average gpa is the measure aggregate we are interested in computing in this data warehouse example, the 3-dimensional data cube for this warehouse has the 2^8 subviews as SCT (standing for average gpa group by studentid, course# and term), SC, ST, CT, S, C, T and ( ). The cube lattice for this data cube is given as Figure 4.

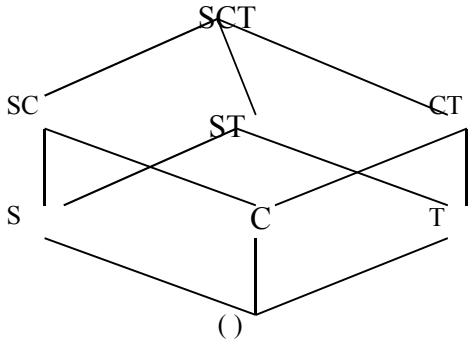


Figure 4: The Main Cube for the University Warehouse

The very top view of the lattice, which has group-by attributes on all the dimension keys is called the base level view. The very bottom one, which has only one row and denoted as ( ) is called the ALL view. These two views can be generated from the following SQL statements:

SCT: create view SCT (studentid, course#, term, avggpa) as

select studentid, course#, term, avg(grade) as avggpa

from gradepointaverage

group by studentid, course#, term;

ALL: create view All (avggpa) as

select avg(grade) as avggpa

from gradepointaverage;

The dimension hierarchies (given as Figure 5), from the dimension tables are used to answer descriptive queries, queries performing drill-down analysis (going from higher level summarization to lower level, e.g., computing grouping of gpa by term from grouping by year), or roll-up (summarizing from most detailed to higher level aggregation) analysis.

