Selection of Materialized Views

Dr. Madhu Bhan  
Department of Computer  
Applications  
Ramaiah Institute of Technology  
Bangalore, India  
madhoobhan@yahoo.co.in

Dr. K Rajanikanth  
Department of Information  
Science and Engineering  
Ramaiah Institute of Technology  
Bangalore, India

Dr. T.V. Suresh  
Department of Computer  
Applications  
Ramaiah Institute of Technology  
Bangalore, India

Abstract—An important technique to improve the performance of a Data Warehouse is to evaluate a-priori the frequent queries and store the results as materialized views for subsequent use. The best response times can be achieved when all the views corresponding to all possible queries are materialized. However, this is not a feasible approach for two main reasons. Materialized views are just like tables and thus occupy storage. Further, the views need to be updated whenever the base tables change. Considering the exponential growth in the number of views with increasing dimensions, and the consequent increase in the storage costs and update costs, it would be prohibitively expensive to materialize all the views. Thus one needs to make a selection of a sub-set of views to be materialized. The problem of the selection of views to be materialized, generally called “Materialized View Selection (MVS)” problem is an NP-hard problem. Thus, often the only practical approach to solve this problem is to use approximation techniques. The challenge is to materialize a subset of views which results in best benefit in terms of query response times while taking into consideration the storage cost and maintenance cost. Approximation algorithms for view selection are proposed in [8,9]. The classic greedy algorithm for view selection was proposed by Harinarayana et al.[6]. The algorithm selects the “most valuable” view from a set of as yet not materialized views, based on the benefit of a view which is calculated as the reduction in the cost for processing that view and its descendant views if that view were to be materialized. The authors propose a lattice structure for the set of views. The lattice is induced by a partial ordering relation that denotes the concept that it may be possible to derive a view V2 from another view V1. They use this lattice structure to pose the problem of selecting a pre-specified number of views and also the problem of selecting the best sub-set of views subject to a constraint on the maximum storage space allowed. Much of the formalism introduced in this seminal problem continues to be used by researchers in this area [10,11,12,13,14,15]. The authors in these works propose two criteria for
stopping the selection process. In the first version, the maximum number of views to be materialized is specified as the constraint and in the second version the maximum storage available for the materialized views is specified as the constraint. Although the results obtained by greedy algorithm are near optimal, it is generally difficult to specify the required constraints. While the algorithm works well once the constraints are specified, the specification of constraints in itself is difficult. How does one specify the maximum number of views to be materialized given a view lattice with certain number of views? It may be easier to specify the maximum storage constraint. However, even this may be rather difficult because users are now willing to pay more for the storage costs if they can get better performance. We propose two simple stopping criteria for the classic greedy algorithm. We also propose new approximation algorithms and evaluate them through simulation studies.

B. Notation

Let V be the set of all views that may be materialized. We assume that the views in the set V form a lattice induced by a partial order \( \leq \) as defined below:

\[ v_1 \leq v_2 \] if and only if the view \( v_1 \) can be derived from the view \( v_2 \). Associated with each view \( v_j \in V \), is the number of rows in that view, denoted as size \( (v_j) \) that is used as surrogate for the number of disk pages required to store that view and thus as the cost for answering the corresponding query [6]. As each view corresponds uniquely to a query, we can use “query” in the place of “view” without any loss of meaning. Further, the queries in the context of present discussion are all aggregations from a “data cube” grouped by some sub-set of the set of attributes of the underlying fact table and thus are called “cuboids” also. If the set of views form a hypercube lattice, and if there are d dimensions, then the number of views in the view set \( V \), i.e., the cardinality of \( V \), \( |V| = 2^d \). If the attributes have hierarchies, the corresponding lattice can be formed as a “direct product” of the dimensional lattices [6]. Further, for the lattice structure, we only require that the relation \( \leq \) is a partial order and that there is a “top view” from which all other views in the set can be derived [6]. The Materialized View Selection (MVS) problem can now be stated as follows:

C. Problem MVS:

Given \( V \), a set of views with associated sizes (costs) and a partial order \( \leq \), determine \( V_s \), \( V_s \subseteq V \), the sub-set of views to be materialized based on specified criteria

II. PROPOSED ALGORITHMS

In this section we propose some simple stopping criteria as well as new greedy algorithms and evaluate them through simulation studies.

