A Progressive View Materialization Algorithm

Hidetoshi Uchiyama
Department of Electrical Engineering
and Computer Science
University of Michigan, Ann Arbor
uchiyama@eecs.umich.edu

Kanda Runapongsa
Department of Electrical Engineering
and Computer Science
University of Michigan, Ann Arbor
krunapon@engin.umich.edu

Toby J. Teorey
Department of Electrical Engineering
and Computer Science
University of Michigan, Ann Arbor
teorey@eecs.umich.edu

Abstract

A data warehouse stores materialized views of aggregate data derived from a fact table in order to minimize the query response time. One of the most important decisions in designing the data warehouse is the selection of materialized views. This paper presents an algorithm which provides appropriate views to be materialized while the goal is to minimize the query response time and maintenance cost. We use a data cube lattice, frequency of queries and updates on views, and view size to select views to be materialized using greedy algorithms. In spite of the simplicity, our algorithm selects views which give us better performance than views that selected by existing algorithms.

1 Introduction

In many companies, decision support systems (DSS) play an important role for businesses. The main reason for DSS's popularity is that the companies use it to make a good decision quickly in today's increasingly competitive marketplace. The volume of information available to corporations is rapidly increasing and overwhelming. Those companies which want to effectively manage extremely large volumes of data for business decisions thus realize the significance of using effective DSS.

Data warehouses are historical repositories to store these large volumes of data efficiently, while operational databases maintain state information. Since queries to data warehouses tend to be queries which identify trends in large multidimensional data, these queries typically make large use of aggregations. This leads to a necessity of multidimensional data analysis: *On-Line Analytical Processing*. On-Line Analytical Processing (OLAP) is a technique which guarantees extremely fast response time for multi-dimensional queries in data warehouses. Before OLAP begins accepting queries, it calculates possible aggregations during an off peak period, such as at night. Storing these preaggregations requires extra space. However, for most environments, data warehouses cannot store all possible aggregations because the size of aggregations becomes gigabytes in many companies. In large enterprises, they explode to terabytes.

To solve this problem, this paper proposes a new algorithm that determines which views should be aggregated so that query response time and maintenance cost are minimized.

There has recently been a lot of interest on the problem of selecting views to materialize in a data warehouse. Harinarayan et al. [HRU96] provide algorithms to select views to materialize in order to minimize the query response time, where there is a space constraint. The solution proposed in the paper [HRU96] is a polynomial-time greedy algorithm that performs within a constant factor of an optimal solution. Gupta et al. [GHRU97] extend the work [HRU96] to selection of views and indexes in data cubes. Unlike all the above-mentioned works, Gupta and Mumick [GM99] address the problem of selecting views to materialize under the constraint of a given amount of total view maintenance time. There are other works that solve this problem without any resource constraints and performance guarantees on the quality of the solution [RSS96, YKL97, BPT97, TS97]. Our algorithm simplifies the above models in a practical development time based on the algorithm in [HRU96]. However, our algorithm considers the frequency of selection queries and the cost of update operations on materialized views in order to select views which give us better performance than the algorithm in [HRU96].

The rest of the paper is organized as follows: Section 2 provides the *Progressive View Materialization Algorithm (PVMA)*. Section 3 describes three experiments to show the effectiveness of the PVMA algorithm. Section 4 discusses the conclusion and future work.

2 Progressive View Materialization Algorithm (PVMA)

In data warehousing, it is necessary to distinguish between query and update accesses. The change due to an update affects to all views while a query may need to access to only one view. The benefit and cost metrics that we use in selecting views to materialize are base on the frequency of updates and accesses on each view and the view size. The Progressive View Materialization Algorithm assumes that the selection of each materialized view is done independently. It also assumes that there is no space constraint in the warehouse and that the OLAP uses relational database systems (ROLAP). At each iteration in PVMA, the selected views v_s from the previous iterations is used as a basis. The benefit $benefit_k(v)$ of selecting a view v is considered for all views v that are not in the set of selected views in iteration k. The profit $profit_k(v)$ of a view v is the subtraction of benefit and cost of view v. The view that yields the maximum positive profit is selected to be materialized. This hill climbing

heuristic reduces the complexity of the algorithm. PVMA converges toward a relative maximum as the overall benefit monotonically increases in each iteration. The search terminates when it is not possible to increase the profit further. If we want to have an optimal solution, we need to process through all the paths that have possible benefit to achieve the global maximum profit. This approach is not feasible since the required computation increases exponentially with the branching of possible paths. PVMA is described in the later sections.