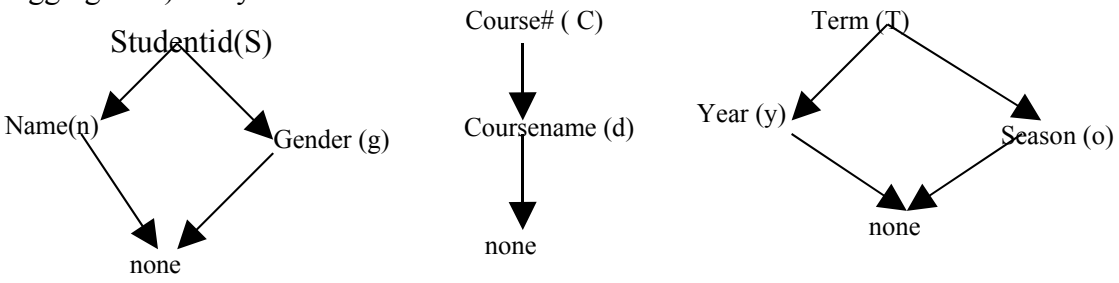


Figure 5: Dimension Hierarchies of the University Warehouse

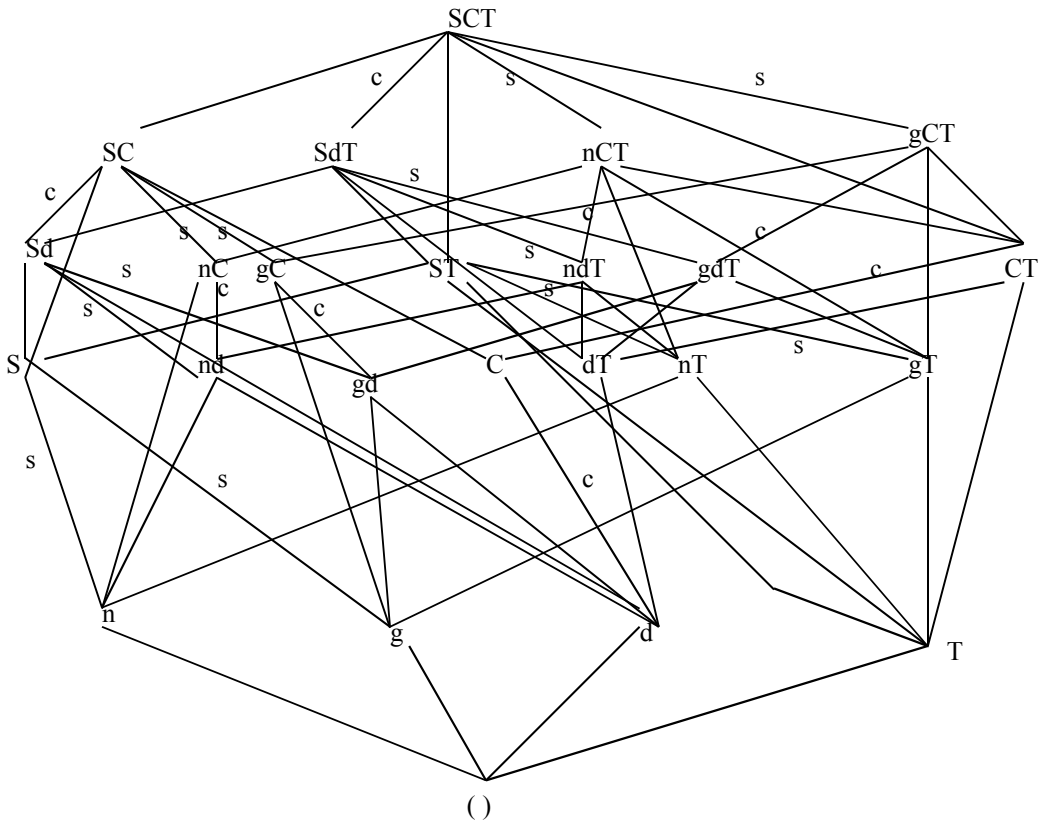
Many queries demand grouping by non-primary key attributes causing the need to join the fact table with dimension tables. For example, the query "what is the average gpa for each student for the courses 'Data Structures' and 'File Structures' in 97W", can be answered with the following SQL query:

Create view *Sn* (studentid, coursename, avg-grade) as

Select studentid, coursename, avg(grade) as avg-grade

```
From gradepointaverage g, course c
Where g.course# = c.course#
And c.coursename in ('Data Structures','File Structures')
Group by g.studentid, c.coursename;
```

This type of view that involves joining the fact and dimension tables is called dimension view, and materializing view *Sn* would save the cost of performing this huge table join. The combined cube lattice is a direct product of the cube lattice for the fact table and the dimension hierarchies [HRUl96]. To construct the combined cube lattice, starting with the base level view (SCT), first list all the possible main subviews which have group-by on the primary key attributes denoted with capital letters, and all the dimension subviews, which have group-by on the non-primary key attributes. Then, connect edges downward from a parent view or node to its child view or node according to the lattices, to either reduce one group-by attribute (e.g., SCT -> SC), or substitute one dimension attribute with the non-primary key attribute (e.g., SCT -> SdT). Repeat the connecting process until all views are processed. If the edge is drawn due to the substitution of a dimension attribute, we label that edge with a corresponding lower case letter of the primary key attribute. The lower case letter on the edge between the parent and child views indicates that a table join is required with the dimension table of the label to derive the child view from the parent view.



S=studentid, C=course#, T=Term, n=studentname, g=gender, d=coursename

Figure 6: The Combined Cube Lattice of the University Warehouse for only dimensions S and C

This lower case letter label is called the table join link. For the sake of simplicity, only the two dimensions *student* and *course* have been combined with the fact table *gradepointaverage* in the university warehouse system to give Figure 6.

Note that the total number of views to consider for selection has grown from 8 to 24 for this 3-dimensional datacube when only 2 dimension hierarchies are considered.

2.2 Constructing the Partial Combined Lattice

Although the total number of views in a cube lattice can be very large, only a portion of them needs to be considered in the warehouse view-selection process due to either storage space constraints or business requirements. To construct the partial combined cube lattice, we identify a set of common warehouse queries. Common warehouse queries have one of the following characteristics: (1) high access frequency (meaning they are accessed most frequently) or (2) a critical or high query value (meaning they require fast response time). Note that some queries that have a critical or high query value and are included in the set of common queries may have low access frequencies. The steps for generating the partial combined cube lattice given a set of common queries and a full combined cube lattice are:

1. Identify and mark all youngest views (excluding the ALL view), which can be used to answer the common queries.
2. From each of the marked views, highlight the edge from this view, v, to its ancestor views including the table join link denoted by a lower case letter, marking all of its ancestor views as well.
3. Repeat step 2 for all the marked ancestor views until no more views are left for marking.
4. Remove all the nodes that have not been marked as well as any dangling edges from the combined cube lattice. The resultant (marked) combined cube lattice is the partial combined cube lattice.

