

View Relevance Driven Materialized View Selection in Data Warehousing Environment

Satyanarayana R Valluri*

Soujanya Vadapalli*

Kamalakar Karlapalem

IIIT, Gachibowli, Hyderabad-500 019 INDIA

satya@gdit.iiit.net

soujanya@gdit.iiit.net

kamal@iiit.net

Abstract

A data warehouse uses multiple materialized views to efficiently process a given set of queries. These views are accessed by read-only queries and need to be maintained after updates to base tables. Due to the space constraint and maintenance cost constraint, the materialization of all views is not possible. Therefore, a subset of views needs to be selected to be materialized. The problem is NP-hard, therefore, exhaustive search is infeasible. In this paper, we design a View Relevance Driven Selection (VRDS) algorithm based on view relevance to select views. We take into consideration the query processing cost and the view maintenance cost. Our experimental results show that our heuristic aims to minimize the total processing cost, which is the sum of query processing cost and view maintenance cost. Finally, we compare our results against a popular greedy algorithm.

Keywords: data warehouse, views, selection, materialization, heuristic algorithm, view relevance.

1 Introduction

A data warehouse is a repository of integrated information available for OLAP querying and analysis. It contains multiple views where a view is a derived relation defined in terms of base (stored) relations. When these views are defined over overlapping portions of the base relations, it would be more efficient to materialize commonly shared views rather than materializing all the views.

Materialized views are the derived relations, which are stored as relations in the database. Materialization of views is one of the classical problems in data warehousing. Materialized views can be used for reducing the query response time. Because of the query intensive nature of data warehousing, materialized views approach is quite promising in efficiently processing the queries.

A data warehouse stores materialized views derived from one or more sources for purpose of efficiently implementing decision support or OLAP queries [2].

When a base relation is updated, all its dependant materialized views have to be updated in order to maintain the consistency and integrity of the database. The process of updating a materialized view in response to the changes in the base relation is called 'View Maintenance' that incurs a 'View Maintenance cost'. We apply our View Selection algorithm to obtain the optimal set of views to materialize using query processing plans.

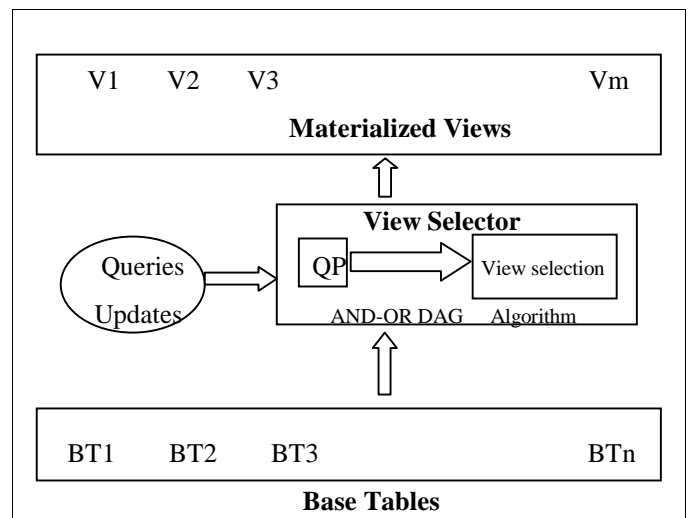


Figure1: Materialized View Selection Process

In the above architecture the view selector interacts with the query processor (QP). Based on the query processing plan it applies the notion of view relevance to select the views for a given set of queries.

The view selection problem is addressed in [2] and [14]. In [2], the notion of benefit is used to select the views. But when the frequency of updates is high, it fails to

* Currently, fourth year undergraduate students, set to graduate in June 2002.

select the view with least update cost. In [14], the view selection is applied on MVPP (Multiple View Processing Plan). But both these approaches do not take into consideration the relevance of views with respect to each other, that is, how a view fits with an already selected set of materialized views. We developed a novel way of addressing the view selection problem.

1.1 Background: AND-OR Graph

To implement the selection of most useful views, a notion of AND-OR DAG is introduced.

Every possible execution plan for all the queries forms an **AND-DAG**. All the possible plans together become the **AND-OR DAG**.

The union of individual **AND-OR DAG** of each query forms an AND-OR view graph. It consists of a **set of Equivalent Nodes** and a **set of Operational Nodes**. Thus the global AND-OR View Graph represents the all-possible ways of answering a given set of queries.

An **equivalent node (eq node)** in the DAG represents the equivalence classes of logical expression that generate the same result set, each expression being defined by a child operation node of the equivalence node and its inputs.

An **operation node (op node)** in the DAG corresponds to an algebraic relational operation such as 'join', 'select'. It represents the expression defined by the operand and its inputs.

The children of an Eq node are one or more Op nodes and the children of an Op node can be one or two Eq nodes. The existence of more than one children at any Eq node indicates an OR node. Every Op node represents an AND node. An example of AND-OR DAG is shown in figure 2.

A global AND-OR DAG is formed from the union of individual AND-OR DAGs of each query.

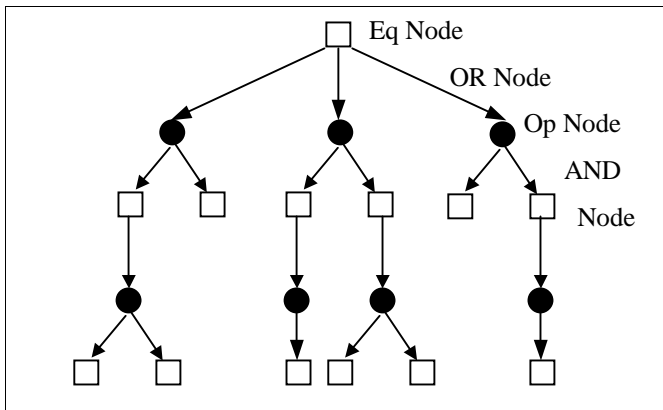


Figure 2: An AND-OR DAG

1.2 Motivation

From our experimentation on greedy algorithm, we observed that its greediness overshadows the update cost. It does not select the views which has minimal update cost. So, there is a need for a heuristic, which gives equal importance to both the query processing cost and the update cost. The heuristic that caters to these requirements is **view relevance**. The view relevance selects the best-fit view with the existing set of materialized views. It does this by observing how the query processing cost and the view maintenance cost vary if this view is included in the set of materialized views. The complexity of the view relevance algorithm is Kn^2 , which is the same as that of the greedy algorithm.