2.1 Data Cube Lattice

As discussed in the paper [HRU96], a data cube can be presented by using a lattice. For example, consider that there are 2 dimensions: *Product* and *Region* as a sample. The description of each dimension is described in Table 1. The representation of these 2 dimensions as a lattice is shown in Figure 1. A number at lower left of the box in Figure 1 represents a view. A number at lower right of the box is the frequency of queries on the view. The top view $P1.R1$ corresponds to the fact table.

As can be seen in Table 1 and Figure 1, the size of view $P1.R1$ (1,000,000) is much smaller than the multiplication of $P1$ and $R1$ (7,000×100,000). A data warehouse is usually very sparse, which means that values at many combinations of dimensions are *null*. In this case, the *sparsity* of this data cube is $\frac{1,000,000}{7,000 \times 100,000} = 1.4 \times 10^{-3}$.

Table 1: Dimension *Product* and *Region*

Dimension	Hierarchy	Alias	Number of rows
Product	top	P3	5
	category	P2	1,000
	name	P1	100,000
Region	continent	R4	5
	country	R3	150
	state	R2	1,000
	store	R1	7,000

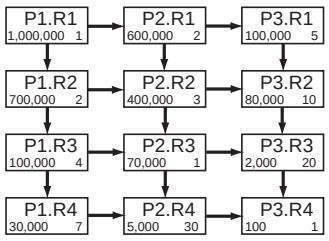

Figure 1: Lattice of data cube

As shown in Figure 1, this lattice is a *directed acyclic graph*. An edge from view v to view u in this graph means that view u can be computed from view v. For example, the view $P2.R2$ can be computed from view $P2.R1$, view $P1.R2$, or view $P1.R1$. The relationship of these views is in *partial order* [HRU96].

Over the past few years, a considerable number of studies have been made on estimating the view size and the cube size [SDNR96, HNSS95]. These studies enable us to form the data cube lattice.

The frequency of queries and updates on each view can be kept track. When the data warehouse accepts queries, it analyses these queries and decides which view is used. If it stores this information, the frequency can be finally calculated. In the following sections, let $R(v)$ be the number of rows in view v, and f_v be the frequency of operations on view v.

2.2 Update Operations

Among database update operations, we consider insert, delete and update in the following sections. Because data warehouses have few update operations in daytime, the data warehouse administrator has a plan to aggregate or change the data warehouse during off peak period. Therefore we can get the information about update operations, their frequencies and the number of updated rows. Here, we define these operations in Table 2.

Table 2: Symbols for update operation i

Operation	Frequency	Number of updated rows
Insert I_i	f_{I_i}	N_{I_i}
Delete D_i	f_{D_i}	N_{D_i}
Update U_i	f_{U_i}	N_{U_i}

2.3 Nearest Materialized Parent View (NMPV)

Before we discuss the benefit and cost that we use in the Progressive View Materialization Algorithm, let us introduce a notion *Nearest Materialized Parent View (NMPV)*. We say that a view u is a parent of view v if view v can be computed from view u. The view $NMPV_k(v)$ is a materialized view u where the difference between the size of view v and the size of view u is minimum among all materialized views in iteration k. That is,

$$NMPV_k(v) = \min(R(v_s), R(v)) \quad \forall v_s \in S \text{ , where } v_s \stackrel{*}{\rightarrow} v$$

where $v_s \stackrel{*}{\rightarrow} v$ means that view v_s is one of the parents of view v, and S represents a collection of views which the PVMA algorithm decides to materialize.

Consider the following example. In Figure 1, assume that $S = \{P1.R1, P2.R1, P1.R3\}$. In this case, $NMPV(P2.R3) = P1.R3$ because $P1.R3$ has been already materialized, is a parent view of $P2.R3$, and has the minimum difference of the number of rows. Note that S contains $P1.R1$ for any k because $P1.R1$ represents the fact table.