Assume the following six queries constitute the common warehouse queries for our university warehouse example,

Q1: Get the list of all students who have maintained an average gpa of 12 or more in any term. The view needed is ST.

Q2: Get the average gpa for each cs300 level or above course each term (view CT).

Q3: Get the average gpa for each student on each course taken, listed by studentid and course name (view Sd).

Q4: Get the average gpa for each female student on each course name taken in each term (view SdT).

Q5: Get the average gpa for both male and female students on each course name in each term (view gdT).
Q6: Get the average gpa for each student on each course in any term, listed by student name, course number and the term offered (view nCT).

Suppose the above six queries are the common queries accessed in the university data warehouse and which have the following access frequencies:

Q1	Q2	Q3	Q4	Q5	Q6
80	90	100	75	80	85

Since a pre-processing procedure had been used to identify these common queries with these high access frequencies, the veiws that can answer these six common queries are ST, CT, Sd, SdT, gdT and nCT. The partial combined cube lattice is constructed by first writing the youngest common views CT, gdT, ST and Sd at the third level as in the full combined cube lattice. Then, an edge is drawn from each of these views to its ancestor views specifying the join link as seen on the full combined cube lattice. The process is repeated until all common views are connected and all ancestors connected to the root node. The partial combined cube lattice created for the university warehouse is given below as Figure 7. The number beside the views represent the sizes of the view in terms of the number of rows.

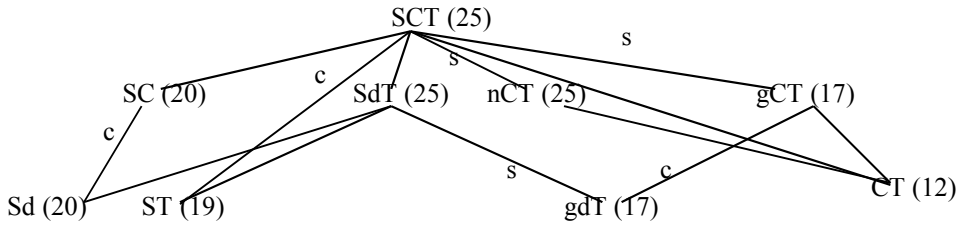


Figure 7: The Partial Combined Cube Lattice of the University warehouse

2.3 **Applying the Proposed View Selection Scheme to the Example Warehouse**

The proposed view selection scheme first improves on scalability using the concept of the partial combined cube lattice and then, for selecting the views to materialize, it proceeds as the greedy algorithm [HRU96] would but considering only views on the partial combined cube lattice, and using a more practical cost/benefit model for computing the benefit of selecting any view. In computing the benefit of selecting a view, the greedy algorithm considers the cost of computing a view as the number of rows in the view. Thus, the view in the lattice with the maximum benefit is the selected view in each selection iteration. The benefit of a view is computed as the difference between the cost of computing its descendant views with it rather than with the smallest view already selected. Another limitation of the greedy algorithm is that each view on the lattice is treated equally when computing its benefit. When calculating the benefit gain of a dimension view, which requires huge table joins, the time required to perform these table joins is not accounted for.

In order to address these limitations, the proposed selection scheme takes into consideration the following factors:

- Common queries and their access frequencies are incorporated
- Candidate views associated with a common query, dimension views associated with a common query as well as the number of table joins needed to answer them are used to decide higher weight factors to be assigned to these views in order to increase the chances of they being selected.

The number of table joins needed to compute a dimension view from its ancestor view is the same as the number of small letter (dimension names) in the view name different from the ancestor view name. For example, view nCT is computed from its ancestor, SCT and would need only one table join because of the *n* letter.