The total cost of a node is the sum of the cost of query processing and the view maintenance cost. In the greedy algorithm, the benefit of a node v is the difference of total cost of the set of materialized views M , and that of the same set of materialized views along with this node included ($M \cup v$). When the access frequencies of the update queries are very high, the update part can get cancelled. So the views selected by the algorithm in this case do not minimize the view maintenance cost. On the other hand, the View Relevance Driven Algorithm selects the views such that the view maintenance cost is minimized. This is because the view relevance cost is the difference of the query processing cost and the view maintenance cost.

1.3 Example

In this section, we will present an example to illustrate the basic idea of the algorithm. This example is taken from [14]. Suppose that the database contains the following relations:

Product (Pid, name, Did)

Division (Did, name, city)

Order (Pid, Cid, quantity, date)

Customer (Cid, name, city)

Part (Tid, name, Pid, Cid, supplier)

We use the short hands **Pd**, **Div**, **Ord**, **Cust**, and **Pt** to stand for the above relations, respectively. Further, we assume that all the relations are at the same site and hence the cost of data communication is not considered for the calculation.

Obviously, there are many ways of answering the set of read-only queries. We can construct a global AND-OR View Graph, which gives all the possible ways of executing the queries from which the cost effective ways of executing the queries are selected. The AND DAG is generated from the AND-OR View Graph. Suppose that the frequent set of updates to be performed and the read-only queries are given in Tables 1 and 2:

UQ1: Update Pd Set Did = 100 Where name = "Computer".	UQ4: Update Cust Set city = "SF" Where name = "Scanner".
---	--

UQ2: Update Div Set city = "LA" Where name = "Printer".	UQ5: Update Pt Set supplier= "IBM" Where name = "Monitor".
UQ3: Update Ord Set quantity = 150 Where date > 7/1/96.	

Table 1: Update Queries

RQ 1: Select Pd.name From Pd, Div Where Div.city = "LA" (O1) and Pd.Did = Div.Did. (O3)
RQ 2: Select Pt.name From Pd, Pt, Div Where Div.city = "LA" (O1) and Pd.Did = Div.Did (O3) And Pt.Pid = Pd.Pid. (O5)
RQ 3: Select Cust.name, Pd.name, quantity From Pd, Div, Ord, Cust Where Div.city = "LA" (O1) and Pd.Did = Div.Did (O3) And Pd.Pid = Ord.Pid (O6) and Ord.Cid = Cust.Cid (O2) Pt.Cid = Cust.Cid (O4) And date > 7/1/96. (O10)
RQ 4: Select Cust.city, date From Ord, Cust Where quantity > 100 (O11) and Ord.Cid = Cust.Cid. (O2)

Table 2: Read-Only Queries

We generate the AND DAG for the above example, which is shown in figure 3. Thus this graph generates the processing plans for queries. The nodes labeled with 'E' are the equivalent nodes and the nodes labeled with 'O' are the operational nodes.

The read-only queries are represented by diamond shapes and labeled 'RQ' and access frequencies shown beside them. The frequencies of the update queries are varied for the experimentation. The sizes of the relations are shown at the bottom of the figure. The sizes of the intermediate nodes (eq nodes) are shown just beside the node label. We set the join selectivity factor to be 0.2 and the selectivity factor to be 0.001 for the example.

1.4 Contributions

In this paper, we introduce the concept of view relevance as a new approach for the view selection problem. The main contributions of the paper are:

- Developing and formalizing the concept of view relevance. A cost model was developed to define view relevance.
- VRDS algorithm: An algorithm that applies view relevance to select materialized views is developed. It chooses a set of views to materialize by examining the AND DAG for a given set of queries.
- We implemented this algorithm and greedy algorithm from [2,5]. We show that for high percentage of updates and space constraints our algorithm outperforms the Gupta's greedy [2,5] algorithm.

2 Framework for Materialized View Selection

In this section we present an algorithm for the selection of views to materialize based on View Relevance.

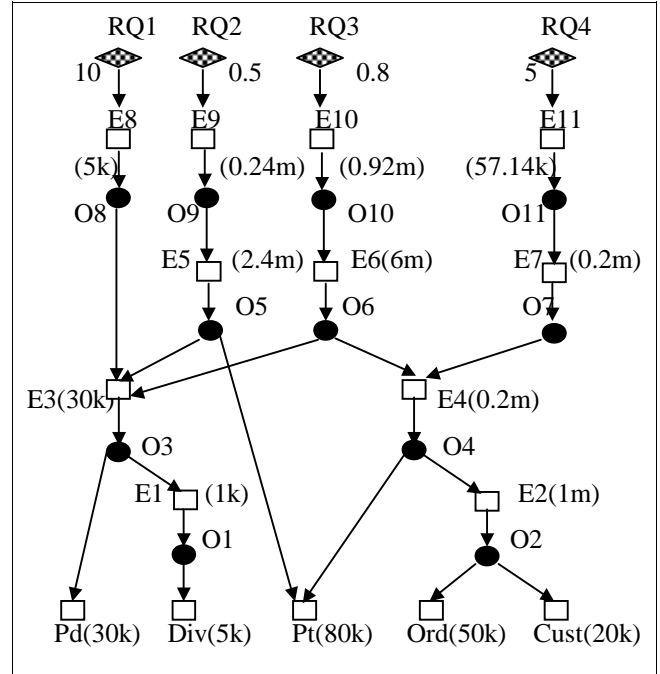


Figure3: AND DAG for the example

2.1 View Relevance

After having obtained the optimal AND DAG, we want to select a subset of the views to materialize. This is based on the **view relevance**, which indicates how the presence of a view in the set, affects the benefit of the other views, thus affecting the total query processing cost and update maintenance cost.