A. Algorithm GSC1 (Greedy with Stopping Criterion 1)

The stopping criteria that we use here is the cost/space ratio. The cost/space ration represents the trade-off between decrease in total processing cost of all the queries and increase in total storage space. The rationale for stopping criterion is discussed here. The root view \( v_0 \) is always materialized because it can’t be derived from any other view. If this is the only view selected, the total cost for processing all the queries, assuming that they are all equally frequent, is evidently \( |V| * \text{size}(v_0) \). The total space required is just \( \text{size}(v_0) \). As more and more views are selected for materialization, the total cost will keep decreasing and the total space required will keep increasing. Finally, if all the views are materialized, the total cost will become the sum of the sizes of all the views and the total size also will be same. Thus, the ratio of total cost to total space starts with a value equal to \( |V| \) and keeps decreasing as more and more views are selected for materialization and finally becomes 1 when all the views are materialized. This ratio is thus easy to specify based on business considerations. So, we propose a bound on this value as the stopping criterion. The resulting algorithm called GSC1 (Greedy algorithm with Stopping Criterion 1) is outlined below:

\[ V_0 = \{v_0, \text{the root view}\} \]

While there are views available for selection do {

Select the view \( v_k \) such that \( v_k \notin V_s \) and \( v_k \) has the maximum benefit among all the views \( \notin V_s \):

\[ V_s = V_s \cup \{v_k\} \]

if \( \left( \frac{\text{total cost}}{\text{total space}} \right) < \text{cost-space-ratio-limt} \) break;

}

The procedure for computing the benefit of a view is as described in [6]. It is clear that as the specified cost-space-ratio-limit gets smaller and smaller, more and more views get selected for materialization leading to increased space cost but providing faster average response to the queries. Thus this cost-space-ratio-limit must be set based on the cost user is willing to pay for a given improvement in query processing performance.

B. Algorithm GSC2 (Greedy with Stopping Criterion 2)

Reasoning along the same lines as in Algorithm GSC1, the stopping criteria that we use here is the percentage cost/space ratio. The percentage cost/space ratio represents the percentage reduction in the total cost for processing all the queries to the percentage increase in the total storage requirement as the next most valuable view is added to the set of views selected for materialization. We stop the selection process when this ratio goes below a specified limit. Again, the limit is set based on business considerations of what the users are willing to pay for additional storage to get further improvement in the total processing time. This rationale leads to the algorithm called GSC2 (Greedy algorithm with Stopping Criterion 2) which is outlined below:

\[ V_0 = \{v_0, \text{the root view}\} \]

While there are views available for selection do {

Select the view \( v_k \) such that \( v_k \notin V_s \) and \( v_k \) has the maximum benefit among all the views
Another approach to selecting views for materialization with easy stopping criterion is based on eliminating candidate views based on their relative sizes. In [7], the authors propose eliminating some views a-priori based on comparing their sizes with the sizes of their parents. However, an alternative approach proposed here considers such elimination of candidate views after selecting a view! The rationale is discussed now. Let us assume that a view is selected for materialization based on the greedy approach. Now a descendant of this view, is not worth materializing if the size of descendant view is “nearly same” as the selected view(parent). As the size of descendant view is nearly same as size of parent view, queries which can be answered using descendant view can as well be answered using the already materialized parent view at only a marginal increase in the processing cost while saving on storage and update costs. Further, eliminating such descendents from the computation of benefit of any other remaining views may help in making better choices in subsequent view selections. As “unworthy” views are being eliminated with each selection of a view there is no need for an explicit stopping criterion; we continue selecting views as long as there are views available for selection! The Algorithm GVE (Greedy algorithm with View Eliminations) is thus defined as follows:

\[
\forall v_j, \quad V_r = V_r \cup \{ v_j \};
\]

for each descendant \( v_j \) of \( v_k \) do \

\[
\text{if } ((\text{percentage decrease in total cost} / \text{percentage increase in the total space}) < \text{percentage-cost-space-ratio-limit}) \text{ break;}
\]

However, this ratio decreases only up to a point and then increases to dramatically high values. This high increase occurs as we move towards selection of lower-sized views present at lower levels in the lattice. As these views have very low sizes, the percentage increase in the total space when these views are added is near zero. Such a near-zero value appearing in the denominator makes the ratio reach very high values. This behavior must be taken into account when setting a limit on the ratio of percentage decrease in total cost to percentage increase in the total space!