Some of the candidate views on the partial combined cube lattice are associated with a query with known access frequencies while others are not. The following method is used to determine the access frequencies of views not associated directly with a query.

If u is a candidate view on the partial combined cube lattice not associated with a query, and u is the immediate, smallest ancestor of the views v_1, v_2, …, v_n with access frequencies $f(v_1)$, $f(v_2)$, .. $f(v_n)$ respectively. Then, the access frequency of u is the maximum of the frequency set, that is, $f(u) = \text{maximum}\{ f(v_1), f(v_2), .. f(v_n)\}$.

Continuing with the running example, the access frequencies of the candidate views SC, gCT, and SCT are:

F(SC) = max{f(Sd)} = max{100} = 100

F(gCT) = max{f(gdT), f(CT)} = max{80, 90} = 90

F(SCT) = max{f(SC), f(SdT), f(nCT), f(gCT), f(ST)} = max{100, 75, 85, 90, 80} = 100

Assume the least cost view in already selected set S is u. The benefit of a view, v with respect to S, B(v, S) is given as the benefit for computing the view with itself (B_v) plus the total benefit of computing each descendant view (w) of v with v, (B_w). To accommodate other factors, the benefits Bv and Bw have been redefined as follows:

B_v = [cost of (view u) - cost of (view v)] * F(v) * (k + l + 1)

B_w = [cost of (view u) - cost of (view v)] * F(w) * (m-n + 1)

if the cost of v is less than the cost of u, where F(v) is the access frequency of view v, k is the number of joins needed to compute view v with view u, m is the number of joins needed to compute view w with view u, while n is the number of joins needed to compute view w with view v. If the cost of v is bigger than the cost of u, then, B_v and B_w are 0. Finally, the benefit of the view v with respect to set S, $B(v, S) = B_v + \Sigma_{(\text{for all } w)} B_w$

The proposed view selection scheme runs the greedy algorithm using the re-defined method given above for computing the benefits of all views on the partial combined cube lattice. Thus, to select the set of p views from the lattice, it first constructs the partial combined cube lattice, then, it picks the base level view into set S. Next, it proceeds to compute the benefit of each view B_v on the partial combined cube lattice, in each of the

remaining p-1 iterations. The view with the highest benefit is selected during each iteration.

Applying the proposed selection scheme to the university warehouse example using the common queries and the partial combined cube lattice, assume the number of views for materialization in addition to the base level view SCT is 3. The benefits of the views during each of the 3 iterations after selecting the top level view are shown below:

First iteration (base level view selected):

S = {SCT}

Second iteration (pick next view with highest benefit):

$B(SC, S) = B_{Sc} + B_{Sd} = ((25 - 20) * 100 * 1) + ((25 - 20) * 100 * (1-1+1) = 1000$

$B(SdT, S) = B_{SdT} + B_{Sd} + B_{ST} + B_{gdT} = 0$

$B(nCT, S) = B_{nCT} + B_{CT} = 0$

$B(gCT, S) = B_{gCT} + B_{CT} + B_{gdT}$

$= ((25 - 17) * 9 * (1+1)) + ((25-17) * 90 * (0-0+1)) + ((25-17)*80*(2-1+1))$

$=1440 + 720 + 1280 =$ **3440**

$B(Sd, S) = B_{Sd} = (25-20) * 100 * (1+1+1) = 1500$

$B(ST, S) = B_{ST} = (25-20) * 100 * (1+1+1) = 960$

$B(gdT, S) = B_{gdT} = (25-17) * 80 * (2+1+1) = 2560$

$B(CT, S) = B_{CT} = (25-12) * 90 * (1+1) = 2340$

S = {SCT, gCT} because the view gCT with maximum benefit of 3440 is selected in this round.