B = Set of Base Tables
N = Set of nodes in MVPP except the base tables
$ B = b$
$ N = n$
RQ = Set of read-only queries
$ RQ = rq$
UQ = Set of update queries
$ UQ = uq$

Table 3: Notations used in formulating View Relevance

In order to define the view relevance of a pair of views we use the following matrices.

<p>BaseView Matrix: This matrix defines the relationship between a view and a base table. It is a $b \times n$ matrix. BV $[B_i][N_j] = 1$, if base table B_i is a child of node N_j. $= 0$, otherwise.</p>										
<p>Connectivity Matrix: This $(n \times n)$ matrix defines whether a node $\in N$, is connected to another node $\in N$. Conn $[N_i][N_j] = 1$, if node N_i is a child of N_j or vice versa. $= 0$, otherwise.</p> <p>QueryUsage Matrix: This $(rq \times n)$ matrix defines whether a read-only query accesses a node $\in N$.</p>										
<p>UpdateBaseUsage Matrix: This $(uq \times b)$ matrix defines whether an update query accesses a base table UBU $[UQ_i][B_j] = 1$, if update query UQ_i accesses the base table B_j. $= 0$, otherwise.</p>										
<p>UpdateUsage Matrix: This $(uq \times n)$ matrix defines whether an update query accesses the node $\in N$. UU $[UQ_i][N_j] = 1$, if update query UQ_i accesses the node N_j. $= 0$, otherwise.</p>										
<p>UpdateFreq Matrix: This $(uq \times n)$ matrix defines the frequency with which an update query accesses a node $\in N$. UF $[UQ_i][N_j] =$ frequency of query UQ_i, if UU $[UQ_i][N_j] = 1$ $= 0$, otherwise.</p>										
<p>$QCost(V_i, M) =$ Cost of formation of the node V_i in the presence of the materialized set of views M.</p>										
<p>$UCost(V_i, M) =$ Cost of maintenance of the node V_i in the presence of materialized set of views M.</p>										
<p>Direct child matrix: This matrix defines what are the direct children of a node. All the nodes, which are directly connected to a node, are called direct children. directchild $[V_i][V_j] = 1$ if V_j is the direct child of V_i. $= 0$ otherwise. It is an array of order $n \times n$.</p>										
<p>Flags Matrix: This $(n \times n)$ matrix defines which view(s) is (are) to be materialized. After every iteration, the View Relevance matrix and the Flags matrix is recomputed. It has four values, each value corresponding to a particular view(s) is (are) to be materialized.</p> <table border="1"> <thead> <tr> <th>Flags $[V_i][V_j]$</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Both V_i and V_j are to be materialized</td> </tr> <tr> <td>1</td> <td>Only V_i is to be materialized</td> </tr> <tr> <td>2</td> <td>Only V_j is to be materialized</td> </tr> <tr> <td>3</td> <td>No View is to be materialized</td> </tr> </tbody> </table>	Flags $[V_i][V_j]$	Action	0	Both V_i and V_j are to be materialized	1	Only V_i is to be materialized	2	Only V_j is to be materialized	3	No View is to be materialized
Flags $[V_i][V_j]$	Action									
0	Both V_i and V_j are to be materialized									
1	Only V_i is to be materialized									
2	Only V_j is to be materialized									
3	No View is to be materialized									

View Relevance Matrix: The view relevance matrix is calculated by the formulae shown in the table. Also, the **Flags** matrix is calculated along with it accordingly. In the table:

$$B(V_i) = \sum_{k|QA[RQk][Vi]=1} QF[RQk][Vi] * QCost(V_i, M) - \sum_{k|UA[UQk][Vi]} UF[UQk][Vi] * UCost(V_i, M)$$

$$B(V_i \wedge \neg V_j) = \sum_{k|QA[RQk][Vi]=1 \wedge QA[RQk][Vj]=0} QF[RQk][Vi] * QCost(V_i, M) - \sum_{k|UA[UQk][Vi]} UF[UQk][Vi] * UCost(V_i, M)$$

$$B(V_i \wedge V_j) = \sum_{k|QA[RQk][Vi]=1 \wedge QA[RQk][Vj]=1} QF[RQk][Vi] * QCost(V_i, M) - \sum_{k|UA[UQk][Vi]} UF[UQk][Vi] * UCost(V_i, M)$$

2.2 View Relevance Matrix

The view relevance is calculated by using various formulae shown in the table 4 that enumerates an exhaustive list of possibilities. Below, benefit of a node is described as the difference of sum of all the queries that access that node and the cost of maintenance of that node. The explanation for each case of View Relevance of V_i and V_j is as follows:

1. If $V_i = V_j$, the view relevance depends on the benefit of V_i alone. Hence the value is the benefit of V_i and the flag value is 1.
2. If V_i and V_j are not connected and neither of them belong to M , the set of materialized views, then the benefit of materializing V_i and V_j is the difference of sum of query processing costs of all the queries which access both V_i and V_j and the maintenance cost of V_i and V_j . Since, both V_i and V_j are to be materialized, its flag is 0.
3. If V_i is materialized and V_j is not materialized and neither of them are connected, then only V_j can be materialized and its benefit is the difference of the query processing cost of queries which access V_j and the cost of maintenance of V_j . So, the value of flag is 2.
4. If V_i is not materialized and V_j is materialized and neither of them are connected. Then V_i can be materialized if it is beneficial. Hence, the flag value is 1.
5. If both V_i and V_j are already materialized, then their view relevance is $-\infty$ and the flag value is 3.
6. If neither V_i nor V_j is materialized and V_i is a child of V_j , then we take the maximum of benefit of V_i and V_j . If the benefit of V_i is maximum, then the flag value is 1. Other wise, V_j is selected for materialization with flag value 2. The same is the true if V_j is a child of V_i .