C. Algorithm GVE (Greedy with View Elimination)

Another approach to selecting views for materialization with easy stopping criterion is based on eliminating candidate views based on their relative sizes. In [7], the authors propose eliminating some views a-priori based on comparing their sizes with the sizes of their parents. However, an alternative approach proposed here considers such elimination of candidate views after selecting a view! The rationale is discussed now. Let us assume that a view is selected for materialization based on the greedy approach. Now a descendant of this view, is not worth materializing if the size of descendant view is “nearly same” as the selected view(parent). As the size of descendant view is nearly same as size of parent view, queries which can be answered using descendant view can as well be answered using the already materialized parent view at only a marginal increase in the processing cost while saving on storage and update costs. Further, eliminating such descendents from the computation of benefit of any other remaining views may help in making better choices in subsequent view selections. As “unworthy” views are being eliminated with each selection of a view there is no need for an explicit stopping criterion; we continue selecting views as long as there are views available for selection! The Algorithm GVE (Greedy algorithm with View Eliminations) is thus defined as follows:

\[
V_r = V_r \cup \{ v_j \};
\]

for each descendant \( v_j \) of \( v_k \) do \

\[
\text{if } (v_j \in V_r \quad \text{still under consideration, not an already eliminated view}
\]

\[
\text{and } (\text{size}(v_j) > \text{size-ratio-limit} \ast \text{size}(v_k))
\]

\[
V_r = V_r \setminus \{ v_j \} \quad // \text{eliminate } v_j \text{ from further consideration}
\]

while \( (V_r \neq \emptyset) \) do \\

\[
\text{Select the view } v_k \text{ such that } v_k \in V_r,
\]

\[
\text{and } v_k \text{ has the maximum benefit among all the views}
\]

\[
V_r = V_r \setminus \{ v_k \}
\]

D. Algorithm GVES1 (Greedy with View Elimination and Stopping Criterion 1)

Algorithm GVE does not have any explicit stopping criterion. It keeps eliminating “undesirable” views and keeps selecting the most valuable view among the remaining ones as long as selections can be made. However, it is possible to apply an explicit stopping criterion after selecting a view. Algorithm GVES1 proposed here is based on this observation. It works exactly like Algorithm GVE in eliminating not-so-useful views but then applies the stopping criterion of Algorithm GSC1 to check if the selection process is to be terminated or not. As the probability of terminating the selection immediately after selecting only the root view is nearly 0, it applies the stopping criterion only inside the “while” loop. Thus, we add only one last step in the “while” loop of Algorithm GVE as shown below to get Algorithm GVES1.

\[
// \text{add one statement as the last statement in the while loop of Algorithm GVE …}
\]

\[
\text{if } ((\text{total cost} / \text{total space}) < \text{cost-space-ratio-limit}) \text{ break;}
\]

E. Algorithm GVES2 (Greedy with View Elimination and Stopping Criterion 2)

Algorithm GVES2 is quite similar to Algorithm GVES1 and follows the same logic except that the stopping criterion is the one used in Algorithm GSC2 rather than the one used in Algorithm GSC1. Thus, only the last step in the “while loop” of Algorithm GVES1 is changed as shown below to get Algorithm GVES2.

\[
// \text{Change only the last statement in the while loop of Algorithm GVES1…}
\]

\[
\text{if } ((\text{percentage decrease in total cost} / \text{percentage increase in the total space}) < \text{percentage-cost-space-ratio-limit}) \text{ break;}
\]

The comment made earlier with respect to Algorithm GSC2 that the ratio of percentage decrease in total cost to percentage increase in the total space decreases only up to a point and then increases to dramatically high values as views of smaller sizes get selected is true for this algorithm also. Thus, one must set the limit after due consideration of the nature of the sizes of views in the data cube.
III. ILLUSTRATION AND EVALUATION OF PROPOSED ALGORITHMS

The five algorithms proposed above are all illustrated with the same example in this section. The example considered is a lattice based on TPC-D database[6]. This lattice is shown in the following Fig 1. The lattice has two dimensions, the customer dimension and the part dimension. The customer dimension has two hierarchical levels: The individual customer denoted by attribute c and grouped by country denoted by attribute n. For the part dimension, individual parts are denoted by attribute p. These individual parts are grouped by their size denoted by attribute s. They are also grouped by their types denoted by attribute t. Selection of all the views in the greedy order is shown in the following Table 1 [6].