Third iteration (pick next view with highest benefit):

$B(SC, S) = B_{Sc} + B_{Sd} = ((25 - 20) * 100 * 1) + ((25 - 20) * 100 * (1-1+1) = 1000$

$B(SdT, S) = B_{SdT} + B_{Sd} + B_{ST} + B_{gdT} = 0$

$B(nCT, S) = B_{nCT} + B_{CT} = 0$

$B(Sd, S) = B_{Sd} = (25-20) * 100 * (1+1+1) =$ 1500

$B(ST, S) = B_{ST} = (25-20) * 100 * (1+1+1) = 960$

$B(gdT, S) = B_{gdT} = (17-17) * 80 * (1+1+1) = 0$

$B(CT, S) = B_{CT} = (17-12) * 90 * (1+1) = 900$

S = {SCT, gCT, Sd} because the view Sd with maximum benefit of 1500 is selected in this round.

Fourth iteration (pick next view with highest benefit):

$B(SC, S) = B_{Sc} = ((25 - 20) * 100 * 1) = 500$

$B(SdT, S) = B_{SdT} + B_{Sd} + B_{ST} + B_{gdT} = 0$

$B(nCT, S) = B_{nCT} + B_{CT} = 0$

B(ST, S) $= B_{ST} = (25-20) * 100 * (1+1+1) =$ **960**

$B(gdT, S) = B_{gdT} = (17-17) * 80 * (1+1+1) = 0$

$B(CT, S) = B_{CT} = (17-12) * 90 * (1+1) = 900$

S = {SCT, gCT, Sd, ST} because the view ST with maximum benefit of 960 is selected in this round.

The proposed scheme selects a different set of views than the straight greedy algorithm which selects the set **S = {SCT, gCT, SC, ST}**.

3. Formal Presentation of Proposed View Selection Schemes

This section starts by first presenting some formal definitions of the terms used in presenting the algorithms, then, the algorithms for creating the partial combined cube lattice and selecting the set of views to materialize are presented.

3.1 Definitions

Definition 3.1 *A Common Warehouse Query* is a query with either high access frequency or high query value because it is a time critical query.

■

Definition 3.2 *An m-join subview w of view u,* is a descendant view of u which needs m table joins to be computed from its ancestor view u. ■

Definition 3.3 *Benefit of a view v with respect to selected set S, B(v,S),* is defined as $B(v, S) = B_v + \Sigma_{(\text{for all } w)} B_w$ such that u is the lowest cost view already in S and,

B_v = [cost of (view u) - cost of (view v)] * F(v) * (k + 1 + 1)

B_w = [cost of (view u) - cost of (view v)] * F(w) * (m-n + 1)

if the cost of v is less than the cost of u, where F(v) is the access frequency of view v, k is the number of joins needed to compute view v with view u, m is the number of joins needed to compute view w with view u, while n is the number of joins needed to compute view w with view v. ■

Definition 3.4 *Query Response Time for a query q, (RT_q)* is defined as RT_q = number of rows in (v) + (1+m) * t, where v is the smallest view that can compute q, and m is the number of table joins involved in answering q and t is the unit response time for answering one row of the view.

■

Definition 3.5 *Total Query Response Time for the system (TRT_S)* is the sum of all the query response times for each query in the system. That is,

$RT_q = \Sigma_{(I=1)}^{n} (RT_{qi}) = \Sigma_{(I=1)}^{n}$ (number of rows in (v) + (1+m) * t) ■

3.2 The Formal Algorithm for Creating a Partial Combined Cube Lattice

The steps for creating a partial combined cube lattice is discussed in full with an example in section 2.2. The formal presentation of the algorithm for generating this partial lattice is provided as Figure 8.

Algorithm 3.1 (Partial_CombCubeLattice(Q, L, L')

Input: A set of common warehouse queries $Q_i = \{ Q_1 , Q_2 , \ldots, Q_n\}$
The Combined cube lattice $L_j = \{ V_1 , V_2 , \ldots, V_n\}$

Output: The partial combined cube lattice $L' \subseteq L_j$
begin
$L' = \{\}$
// Identify the set of smallest views $V_j = \{ V_1 , V_2 , \ldots, V_n\}$ that answer the set of common queries $Q_i = \{ Q_1 , Q_2 , \ldots, Q_n\}$, excluding the ALL view //

$V_p = \{ \}$ // for keeping ancestor views //
for each view v in V_j do
V_p = ancestor (v)
$L' = L' \cup V_p$
end
// the resulting lattice is the partial combined cube lattice L' //
end