S.No	Condition	VR [V _i] [V _j]	Flags [V _i] or [V _j]
1	If V _i = V _j	B (V _i)	1
2	If V _i ∉ M, V _j ∉ M, and V _i and V _j are not connected	B (V _i ∧ V _j)	0
3	If V _i ∈ M, V _j ∉ M, and V _i and V _j are not connected	B (V _j)	2
4	If V _i ∉ M, V _j ∈ M, and V _i and V _j are not connected	B (V _i)	1
5	If V _i ∈ M, V _j ∈ M, and V _i and V _j are not connected	-INFINITY	3
6	If V _i ∉ M, V _j ∉ M, and conn [V _j][V _i]=1 Or conn [V _i][V _j]=1	Max (B (V _i), B (V _j))	1 or 2
7	If V _i ∈ M, V _j ∉ M, and conn [V _j][V _i]=1	Max (B (V _i ∧ ¬V _j), B (V _j)) or -INF	3 or 2
8	If V _i ∉ M, V _j ∈ M, and conn [V _j][V _i]=1	B (V _i ∧ ¬V _j)	1
9	If V _i ∈ M, V _j ∈ M, and conn [V _j][V _i]=1	-INFINITY	3
10	If V _i ∈ M, V _j ∉ M, and conn [V _i][V _j]=1	B (V _j ∧ ¬V _i)	2
11	If V _i ∉ M, V _j ∈ M, and conn [V _i][V _j]=1	Max (B (V _i), B (V _j ∧ ¬V _i)) or -INF	1 or 3
12	If V _i ∈ M, V _j ∈ M, and conn [V _i][V _j]=1	-INFINITY	3

Table 4: Formulae to calculate the view relevance matrix

7. If V_i is materialized and V_j is not materialized and V_i is a child of V_j, we first calculate the benefit of V_i with respect to the queries that access V_i but not V_j. We then calculate the benefit of V_j. If the former is greater than the latter, that means V_j need not be considered for materialization. And we would never consider the pair V_i and V_j again. Hence its View Relevance value is made -INFINITY and the flag value is 3. If the former is less than that of latter that means though V_i is the child of V_j and V_i is already materialized, V_j can still be materialized since its benefit is more. Hence, the view relevance value is the benefit of V_j and the flag value is 2.
8. If V_i is not materialized and V_j ∈ M, and V_i is a child of V_j, then we consider the benefit of V_i with respect to the queries that access only V_i but not V_j. If this benefit is high, it means there are many queries, which do not access V_j but access V_i, and we can materialize V_i. The value of the flag is 1.
9. Again, if V_i and V_j belong to M, the view relevance value is assigned as -INFINITY with flag value as 3.
10. If V_i ∈ M and V_j ∉ M, and V_j is a child of V_i, then the View Relevance value is the extra benefit of V_j and the flag value is 2.
11. If V_i ∉ M and V_j ∈ M and V_j is a child of V_i, then we first find the benefit of V_i. Then we find the benefit of V_j with respect to the queries, which access only V_j but not V_i. If the former value is greater than the latter, it means we can materialize V_i and the flag value is 1. Otherwise, if the latter value is high, it

means there is no need for V_i to be materialized. Since, the pair (V_i, V_j) will not be considered again, we will make the view relevance of the combination as -INFINITY and the flag value is 3.

2.3 View Selection

For implementing the View Selection algorithm, we take an array **NodesVisited** to indicate whether a node is visited or not. It is an array of size equal to the number of nodes. All the elements of the array are initially assigned a value of 0. Whenever a node is considered for materialization, its value is changed to 1.

2.3.1 View Relevance Driven Selection (VRDS) Algorithm

The intuition behind the algorithm is as follows: We want to select those views, which help each other. Till now all the view selection algorithms were considering only the most beneficial views, but not the interdependency between the views that are being selected. The relationship between the selected views is important because they help each other in their maintenance. So, the view relevance of a view quantitatively measures how it fits into the already selected views.

The algorithm is given below. Initially, M, the set of materialized views is empty. S (M) is the space occupied by M. The view selection is carried out until all the nodes are marked as visited. In each iteration of selection of views, the view relevance matrix is calculated. The cell with maximum view relevance is found. Its view relevance value is made -INFINITY, since that pair of

views need not be considered again. Its corresponding cell in the **FlagsMatrix** is observed. If its value is 0, it means both V_i and V_j can be materialized. So, according to the space available, we materialize either both of them or only the smaller of the two. If its value is 1 or 2, we materialize V_i or V_j respectively if enough space is left. Also mark them as visited. The **FlagsMatrix** value is also made 3 since the pair V_i and V_j need not be considered again.

```

Given: An optimal AND DAG, and S the space constraint.
M = set of materialized views.
S (M) = sum of size of the views in M.
BEGIN
M =  $\phi$ ;
Initialize all the elements of NodesVisited to zero.
While (All the nodes in the DAG are not visited)
{
    Calculate the view relevance matrix
    and flag matrix.
    Let  $V_i$  and  $V_j$  be the pair of views such that
ViewRelevance [ $V_i$ ][ $V_j$ ] is maximum.
    if (ViewRelevance [ $V_i$ ][ $V_j$ ] < 0.0)
        break;
    Assign ViewRelevance [ $V_i$ ][ $V_j$ ] = -INFINITY.
    Switch (FlagsMatrix [ $V_i$ ][ $V_j$ ])
    {
    case 0:
        mark both  $V_i$  and  $V_j$  as visited.
        if both  $V_i$  and  $V_j$  can be materialized
in the remaining space, then materialize both of them.
        Else materialize the smaller size view
that fits into the space left.
    case 1:
        mark  $V_i$  as visited.
        If space is available, materialize  $V_i$ .
    case 2:
        mark  $V_j$  as visited.
        If space is available materialize  $V_j$ .
    }
    Assign FlagsMatrix [ $V_i$ ][ $V_j$ ] = 3;
    // Do not consider this pair again.
}
return M;
End while;
return M;
END