A. Algorithm GSC1

For the example given, the ratio of total cost to total space starts with a value of 12 when only the root view is materialized and ends with the value of 1 when all the views are materialized. If we set the cost-space-ratio-limit as 2.5, then GSC1 will select 6 views.

The total cost would be 23312500 and the total storage would be 11305000. Thus the total cost is only about 4.6% more than the minimum possible value of 22295226 with the total storage being just about 50% of the maximum storage value of 22295226, a reasonably good choice of the views to be materialized without explicitly specifying the maximum number of views to be materialized. From, the examination of the entire Table 1, we can of course conclude that selecting only 5 views gives a slightly better solution. But, as noted earlier, determining this number beforehand is not generally feasible!

B. Algorithm GSC2

For the same example given above, the ratio of the percentage reduction in the total cost for processing all the queries to the percentage increase in the total space starts with a value of 12 when only the root view is materialized, decreases to a value of 0.001 and then starts increasing again to end with the value of 24.0 when all the views are materialized. The increase in this ratio occurs because the views at lower level nodes in the above lattice diagram have negligible storage sizes. This need not be true in all cases. If we set the limit for this ratio as 0.5, then GSC2 will also select 6 views as GSC1. Again, as we see later in the simulation results, this need not be the case always. As the number of views selected in this case is same as the number of views selected by GSC1, the total cost and the total storage have the same values and similar remarks regarding the best choice apply.

C. Algorithm GVE

The approach adopted by Algorithm GVE is quite different from the approach adopted in the earlier two algorithms, GSC1 and GSC2. There is no explicit stopping condition specified in this algorithm. We set the size-ratio-limit to 0.8 and show the working of this algorithm with the same example described above.

Step 1: Root view, cp is selected. Its size is 6000000. All the other views are descendents of this view. Among them, ct with a view size of 5990000, cs with a view size of 5000000, and np with a view size of 5000000 have sizes which are greater than 0.8 * 6000000 and thus get eliminated from further consideration.

Step 2: ns is now the most valuable view and is selected. Its size is 1250. None among its descendents has a size that is greater than 0.8 * 1250 and so none get eliminated.

Step 3: nt is the next view selected. Again, no view gets eliminated by size comparison.

Step 4: c is the next view selected. Again, no view is eliminated.

Step 5: p is the next view selected. No view is eliminated.

Step 6: t is selected. No view is eliminated.

Step 7: n is selected. No view is eliminated.

Step 8: s is selected. No view is eliminated.

Step 9: The bottom view "NONE" is selected. No view is eliminated.

No more views exist for selection and thus the algorithm terminates. Not many views get eliminated in this example because the views at lower level have values which are very

<table>
<thead>
<tr>
<th>Sl. Number</th>
<th>View Selected</th>
<th>Total cost</th>
<th>Total Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cp</td>
<td>72000000</td>
<td>6000000</td>
</tr>
<tr>
<td>2</td>
<td>ns</td>
<td>48005000</td>
<td>6001250</td>
</tr>
<tr>
<td>3</td>
<td>nt</td>
<td>36012500</td>
<td>6005000</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>30112500</td>
<td>6105000</td>
</tr>
<tr>
<td>5</td>
<td>p</td>
<td>24312500</td>
<td>6305000</td>
</tr>
<tr>
<td>6</td>
<td>cs</td>
<td>23312500</td>
<td>11305000</td>
</tr>
<tr>
<td>7</td>
<td>np</td>
<td>22312500</td>
<td>16305000</td>
</tr>
<tr>
<td>8</td>
<td>ct</td>
<td>22302500</td>
<td>22295226</td>
</tr>
<tr>
<td>9</td>
<td>t</td>
<td>22298000</td>
<td>22295150</td>
</tr>
<tr>
<td>10</td>
<td>n</td>
<td>22296450</td>
<td>22295175</td>
</tr>
<tr>
<td>11</td>
<td>s</td>
<td>22295250</td>
<td>22295225</td>
</tr>
<tr>
<td>12</td>
<td>none</td>
<td>22295226</td>
<td>22295226</td>
</tr>
</tbody>
</table>