2.4 Benefit Calculation

If a view v is materialized, view v and the children of view v receive benefits. This is because if view v is materialized, we also reduce response time for aggregating view v, as we respond to the user. In addition, queries which access the child views of view v can aggregate the necessary view from view v which is smaller than the fact table. Let S be a set of materialized views. The benefit $benefit_k(v)$ of view v in iteration k is:

$$benefit_k(v) = \left(\frac{(R(NMPV(v)) - R(v))}{bf} \sum_{p \in child(v) \cup v} f_p \right) T_{rba}$$

where T_{rba} is the time for one random block access, and bf is the blocking factor of the view [Teorey99]. The view $child(v)$ represents child views, whose NMPV is v. That is, the function $child(v)$ is an inverse function of $NMPV$. The

reason that we use random access in benefit calculation is that the OLAP server is shared by many users. Therefore even if the data itself is contiguous in the disk, it is accessed randomly.

2.5 Cost Calculation

Each change to the fact table in OLAP involves update to each view - all dimensions and all levels of each dimension are affected. This can cause large overhead. We recommend that when aggregated views are created, their primary key indexes are created as well. We assume that their primary key indexes are implemented as B^+trees, where all the non-leaf nodes are in main memory and only the leaf nodes and data are on disk. We derive the time estimation for each update operation as follows, where f represents the frequency of each operation on a view.

Insert If one row is inserted to the fact table, materialized views are affected. In this case, the following disk accesses are necessary: (i) read a block of a view, (ii) rewrite a block of the view, (iii) read a leaf node (block), and (iv) rewrite the leaf node. Thus total cost for the insertion of one row is $4f$ (rba).

Delete The following disk accesses are necessary in delete operation: (i) read a leaf node, (ii) read a block of a view, (iii) rewrite a block of the view, and (iv) rewrite the leaf node. Thus total cost for the deletion of one row is $4f$ (rba).

Update The following disk accesses are necessary in update operation: (i) read a leaf node, (ii) read a block of a view, and (iii) rewrite the block of a view. Thus total update cost for one update is $3f$ (rba).

Cost Calculation We can derive a formula of the cost of update operations on view v as:

$$cost(v) = \left(\sum_{i \in I} 4N_i f_i + \sum_{d \in D} 4N_d f_d + \sum_{u \in U} 3N_u f_u \right) T_{rba}$$

where I, D and U respectively represent a set of insert, delete and update operations, shown in section 2.2.2. T_{rba} is the time for one *random block access*.

The cost of update operations on each view does not change in any iterations because even if some views have already been materialized, they do not affect the other views. In addition, the costs of update operations on all views are the same because the formula does not include any information about view v.

2.6 Profit Calculation

PVMA calculates benefit and cost for each view in each iteration. The function $profit_k(v)$ which is the profit of view v, and is defined as:

$$profit_k(v) = benefit_k(v) - cost(v)$$

In each iteration k, the algorithm chooses the view which has the maximum profit. This means that the view should be materialized because if the view is materialized, OLAP receives the maximum benefit minus cost. Conversely, views which have negative profit are not considered any more. Since the value of $benefit_k(v)$ decreases as k increases, profit also decreases. Therefore if the profit becomes negative once, it cannot be positive again. The algorithm is described below.

$S = \{v_1\}$ **(S: views which should be materialized)**
$NR = \phi$ **(NR: views which should not be materialized)**
$cost = \left(\sum_{i \in I} 4N_i f_i + \sum_{d \in D} 4N_d f_d + \sum_{u \in U} 3N_u f_u \right) T_{rba}$
FOR k=1 to $|V|$ **($|V|$: total number of views)**
BEGIN
  FOR all views v
  BEGIN
    IF $(v \notin S$ *&* $v \notin NR)$ *THEN*
      $benefit_k(v) = \left(\frac{(R(NMPV(v)) - R(v))}{bf} \sum_{p \in child(v) \cup v} f_p \right) T_{rba}$
      $profit_k(v) = benefit_k(v) - cost(v)$
      IF $(profit_k(v) < 0)$ *THEN Add v into NR*
    END
  END
  $Bview = \phi$ **($Bview$: a view with maximum profit)**
  $Bvalue = 0$ **($Bvalue$: profit of $Bview$)**
  FOR all views v
  BEGIN
    IF $(v \notin S$ *&* $v \notin NR)$ *THEN*
      IF $(Bview = \phi)$ *THEN*
        $Bview = v$, $Bvalue = profit_k(v)$
      IF $(Bvalue < profit_k(v))$ *THEN*
        $Bview = v$, $Bvalue = profit_k(v)$
    END
  END
  Add Bview to S
END