```

2.3.2 Complexity

Suppose, n is the number of nodes in the AND graph. The order of the while loop in the above algorithm is K , where K is a constant. In every iteration, we go through the elements of the relevance matrix, whose complexity turns out to be n^2 . Hence, the total complexity of the algorithm is Kn^2 , which is the same as that of the greedy

algorithm. Here n is the number of nodes in the AND DAG.

3 Example

For the AND-DAG discussed in the example section (Section 1.3), in the first iteration, the algorithm chooses the views {1, 8}, since the view relevance is maximum for that cell. In the next iteration, the view selected is 3. After that, the maximum view relevance is negative and the selection of views is complete.

We applied Greedy algorithm and View Relevance Driven Selection (VRDS) Algorithm on the input discussed in Section 1.3. With 50% space constraint, and 50% update frequency, the greedy algorithm selects only E8, whereas the VRDS Algorithm selects E1, E3 and E8. Note that in table 8, the values in the View Relevance matrix, after the last iteration, become negative, implying that there are no more relevant views to be considered for materialization and the algorithm terminates. The UpdateFreq matrix will take the corresponding update frequencies wherever there is a '1' in UpdateUsage Matrix, which is shown in Table 9.

Base/ Node	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11
Product	0	0	1	0	1	1	0	1	1	1	0
Division	1	0	1	0	1	1	0	1	1	1	0
Part	0	0	0	1	1	1	1	0	1	1	1
Order	0	1	0	1	0	1	1	0	0	1	1
Customer	0	1	0	1	0	1	1	0	0	1	1

Table5: BaseView Matrix for the example

Query/ Node	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11
RQ1	1	0	1	0	0	0	0	1	0	0	0
RQ2	1	0	1	0	1	0	0	0	1	0	0
RQ3	1	1	1	1	0	1	0	0	0	1	0
RQ4	0	1	0	0	0	0	1	0	0	0	1

Table6: QueryUsage Matrix for the example

Query/ Node	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11
RQ1	10	0	10	0	0	0	0	10	0	0	0
RQ2	0.5	0	0.5	0	0.5	0	0	0	0.5	0	0
RQ3	0.8	0.8	0.8	0.8	0	0.8	0	0	0	0.8	0
RQ4	0	5	0	0	0	0	5	0	0	0	5

Table7: QueryFreq Matrix for the example

Node/ Node	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11
E1	-INF	-INF	-INF	-1.37E+07	-INF	-INF	-INF	-INF	-INF	-INF	-INF
E2	-INF	-INF	-INF	-INF	-6.73E+06	-INF	-INF	-INF	-6.74E+08	-INF	-INF
E3	-INF	-INF	-INF	-1.37E+07	-INF	-INF	-INF	-INF	-INF	-INF	-INF
E4	-1.37E+07	-INF	-1.37E+07	-INF	-6.73E+08	-INF	-4.35E+07	-1.37E+07	-6.74E+08	-1.37E+07	-4.35E+07
E5	-INF	-6.73E+08	-INF	-6.73E+08	-INF	-8.76E+10	-6.73E+08	-5.52E+08	-INF	-8.77E+10	-6.73E+08
E6	-INF	-INF	-INF	-INF	-8.76E+10	-INF	-8.69E+10	-7.09E+10	-8.76E+10	-INF	-8.69E+10
E7	-INF	-INF	-INF	-4.35E+07	-6.73E+08	-8.69E+10	-INF	-INF	-6.74E+08	-8.70E+10	-INF
E8	-INF	-INF	-INF	-1.37E+07	-5.52E+08	-7.09E+10	-INF	-INF	-5.52E+08	-7.10E+10	-INF
E9	-INF	-6.74E+08	-INF	-6.74E+08	-INF	-8.76E+10	-6.74E+08	-5.52E+08	-INF	-8.77E+10	-6.74E+08
E10	-INF	-INF	-INF	-1.37E+07	-8.77E+10	-INF	-8.70E+10	-7.10E+10	-8.77E+10	-INF	-8.70E+10
E11	-INF	-INF	-INF	-4.35E+07	-6.73E+08	-8.69E+10	-INF	-INF	-6.74E+08	-8.70E+10	-INF

Table8: View Relevance Matrix after the last iteration

Query Node	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11
UQ1	0	0	1	0	1	1	0	1	1	1	0
UQ2	1	0	1	0	1	1	0	1	1	1	0
UQ3	0	0	0	0	1	0	0	0	1	0	0
UQ4	0	1	0	1	0	1	1	0	0	1	1
UQ5	0	1	0	1	0	1	1	0	0	1	1

Table9: UpdateUsage Matrix for the example

3.1 Gupta's Algorithm Results

% Space	% Update Freq	Views Selected	Query Proc. Cost	View Maint. Cost	Total Proc. Cost
0.00001	0.1	8,1	1.62E+10	6.61E+01	1.62E+10
0.00001	10	8,1	1.62E+10	7.21E+03	1.62E+10
0.00001	50	8	1.62E+10	6.53E+04	1.62E+10
1	0.1	11,8,4,9,3,2	1.60E+10	2.05E+06	1.60E+10
1	10	11,8,3,2	1.61E+10	7.22E+06	1.61E+10
1	50	8	1.62E+10	6.53E+04	1.62E+10
50	0.1	11,8,4,10,9,3	3.22E+06	2.66E+08	2.70E+08
50	10	11,8,3,2	1.61E+10	7.22E+06	1.61E+10
50	50	8	1.62E+10	6.53E+04	1.62E+10