Table 1: Greedy Order of Views Selected for the Example Lattice
small compared to the sizes of their ancestors! Later simulation studies show that this need not be true always. Anyway, this algorithm selects 9 views in all for a total cost of 24305226 and a total size of 6305226. Comparing these results with the results obtained from Algorithm GSC1 and GSC2, we observe that the present algorithm prefers views of smaller size to views of larger size! Thus the view cs selected by those algorithms is ignored by the present algorithm. Instead, the smaller views ct, t, n, s, and the bottom view are selected!! The total cost is marginally higher but the total size is significantly lower. The total cost of this algorithm is actually just about 4.2% higher than the total cost resulting from Algorithm GSC1 or Algorithm GSC2 but the total space is only about 55.7% of the total size resulting from Algorithm GSC1 or Algorithm GSC2!

**D. Algorithm GVES1**

Algorithm GVES1 is quite similar to Algorithm GVE but for the stopping condition. With the same size-ratio-limit of 0.8 and the additional stopping criterion of cost-space-ratio-limit set as 4.0, we show the working of this algorithm with the same example described above.

**Step 1:** Root view, cp is selected. Its size is 6000000. All the other views are descendents of this view. Among them, ct with a view size of 5990000, cs with a view size of 5000000, and np with a view size of 5000000 have sizes which are greater than 0.8 * 6000000 and thus get eliminated from further consideration.

**Step 2:** ns is now the most valuable view and is selected. Its size is 1250. None among its descendents has a size that is greater than 0.8 * 1250 and so none get eliminated.

- cost-space-ratio = 7.999; so continue

**Step 3:** nt is the next view selected. Again, no view gets eliminated by size comparison.

- cost-space-ratio = 5.998; so continue

**Step 4:** c is the next view selected. Again, no view is eliminated.

- cost-space-ratio = 4.93; so continue

**Step 5:** p is the next view selected. No view is eliminated.

- cost-space-ratio = 3.856 < 4.0, the limit specified; so break

The algorithm terminates.

Thus this algorithm selects a set of a total of 5 views, {cp, ns, nt, c, p} for a total cost of 24312500 and total size of 6305226. Thus this Algorithm also, like Algorithm GVE prefers small-sized views to large-sized views but does not pick all of them. This leads to smaller number of views selected which may be advantageous from the maintenance point of view. The total cost and the total size are nearly same as those resulting from the Algorithm GVE but the number of selected views is much less; essentially the small-sized views at lower levels are the ones which are ignored by the present algorithm.

**E. Algorithm GVES2**

Algorithm GVES2 is quite similar to Algorithm GVES1 except that the stopping criterion is different. With the same size-ratio-limit of 0.8 and the additional stopping criterion of the ratio limit set to 6.0, we show the working of this algorithm with the same example described above.

**Step 1:** Root view, cp is selected. Its size is 6000000. All the other views are descendents of this view. Among them, ct with a view size of 5990000, cs with a view size of 5000000, and np with a view size of 5000000 have sizes which are greater than 0.8 * 6000000 and thus get eliminated from further consideration.

**Step 2:** ns is now the most valuable view and is selected. Its size is 1250. None among its descendents has a size that is greater than 0.8 * 1250 and so none get eliminated. ratio = 160; so continue

**Step 3:** nt is the next view selected. Again, no view gets eliminated by size comparison. ratio =40; so continue

**Step 4:** c is the next view selected. Again, no view is eliminated. ratio = 9.85; so continue

**Step 5:** p is the next view selected. No view is eliminated. ratio =5.88 < 6.0, the limit specified; so break

The algorithm terminates.

Thus, in this particular case and for the chosen limit, this algorithm’s behavior is completely identical to that of Algorithm GVES1. This need not be the case always as shown later in the simulation studies. As remarked earlier, the ratio decreases and then increases and so must be set carefully.