2.7 Complexity of the Algorithm

Let, V be the number of views;
I be the number of insert queries;
D be the number of delete queries;
U be the number of update queries.

Then the worst case complexity of PVMA is $O(V^2 + I + D + U)$.

3 Experiment

In this section, we would like to show a series of experiments to clarify the differences between PVMA and the algorithm in [HRU96].

3.1 Progressive View Materialization Algorithm (PVMA)

We carried out the experiments to evaluate the PVMA algorithm. All experiments in this section are carried out using Pro*C with Oracle8 server on Sun Microsystems Solaris operating systems. This experiment was conducted under the following conditions.

Lattice	Figure 1
Update operations	Table 3
Block size	64 kbytes
Record size (fact table)	3276 bytes
T_{rba}	16(msec)
Disk system	SEAGATE ST34520A Medalist Pro 4520

Tables 4 show the benefit, cost and profit for each iteration. "S" in the tables means that the view has already been in S (should be materialized), and "NR" means that the view has already been in NR (should not be materialized) in the PVMA algorithm. That is, once the view is in

Table 3: Description of update operations

Operation	Frequency	Updated rows
Insert I_1	1	3,000
Delete D_1	1	3,000
Update U_1	3	1,000
Update U_2	10	100

NR, its benefit and cost should not be calculated. The view $P1.R1$ is in S because it is the fact table. The fact table must be in S because it is the base view used to compute other views. The underlined number in these tables means that the view has the maximum profit among all views in the iteration.

In the fourth step, only one view is caculated and have negative profit. The other views have been in S or NR. Thus the algorithm terminates, and the number of materialized views is 3. As a result, PVMA materializes views $P1.R3$, $P2.R4$ and $P3.R1$.

3.2 Comparison with the Algorithm in [HRU96]

To evaluate the algorithm, we conducted some experiments with a greedy approach, which was proposed in [HRU96]. Because conditions do not match completely, we used conditions shown as follows:

Dimensions	Table 5
Frequency	Assigned 1∼100 randomly
Update operation	Table 6 D: number of dimensions
Lattice	Figure 2 Upper left number: index of the view
Density in fact table	1% (fact table)
Dimension tables	Keys and other attributes are randomly generated.

Table 5: Description of dimension tables

Name	Base level size	Levels	Record size (byte)
Product	100	4	10,000
Customer	50	3	10,000
Region	10	2	10,000
Time	5	2	10,000

Table 6: Definition of update operations

Operation	Frequency	Updated rows
Insert I_1	1	50*D^2
Delete D_1	1	50*D^2
Update U_1	3	50*D^2
Update U_2	10	1*D^2

The materialized views obtained from both the algorithm in [HRU96] and PVMA tended to be very similar. Tables 7, 8 and 9 show which view is materialized in each iteration using both algorithm. The number in a cell represents the index which is uniquely assigned to views. For example, the index in 2-dimensional lattice is 0∼11.

As shown in Table 7, the first 7 materialized views, which are calculated by the algorithm in [HRU96] and PVMA, are