Table10: Gupta's Algorithm Results

3.2 Relevance Algorithm Results

% Space	% Update Freq	Views Selected	Query Proc. Cost	View Maint. Cost	Total Proc. Cost
0.00001	0.1	1,8	1.62E+10	6.61E+01	1.62E+10
0.00001	10	1,8	1.62E+10	7.21E+03	1.62E+10
0.00001	50	8,1	1.62E+10	6.53E+04	1.62E+10
1	0.1	9,5,11,7,3, 1,8	1.60E+10	2.05E+06	1.60E+10
1	10	2,3,11,7,1, 4,8	1.61E+10	7.24E+06	1.61E+10
1	50	3,8,1	1.62E+10	6.54E+04	1.62E+10
50	0.1	10,9,5,2, 3,11,7,1,4,8	3.22E+06	2.66E+08	2.70E+08
50	10	2,3,11,7,1, 4,8	1.61E+10	7.24E+06	1.61E+10
50	50	3,8,1	1.62E+10	6.54E+04	1.62E+10

Table11: View Relevance Algorithm Results

4 Experimental Results

We have implemented the View Relevance algorithm on the AND DAG discussed in the example. We have compared the results with the greedy algorithm given in [2,5].

4.1 Experiments

We considered a data warehousing environment with four dimension tables and a fact table of varying sizes. We then consider a set of read-only queries and updates. We have varied the ratio of sum of access frequencies of read-only queries to that of update queries from 0.01 to 1 and then from 1 to 50, and have analyzed how the views are selected, the variation in the query processing cost, the view maintenance cost and the total cost. We have also varied the space constraint. We have changed the constraint on the total space (total sum of the sizes of all the views), and observed the variation of views selected.

In the graphs, the X-axis is the percentage of space available for materialized views. The Y-axis shows the cost under analysis. The following two graphs show the variation of total processing cost with percentage of space available with different update access frequencies for two different inputs.

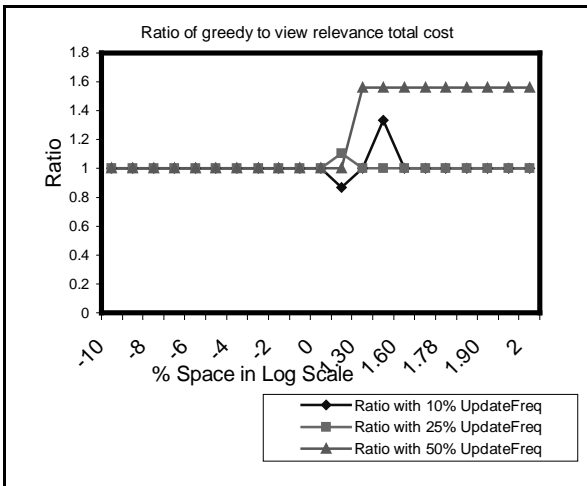


Fig4: ProcessingCost (Greedy)/ProcessingCost (VRDS)

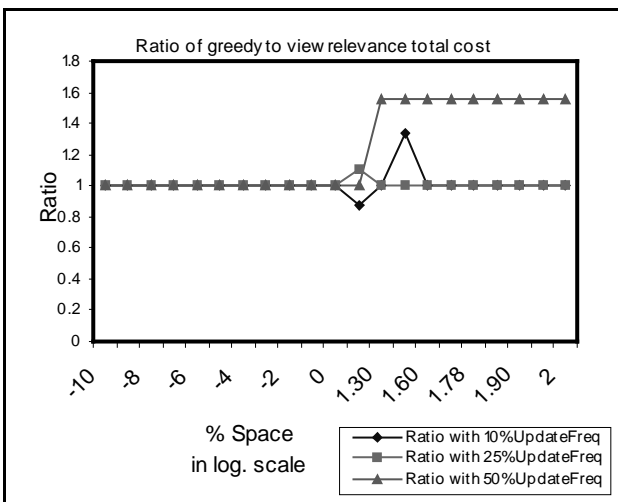


Fig5: ProcessingCost(Greedy)/ProcessingCost(VRDS)

The VRDS algorithm outperforms the greedy algorithm, as the total cost of the selected views by the View Selection algorithm (query processing cost + view maintenance cost) is less than that of the greedy algorithm. The graphs show the performance analysis of both the greedy and heuristic algorithm and also prove the fact that the VRDS Algorithm gives better results when total cost is taken into consideration.

The Fig.6 below shows analysis of total processing cost under 30% space constraint with various update frequencies for a data set.

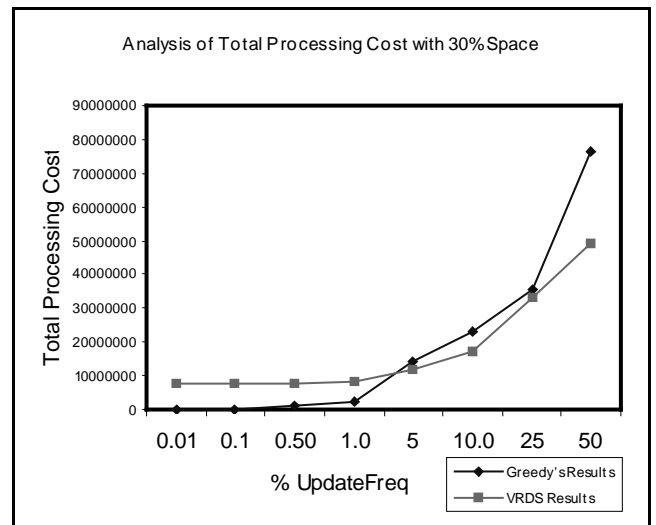


Fig 6: TotalProcessingCost of Greedy Vs. VRDS

Figure 7, Figure 8 and Figure 9 show the variation of query processing cost, view maintenance cost and total cost for another data set.