Figure 8: The Partial Combined Cube Lattice Algorithm

3.3 The Formal Algorithm View Selection - Algorithm DH-Greedy

The steps for selecting a set of views using the proposed algorithm, DH-Greedy is discussed in detail in section 2.3 with an example. It works in a similar way as the greedy algorithm [HRU96] except that it uses the partial combined cube lattice and common queries to cut down on the number of views that will have their benefits computed during each round. Also, the benefit of each view during each round is modified to include access frequency and join factors and the formulas for computing the benefits are given in section 3.1. To select n views, it starts by selecting the top level view in the combined cube lattice into the set S. Then, it calculates the benefit of including each view in the lattice and picks the view with the highest benefit and the process continues until the needed number of views are selected. The formal definition of the algorithm dimension hierarchy greedy (DH-Greedy) being proposed in this paper is given as Figure 9.

Algorithm 3.2 (View-Selection-DH Greedy(Q, F, L)

Input: A set of common warehouse queries $Q_i = \{ Q_1 , Q_2 , \ldots, Q_n\}$
Query access frequencies $F_j = \{ F_1 , F_2 , \ldots, F_n\}$
The Combined cube lattice $L_j = \{ V_1 , V_2 , \ldots, V_n\}$
Number of views to select k
Output: A set of selected views $S \subseteq L_j$
begin
S = {}
// construct partial combined cube lattice //
Lj' = Partial_CombCubeLattice (Q_i, L_j, L_j')
Let V_j be the set of all candidate views on the partial combined cube lattice
S = {base level view}
for I = 1 to k-1 do
for each view v in V_j and v not in S do
Compute B(v,S), the benefit of view v relative to S
Let Vs be the view such that B(Vs, S) = Max (B(v, S)
S = S ∪ Vs
end
end
S = S ∪ ALL view
end

Figure 9: The View-Selection DH Greedy Algorithm

4. Performance Analysis

In order to compare the performance of the DH-Greedy algorithm using partial combined cube lattice and that of the straight greedy algorithm, an experiment was run on two lattices using both algorithms. The first lattice has 23 views SCT, SC, SdT, nCT, gCT, Sd, nC, gC, ST, ndT, gdT, CT, S, nd, gd, C, dT, nT, gT, n, g, d, and T with access frequencies 90, 75, 80, 90, 85, 70, 55, 75, 85, 90, 65, 45, 40, 55, 50, 55, 55, 85, 30, 40, 30, 55, and 20 respectively. The sizes of the 23 views in tuples are respectively 25, 20, 25, 25, 17, 20, 20, 11, 19, 25, 17, 12, 5, 20, 11, 6, 12, 19, 10, 5, 2, 6, and 6. The views that are common queries among these views are SdT, gCT, Sd, gC, ndT, gdT, CT, gd, nT, n, g, d, and T. When the greedy algorithm was run on the full combined lattice of this example case on one hand, and the DH-greedy algorithm was run on the same example starting with its partial combine cube lattice, for selecting a number of views k, the results obtained from the two schemes are summarized in Table 1.

Number selected	Greedy Algorithm	DH-Greedy algoirthm
4	S={SCT, gCT, S, CT}	S={SCT, gCT, n, gC}
5	S={SCT, gCT, S, CT, SC}	S={SCT, gCT, n, gC, Sd}
6	S={SCT, gCT, S, CT, SC, gC}	S={SCT, gCT, n, gC, Sd, nT}
7	S={SCT, gCT, S, CT, SC, gC, ST}	S={SCT, gCT, n, gC, Sd, nT, CT}
8	S={SCT, gCT, S, CT, SC, gC, ST, C}	S={SCT, gCT, n, gC, Sd, nT, CT, d}
9	S={SCT, gCT, S, CT, SC, gC, ST, C, gT}	S={SCT, gCT, n, gC, Sd, nT, CT, d, SC}

Table 1 : View Selected by Greedy and DH-Greedy Algorithms

The total query response times when different sizes of views are materialized are calculated using the query response time formula provided in section 2, and the results obtained are given as Table 2 below.