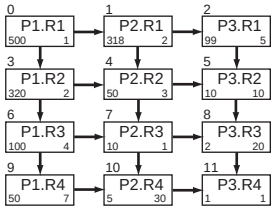

Figure 2: 2-dimensional lattice

Table 7: First 7 materialized views in 2-dimensional lattice

Iteration	1	2	3	4	5	6	7
HRU	4	6	1	3	2	7	9
PVMA	6	4	9	2	3	5	1

Table 8: First 8 materialized views in 3 dimensional lattice

Iteration	1	2	3	4	5	6	7	8
HRU	8	2	6	1	7	3	12	4
PVMA	7	6	9	4	3	2	10	8

Table 9: First 8 materialized views in 4 dimensional lattice

Iteration	1	2	3	4	5	6	7	8
HRU	24	4	12	2	1	6	16	14
PVMA	24	4	16	6	5	3	14	2

very similar although their orders in being selected are different. However, as the number of dimensions becomes higher, this similarity decreases, shown in Tables 8 and 9. This is because lattice with more number of dimensions has more number of views. In addition, the bias of frequencies discriminates the result since PVMA materializes views which have higher frequencies.

Tables 7, 8 and 9 show that the algorithm in [HRU96] and PVMA materialize similar views. Here, we would like to discuss the difference of the overall performance. Figure 3, 4 and 5 show the overall performance of data cube lattice. We measure the performance in terms of the total query response time for all views. Therefore if a view v is not materialized, the nearest materialized parent view $NMPV(v)$ is used for queries of the view v.

As can be seen in Figures 3, 4 and 5, the total query response time when we use PVMA is less than one when we use the algorithm in [HRU96]. However, the total query response time when we use PVMA is obviously better only in the first 8 materialized views. After PVMA and the algorithm in [HRU96] select 8 or more views to materialize, the total query response time of both algorithms is close. The results on this data shows that PVMA selects views that give us smaller response time than views that are selected by the algorithm in [HRU96]. We need to investigate these differences over a wider variety of data sets.

[HRU96] algorithm takes $O(NvV)$ time in the worst case, where Nv represents the maximum number of materialized views and V is the number of views in the lattice. Because Nv is proportional to V, the algorithm in [HRU96] approximately takes $O(V^2)$. Meanwhile, PVMA takes $O(V^2 + I + D + U)$ in the worst case, as discussed in section 2.7. In PVMA, the number of insert, delete and update queries (I, D, U) is usually much less than the number of views in lattice (V). Thus the timing estimation approximately becomes $O(V^2)$. From this discussion, PVMA works in almost

Table 4: Proceeding of PVMA

		First step		Second step		Third step		Fourth step	
	COST	BENEFIT	PROFIT	BENEFIT	PROFIT	BENEFIT	PROFIT	BENEFIT	PROFIT
P1.R1	1,616,000	-	S	-	S	-	S	-	S
P1.R2	1,616,000	18,720,000	17,104,000	3,600,000	1,984,000	1,200,000	-416,000	-	NR
P1.R3	1,616,000	45,360,000	43,744,000	-	S	-	S	-	S
P1.R4	1,616,000	20,488,000	27,872,000	2,128,000	512,000	2,128,000	512,000	392,000	-1,224,000
P2.R1	1,616,000	23,040,000	21,424,000	6,400,000	4,784,000	1,600,000	-16,000	-	NR
P2.R2	1,616,000	31,200,000	29,584,000	6,240,000	4,624,000	1,440,000	-176,000	-	NR
P2.R3	1,616,000	38,688,000	37,072,000	1,248,000	-368,000	-	NR	-	NR
P2.R4	1,616,000	24,676,500	23,060,500	2,356,000	740,000	2,356,000	740,000	-	S
P3.R1	1,616,000	25,920,000	24,304,000	25,920,000	24,304,000	-	S	-	S
P3.R2	1,616,000	22,816,000	21,200,000	22,816,000	21,200,000	496,000	-1,120,000	-	NR
P3.R3	1,616,000	16,766,400	15,150,400	1,647,000	30,400	1,176,000	-440,000	-	NR
P3.R4	1,616,000	799,920	-816,080	-	NR	-	NR	-	NR