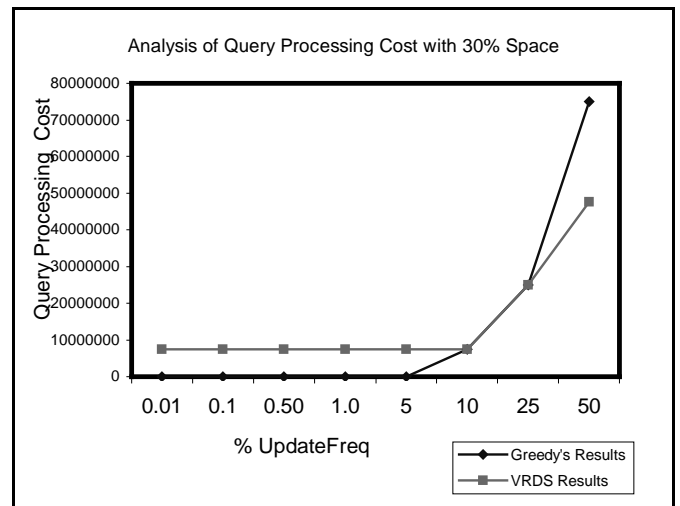


Fig7: Query Processing Cost Greedy Vs. VRDS with 30% Space

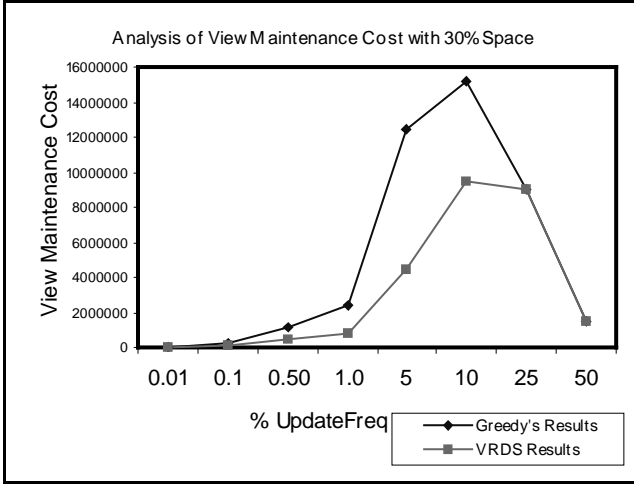


Fig8: View Maintenance Cost with 30% Space

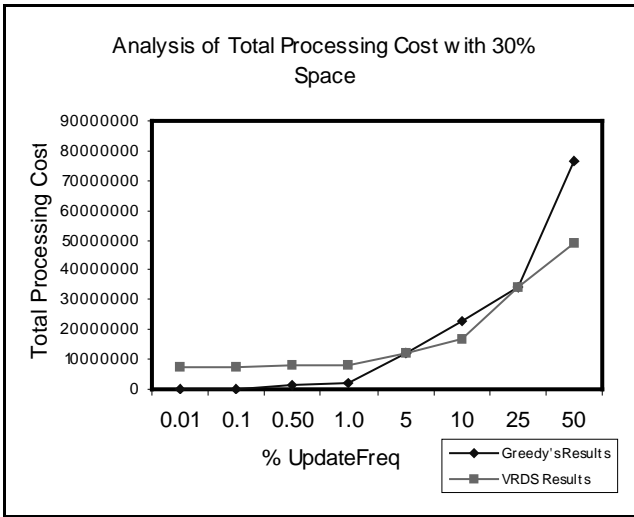


Fig9: Total Processing Cost with 30% Space

4.1.1 Observations

In Figure 4, for the space constraint 1%, there is a decrease in the ratio, implying that the Greedy's cost is less than the View Relevance's cost. This is because; the View Relevance Algorithm is able to accommodate an extra view than the Greedy's Algorithm. Hence, the update cost is slightly high in this case. But as the space for materialization increases, more relevant views are selected which leads to decrease in the total cost. The same is the case in Figure 5. In Figures 6 and 9, with low percentage of updates, Greedy algorithm performs better, but with higher update frequencies, the view relevance selects least cost set of materialized views.

4.2 Comparison with MVPP Algorithm

We have implemented the view selection algorithm given in [14]. The algorithm takes as input a Multiple View Processing Plan (MVPP). The basic MVPP algorithm given does not take into consideration the space constraint. So, we compared the results of greedy

algorithm, VRDS algorithm and MVPP algorithm with no space constraint. That is, we assume we can materialize all the views if they are useful. Though this assumption is not realistic, it serves our purpose of comparing all the three algorithms.

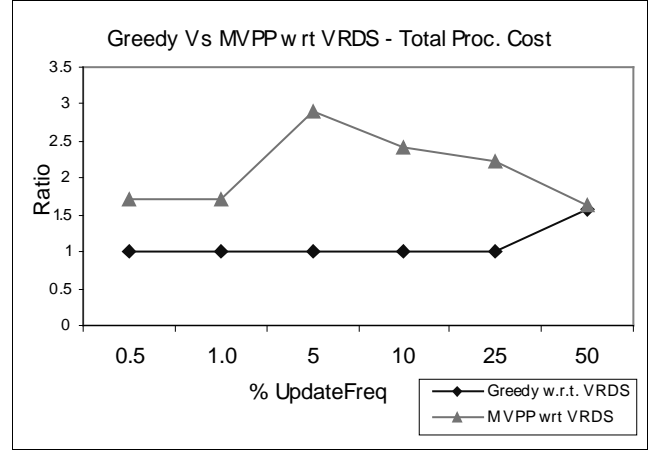


Fig10: Comparison of Total Processing Cost with MVPP.