**IV. EVALUATION OF THE PROPOSED ALGORITHMS**

The proposed algorithms are evaluated using simulation techniques as described in this section. We consider a hypercube lattice of 32 views for this purpose. The lattice is shown in the following Fig 2. (Not all dependencies are shown in the interest of clarity.) The sizes of the 32 views are changed randomly for each problem instance. For a given problem instance, the sizes are set up as follows:

![Fig 2: The Lattice used in Simulation Studies](image-url)
The size of the root view is fixed at 5000000. The size of any view is fixed as a random value between 54% and 99.9% of the size of the minimum-sized parent of that view. For example, to set the size of the view \{AB\}, we examine its parent views \{ABC\}, \{ABD\}, and \{ABE\}. Let the smallest among these 3 views be \{ABD\}. Then the size of the view \{AB\} is set as a random value between 0.54 * size (\{ABD\}) and 0.999 * size (\{ABD\}). Thus there may not be any sharp rise in the Total Size as more and more views are selected; rather the Total Size is likely to grow steadily with increasing number of selected views. Similarly, the Total Cost is also likely to decrease with less sharp changes. The problem instance generated as described above is solved by each of the algorithm to be evaluated. The problem is also solved, for comparison purpose, by the Greedy Algorithm of [6] with no constraint on the number of views or storage space. In each case 10 problem instances are solved for detailed analysis. Let the number of views selected by this algorithm be \(ns\). For comparison purposes, the Total cost/Min cost and Total space/Max space are computed using the greedy approach with view constraint varying from \(ns-3\) to \(ns+3\). In each case, the Total Cost/Min Cost (Total cost ratio) expresses the factor by which the cost of processing all the queries is more than the minimum total cost which corresponds to selecting all the views. Similarly, Total Size/Max Size (Total size ratio) is expressed as the factor by which the total size is less than the maximum total size which corresponds to selecting all the views.

A. Performance of Algorithm GSC1

The 10 problem instances are solved using Algorithm GSC1 with cost-space-ratio-limit set to 0.5. Let the number of views selected by this algorithm be \(ns\). The Table indicates the number of views selected \(ns\) using GSC1 algorithm for all the ten problem instances (numbered from 1 to 10). Using Greedy approach the Total Cost Ratio and Total Space Ratio for the number of views ranging from \(ns-3\) to \(ns+3\) is also presented. The columns in Table 2 corresponding to \(ns\) are shown in bold. For example, the GSC1 algorithm selects 13 views for problem instance 1, and the cost for these 13 views is 89138216 whereas the minimum cost is 81159248. Thus the cost is only 1.098 times the minimum cost. Similarly the space required by these 13 views is 33956419 and the maximum space required is 81159248. Thus the space would just be 0.418 times the maximum space. This again is indicated in bold in the table. The Total cost Ratio and Total space ratio for number of views selected in the range of 10 to 16 is computed as additional information for the designer to choose number of views for selection around the value of 13, should it provide a better trade-off between cost and space. It can be seen that the number of views selected by Algorithm GSC1 is a fairly good choice in all the cases and this was possible without explicit specification of view constraint or size constraint.

B. Performance of Algorithm GSC2

The 10 problem instances are solved using Algorithm GSC2 with the ratio of percentage reduction in the total cost to the percentage increase in the total space set to a limit of 0.1. Table 3 indicates the number of views selected (shown as \(ns\) in the table) using GSC2 algorithm for all the ten problem instances. Let the number of views selected by this algorithm be \(ns\). In each case, the Total Cost is expressed as the factor by which it is more than the minimum total cost which corresponds to selecting all the views. Similarly, Total Size is expressed as the factor by which it is less than the maximum total size which corresponds to selecting all the views. From the Table 3 we can see that for problem instances 1, 2 and 5 the number of views selected using GSC2 is same as the number of views selected by GSC1. As the number of views selected in these cases is same the Total cost ratio and the Total storage ratio have the same values in Table 2 and Table 3. However, the number of views selected in other instances is different. In general, it appears to be more difficult to select the limit for use in this algorithm because the ratio decreases and then increases again! This is due very small sized views at lower levels which need to be selected for materialization.
C. Performance of Algorithm GVE