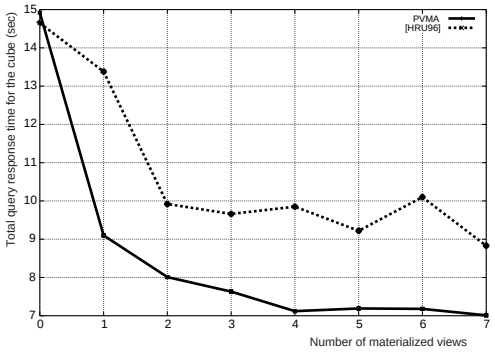

Figure 3: Performance in 2-dimensional lattice

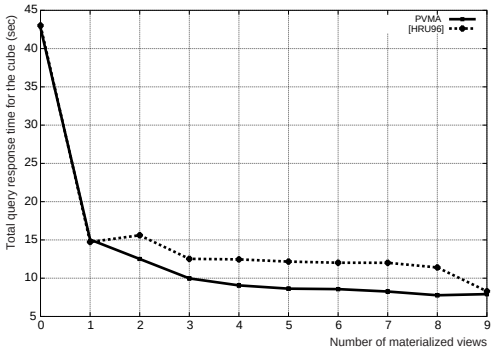

Figure 4: Performance in 3-dimensional lattice

same time as the algorithm in [HRU96].

3.3 Validity of Benefit and Cost Metrics

The PVMA algorithm stops if all non-materialized views have negative profit or all views have already been in S which means they should be aggregated. However, in order to show the effectiveness of PVMA, we did not stop calculation even if the above conditions hold. This experiment is conducted under the following conditions.

Lattice	Figure 1
Update operations	Table 10
Block size	64 Kbytes
Record size	3276 bytes
T_{rba}	16 (msec)

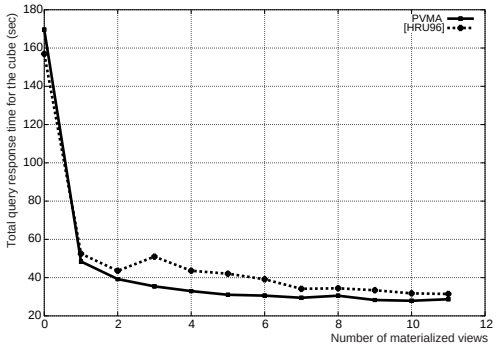

Figure 5: Performance in 4-dimensional lattice

Table 10: Definition of update operations

Operation	Frequency	Number of updated rows
Insert I_1	1	50
Delete D_1	1	50
Update U_1	3	50
Update U_2	10	1

If PVMA does not stop, it chooses views in the following order: $P1.R1$ (fact table) $\rightarrow P2.R2 \rightarrow P1.R4 \rightarrow P2.R1 \rightarrow P2.R3 \rightarrow P1.R3 \rightarrow P3.R1 \rightarrow P1.R2 \rightarrow P3.R2 \rightarrow P2.R4 \rightarrow P3.R3 \rightarrow P3.R4$. If it stops, it terminates calculation at step 3 ($P1.R1 \rightarrow P2.R2 \rightarrow P1.R4 \rightarrow P2.R1$). For each step, we executed selection queries and update operations for the entire database. Figure 6 shows the query response time and total update operation time in each step.

As can be seen in Figure 6, the query response time exceeds the total update operation time from step 0 to step 5. However, after step 6, the update operation time is larger than the query response time. This is because the total update operation time increases as the number of aggregated views increases. In this experiment, if one view is materialized, 0.7509 seconds are necessary to refresh the view when the fact table is changed. Thus the total update operation time is proportional to the number of aggregated views.

Figure 7 depicts the actual profit in PVMA. Here, $profit$ is the time difference between the query response time of iteration k and that of iteration $k+1$. That is, $profit$ shows the saved time when one view is materialized in iteration $k+1$. As Figure 7 indicates, the profit in iteration 3 is the highest point. After iteration 4, the profit decreases as the number of materialized views increases. This means that the

cost of the view exceeds the benefit of the view. Therefore we should stop the algorithm in iteration 3. As discussed above, the PVMA algorithm terminates calculation in iteration 3. This shows that the PVMA stops at the appropriate iteration.