The MVPP algorithm selects a view using the following formula:

$$C_s = \sum_{q \in O_v} \{ f_q(q) * (C_a(v)_q - \sum_{u \in S_u \cap M} C_a(u)_q) \} - \sum_{r \in I_v} \{ f_u(r) * C_m(v)_r \}$$

where

- 1) O_v denotes the global queries which uses V .
- 2) I_v denotes the base relations which are used to produce V .
- 3) S_v is the set of nodes which are used to produce V .
- 4) $f_q(q)$ is the frequency of (read-only/update) query.
- 5) $C_a(V)_q$ is the cost query q accessing V . $C_a(V)_r$ is the cost of maintaining V based on changes made on relation r if V is materialized.

The algorithm computes C_s for all the nodes. If C_s is positive for a given node, then it is materialized. This is because $C_s > 0$ implies positive benefit if V is materialized. The views are selected based on C_s , and are not limited by space constraint. The selection of views varies with the change in the update frequency of the base tables.

With no space constraint, the greedy algorithm chooses the root nodes when the update frequency is less, whereas the MVPP algorithm chooses the views, which are close to the base relations in the MVPP. The VRDS algorithm chooses the root nodes along with some supplementary nodes, which help in reducing the maintenance cost. From the graph, we observe that the total processing cost is more for MVPP algorithm results when compared to other algorithms. This is due to selection of views without considering view relevance.

MVPP selects a view if it has positive benefit, whereas VRDS selects a view if it reduces the total processing cost, and is relevant to already selected views.

4.3 Discussion

From the above graphs, we can infer that when the frequencies of update queries are very high, the greedy algorithm fails to get an optimal set of materialized views. But the VRDS algorithm selects the set of views such that the total cost is minimized. It is observed that, the greedy algorithm selects that view which has the least update frequency. But the VRDS algorithm selects views that balance the query processing cost and update cost to minimize total processing cost. It is also observed that the greedy algorithm selects relatively fewer views than the VRDS algorithm, even though there is space available. The VRDS algorithm manages to select important small-sized relevant views that reduce the total processing cost with a marginal increase in update cost, and larger decrease in query processing cost. Finally, the complete representation and calculation of view relevance matrix is transparent to the end user. Therefore, it can be incorporated in data warehousing design tool kit.

5 Conclusion

In this paper, we developed a framework based on view relevance and designed a VRDS algorithm for selecting views in data warehousing environments. We compared our results against a greedy algorithm [2,5] and MVPP algorithm [14]. The VRDS algorithm strikes a balance between the query processing cost and the view maintenance cost, whereas, greedy algorithm is focused mainly on updates, and MVPP algorithm on selecting all beneficial views. Our results show that the VRDS algorithm performs better than greedy and MVPP algorithms when there is space constraint and update frequency is high. In our future work, we will extend the view relevance concept for the generation of an optimal AND DAG from a global AND-OR view graph. And also develop “relevance-driven” approach to index and materialized view selection.

References

[1] CHUAN ZHANG, JIAN YANG, Materialized View Evolution Support in Data Warehouse

Environment. In *DASFAA, Taiwan, ROC, 1999*, pg. 247-254.

[2] GUPTA, H. Selection of views to materialize in a data warehouse. In *Intl. Conf. On Database Theory, Delphi, Greece, 1997*, pg. 98-112.

[3] GUPTA, A., AND MUMICK, I. S. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin (Special issue on Materialized Views and Data Warehousing)*, June 1995, pg. 3-18.

[4] GUPTA, H., HARINARAYAN, V., RAJARAMAN, A., AND ULLMAN, J. Index selection for olap. In *Intl. Conf. on Data Engineering, Birmingham, UK, April 1997*, pg. 208-219.

[5] GUPTA, H., AND MUMICK, I. S. Selection of views to materialize under maintenance cost constraint. In *Intl. Conf. on Database Theory, Jerusalem, Israel, 1999*, pg. 453-470.

[6] HARINARAYAN, V, RAJARAMAN, A AND ULLMAN, J. Implementing data cubes efficiently. In *ACM SIGMOD Intl. Conf on Management of Data, Montreal, Canada, June 1996*, pg. 205-216.

[7] JEFFREY D. ULLMAN, Efficient Implementation of Data Cubes via Materialized Views Technology. In *ACM SIGMOD record, volume 2, 1997*, pg. 65-74.

[8] JENNIFER WIDOM, Research Problems in Data Warehousing. In *Proc. Of 4th Intl. Conf on Info and Knowledge Mngt. of Data, 1995*, pg. 25-30.

[9] YANG, J., KARLAPALEM, K., AND LI, Q. A framework for designing materialized views in data warehousing environment. *Technical Report HKUST-CS96-35, October 1996*.

[10] PRASAN ROY, Multi-Query Optimization and Applications. *PhD Thesis, IIT Bombay, 2000*.

[11] SHAMKANT B.NAVATHE, MINYOUNG RA, Vertical Partitioning for Database Design: A graphical Algorithm, In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, 1989*, pg. 440-450.

[12] SURAJIT CHAUDHURI, UMESHWAR DAYAL, An Overview of Data Warehousing and OLAP Technology. In *ACM SIGMOD Record, volume 26, 1997*, pg. 65-74.

[13] YUBO FAN, Materialized View Algorithms. *M.S Thesis, Portland State University, 1997*.

[14] YANG, J., KARLAPALEM, K., AND LI, Q. Algorithms for materialized view design in data warehousing environment. In *Proceedings of 23rd International Conference on Very Large Data Bases, Athens, Greece, 1997*, pg. 136-145.