The GVE algorithm uses no stopping criteria but based on the parent-child size ratio, it eliminates some of the children from further consideration for selection. Setting the size-ratio-limit to 0.8, the 10 problem instances are solved using GVE and the results obtained are presented in Table 4. Unlike the earlier algorithms, Algorithm GVE selects views after elimination of some views. Hence the Total Cost and Total size of the selected views do not necessarily correspond to the total cost and total size of the same number of views selected by the Greedy Algorithm! This is the reason why Table 4 shows not only the number of views selected (ns) by Algorithm GVE but also the corresponding Total Cost and Total Space Ratio. For comparison purposes, the Total Cost Ratio and Total Space Ratio are computed using greedy approach with view constraint varying from ns-3 to ns+3. From the table we see that, the number of views selected by GVE algorithm have almost the same Total Cost Ratio as obtained by greedy approach for similar number of views but the Total Space ratio is generally less than the one resulting from the Greedy...
The reason for this, as explained earlier, is that this algorithm prefers smaller views to larger views.

GVES1 algorithm is the GVE algorithm with the additional stopping criterion based on cost-space-ratio-limit. With the same size-ratio-limit of 0.8 and the additional stopping criterion of cost-space-ratio-limit set as 4.0, the 10 problem instances are solved and the results are presented in Table 5.

We observe that the number of views selected is generally less than the number of views selected by the Algorithm GVE as can be expected intuitively.

We observe that many of the problem instances solved with GVES1 have better Total Cost Ratios as compared to Total Cost Ratios resulting from the greedy approach. But the Total Space Ratio tends to be larger because we terminate the algorithm as soon as the set limit is crossed. Thus some of the large sized views do not get eliminated. Evidently, changing the value of this limit would eliminate some of the large sized views do not get eliminated. Evidently, changing the value of this limit would
alter the performance of the algorithm as in the case of Algorithm GSC1.

E. Performance of Algorithm GVES2

Algorithm GVES2 is similar to Algorithm GVES1 except that the stopping criterion is now based on the ratio of percentage reduction in the total cost to the percentage increase in the total space. Setting the size-ratio-limit of 0.8 and the additional stopping criterion of ratio of percentage reduction in the total cost to the percentage increase in the total space set to a limit of 0.1, the 10 instance problems are solved using GVES2, and the results are presented in Table 6. We observe that Total cost ratios with GVES2 are better than Total cost ratios returned by the greedy approach, but the Total space Ratios tend to be larger. If we select larger number of views with greedy approach, we can get the same performance. However, the choice of the cut-off limit, as with GSC2, appears to be difficult as the ratio decreases and then increases.

V. SUMMARY OF THE RESULTS

All the five algorithms presented above have the advantage that no explicit view constraint or space constraint need be specified however the limitation of these algorithms is that they require the sizes of the views as input. Algorithms GSC1 and GSC2 work fairly well except for the fact that the stopping criterion used in GSC2 is difficult to specify. The ratio of percentage reduction in the total cost to the percentage increase in the total space does not vary monotonically and thus setting a cut-off limit a-priori appears to be difficult. The Cost-Space-Ratio on the other hand reduces consistently and thus a cut-off limit can be set more easily. GVE does not require any stopping criterion. Number of views selected by algorithm GVE has almost the same Total cost ratio as obtained by Greedy approach. However the Total Space ratio is less than the one resulting from Greedy approach due to the fact that this algorithm selects views of smaller size as compared to views of large size. Algorithms GVES1 and GVES2 work exactly like GEV. Algorithm GEVS1 uses an additional stopping criteria of GSC1 while algorithm GVES2 uses an additional stopping criteria of GSC2. These algorithms provide better Total cost ratios as compared to Total cost ratios provided by Greedy approach. However their Total space ratios are inferior to the ones obtained from Greedy approach. This is because both these algorithms use stopping criterion. From our simulation study, algorithm GSC1 and GVES1 appear to be better choices for actual use. The values for various ratios have been just taken to illustrate the behavior of Total cost and Total size ratios. The values can be set based on empirical observations when used with specific real systems.

The problem of selecting a subset of views for materialization is a NP-hard problem. Greedy algorithms as well as the five algorithms proposed are only approximate solutions.

REFERENCES

[13]. Sajid Ullah Khan, “Data warehouse enhancement manipulating materialized view hierarchy” Proc. of eight International Conference on Digital Information Management (ICDIM), 2014 DOI: 10.1109/ICDIM.2013.6604037