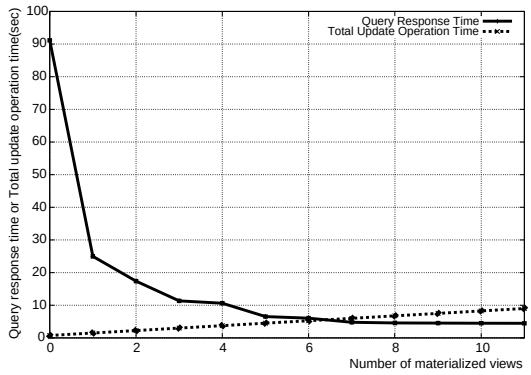

Figure 6: Query response time and total update operation time

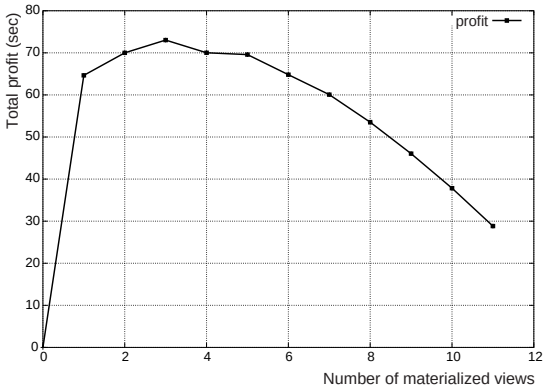

Figure 7: Profit of actual data cube

4 Conclusion

In conclusion, the algorithm in [HRU96] is simpler and would sometimes be the preferred method in simple cases because the Progressive View Materialization Algorithm and the algorithm in [HRU96] often give similar results, discussed in section 3.2. However, in large lattice, especially more than 3 dimensions, the Progressive View Materialization Algorithm provides significantly better results, especially when we have a constraint of small maximum number of materialized views. Our algorithm is also far better in situations which involve different frequencies. The algorithm in [HRU96] does not consider the update cost, whereas PVMA does.

Further research needs to be carried out to derive more accurate formulas of update operations. Our algorithm assumes that all non-leaf nodes of B^+index are stored in main memory. This may not be true in an environment where the OLAP server has small main memory, because the data warehouse tends to have large volumes of data. The cost formula needs to consider this situation.

When a database administrator inserts periodic data, more than one row are inserted at a time. However, our algorithm assumes that the aggregated views are updated when one row is inserted to the fact table. Therefore the insertion formula should be corrected to consider this situation.

There may also be sensitivities to selecting queries which do not require scanning of views by instead using indexes, or queries which have join. Our model does not deal with these factors. In our algorithm, there is an assumption that all queries scan all rows in a view. This is not true if the view has indexes in the primary key and the selectivity of the query is very low. Also, our algorithm works under no space constraint since disk space is adequate and not expensive in many situations.

References

[BPT97] E. Baralis, S. Paraboschi, and E. Teniente. Materialized View Selection in a Multidimensional Database. In *Proceedings of the 23rd VLDB Conference*, 1997.

[GBLP95] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Microsoft Technical Report No. MSR-TR-95-22.

[GHRU97] H. Gupta, A. Harinarayan, A. Rajaraman, and J. D. Ullman. Index Selection for OLAP. In *Proceedings of 13th ICDE,* 1997, pp. 208-219.

[GM99] H. Gupta and I. S. Mumick. Selection of Views to Materialize under a Maintenance-Time Constraint. In *Proceedings of 8th ICDT*, 1999.

[HNSS95] P. Hass, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-Based Estimation of the Number of Distinct Values of an Attribute. In *Proceedings of the 21st VLDB Conference*, 1995, pp. 311-320.

[HRU96] V. Harinarayan , A. Rajaraman , J. D. Ullman. Implementing Data Cubes Efficiently. In *Proceedings of the ACM SIGMOD* 1996, pp. 205-216.

[RSS96] K. Ross, D. Srivastava and S. Sudarshan. Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. In *Proceedings of the ACM SIGMOD*, 1996, pp. 447-458.

[SDNR96] A. Shukla, P. Deshpande, J. Naughton and K. Ramasamy. Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies. In *Proceedings of the 22nd VLDB Conference*, 1996.

[Teorey99] T. Teorey. *Database Modeling and Design.* Third edition, Morgan Kaufmann Publishers, Inc., 1999.

[TS97] D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proceedings of the 23rd VLDB Conference,* 1997, pp. 126-135.

[YKL97] J. Yang, K. Karlapalem and Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *Proceedings of the VLDB*, 1997.