Number selected	Total Query Response Time (T1).. Greedy alg.	Total Query Response Time (T2).. DH-Greedy	Improvement Ratio=(T1-T2)/T1 * 100%
4	395 tuples	381 tuples	3.54
5	385 tuples	361 tuples	6.23
6	365 tuples	318 tuples	12.88
7	353 tuples	308 tuples	12.75
8	343 tuples	294 tuples	14.29
9	341 tuples	294 tuples	13.78

Table 2: Query Response Times of Greedy and DH-Greedy Algorithms

From the results, it can be seen that the proposed algorithm provides consistently better total query response time than the greedy algorithm with an average improvement of approximately 10.68%. Obviously the gain depends on a number of factors including number of dimension views in the lattice, access frequencies of common queries and the number of common queries and thus would vary with different cases.

5. Conclusions and Future Work

This paper contributes by improving on the scalability of warehouse view selection algorithms through the concept of partial combined cube lattice, while incorporating real life factors like access frequencies of queries and common warehouse queries in the selection process to make it more practical. In order to accomplish the paper's objective,

an algorithm is presented for constructing the partial combined cube lattice from a given set of warehouse common queries and full combined cube lattice.

As the number of warehouse dimensions increase, the number of main subviews on a cube lattice increases exponentially. Generating the partial combined cube lattice allows consideration of only the relevant views (main and dimension views) for materialization. Warehouse common queries with high access frequencies and high query values are used in deciding relevant views and this information, in addition to the number of joins necessary to compute a dimension view from its ancestor view, are included in the benefit of a view when considering that view for selection. Future work might consider extending the approach to handle selection of warehouse indexes as well as investigate further the benefits of constructing the partial combined lattice.

References

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An Overview of Dat Warehousing and OLAP Technology. In *Sigmod Record,* Vol. 26, No. 1, March 1997.

[BS97] Alex Berson and Stephen Smith. Data Warehousing, Data Mining and OLAP. *McGraw-Hill* 1997.

[Ez97] C.I. Ezeife. A Uniform Approach For Selecting Views and Indexes in a Data Warehouse. In *Proceedings of the 1997 International Database Engineering and Applications Symposium, Montreal, Canada*, IEEE publication, Aug. 1997.

[Ez00] C.I. Ezeife. Selecting and Materializing Horizontal Partitioned Warehouse Views. Revised and re-submitted to the *Elsevier journal of Data and Knowledge Engineering*, Jan. 2000.

[Gretal96] J. Gray, A. Bosworth, A.Layman, and H.Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Proceedings of the 12th International Conference on Data Engineering*, pp. 152-159, 1996.

[Guetal97] Himanshu Gupta, Venky Harinarayan, Anand Rajaraman, and Jeffrey Ullman. Index selection for OLAP. In *International Conference on Data Engineering*, Burmingham, U.K., 1997.

[HRU96] Venky Harinarayan, Anand Rajaraman, and Jeffrey Ullman. Implementing Data Cubes Efficiently. In *ACM SIGMOD International Conference on Management of Data*, June 1996.

[Hu97] N. Huyn. Multiple View Self-Maintenance in a Data Warehousing Environment. *Proceedings of the 23rd VLDB conference*, Athens, Greece, 1997.

[In96] W.H. Inmon. *Building the Data Warehouse*. John Wiley Symand [] Sons, Inc. second edition, 1996.

[LQA97] W. Labio, D.Quass and B. Adelberg. Physical Database Design for Data Warehousing. *Proceedings of the International conference on Data Engineering*, Binghamton, UK, April 1997.

[MQM97] I. Mumick, D. Quass, and B. Mumick. Maintenance of Data Cubes and Summary Tables in a Warehouse. *Proceedings of the ACM SIGMOD conference*, Tucson, Arizona, May 1997.

[Wi95] J. Widom. Research Problems in Data Warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM),* November 1995.

[Zhetal95] Y. Zhuge , H. Garcia-Monlina, J. Hammer and J.Widom. View Maintenance in a Warehousing Environment. *Proceedings of the ACM SIGMOD conference*, pp. 316-327, 1995.