Compressing Multidimensional Structures – A Case Study

JORGE RIBEIRO¹, HELDER RIBEIRO²
School of Technology and Management
Polytechnic Institute of Viana do Castelo
Avenida do Atlântico, Viana do Castelo
PORTUGAL

¹jribeiro@estg.ipvc.pt ²helderrib@gmail.com http://www.ipvc.pt

Abstract: - The OnLine Analytical Processing (OLAP) operate on the information from the Data Warehouses, pre-calculating and processing all combinations of the group-by operator and materializing them in Multidimensional Structures or Data Cubes. It is a computational task to realize that needs time, space to store the data. Many studies presented various techniques oriented to the Data Cube compression like the Dwarf, Condensed Cube, BU-Condensed Cube, Min Cube, Prefix Cube and Quotient Cube. This paper presents a study of the difficulty to process the Data Cube and we implemented and applied the Prefix Cube and the Bottom-Up Condensed Cube comparing the performance with the traditional Data Cube processing. Good results were achieved reducing in 70% the traditional Data Cube in different types of data sets.

Key-Words: - Multidimensional Structures, Data Reduction, Data Compression, Prefix Cube, Bottom-Up Suppress Cube.

1 Multidimensional Processing

The Multidimensional processing is typically a difficult computational task once that, from the information stored in the Data Warehouses (DW) are processed all possible combinations of the aggregations between the different dimensions of the Data Marts (DM). To analyze this problem, for a general relation \( R \) with \( n \) dimension attributes \( (D_1, D_2,..., D_n) \) and a measure attribute \( M \). The processed Multidimensional structure from this relation \( R \) in all dimensions and applying the group-by operator it will generate \( 2^n \) distinct groups constituting the Data Cube [1]. To exemplify this computational task consider the relation \( R \{A, B, C, M\} \) in the table 1 as an example of a simple relation in a typical DW. This relation has three dimensions \( \{A\}, \{B\}, \{C\} \) and a measure \( M \), and adopted the SUM function as an aggregation function. This relation \( R \) contains three cells or tuples \((t1, t2, t3)\) and will process \( 2^3 \) groups (figure 1) corresponding to eight sub-groups (or cuboids). The interconnection of the cells for each group is presented in the figure 2. The symbol “*” as a dimension value denotes the generalization of the dimension value, or in other words a special value to denote the "ALL" value in which any value can be assumed.

2 The Cube Lattice

Analyzing the cells of the cube lattice presented in the figure 2 and given the increased number of cells interconnected with each other, some peculiar characteristics deserve to be analyzed. One is the existence of various cells with the same value and the other refers to the semantic redundancy that occurs between the various cells. This last feature is one in which many algorithms [2, 3, 4] for Data Cube processing and compression give special attention, trying to cluster cells with sharing dimension attributes, in order to store only a few tuples and thus reduce the storage space of the Data Cube. This characteristic of redundancy between the cells can be divided into two types: prefix redundancy and suffix redundancy. Prefix redundancy can be detected and understood considering the example of the table 1 and can be easily visualized in the representation of the cube lattice in figure.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>t3</td>
<td>2</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1 – Base tables

The figure 1 and 2 shows a relation with three dimensions \( \{A\}, \{B\} \) and \( \{C\} \), where for a given cell the value of the dimension \( \{A\} \) appears in four sub-groups (\( \{A\}, \{AB\}, \{AC\}, \{ABC\} \)) and possibly several times in each sub-group. For example, for the cell t1 of the table 1, the value “1” of the dimension \( \{A\} \) will appear six times in the Data Cube, particularly in the groups \((1,8,7), (1,3,7), (1,8), (1,3), (1,7) \) and \((1)\) corresponding respectively to the groups...
Consider these facts, instead of storing all the tuples of all sub-groups some techniques [2, 3, 4] in order to compress the Data Cube only stores once the corresponding values of the cells that belong to a different sub-group. Indeed, besides it can eliminate the prefix redundancy, can also reduce the storage space of the Data Cube, since that instead of storing all the tuples store just a few. Suffix Redundancy is the most dominant factor in the cells of the cube lattice occurring in most cases when a set of cells shares the same dimension prefix and all cells of the sub-groups have the same value. For example, consider the example of the table 1 and the processing of the several sub-groups presented in the table 2. The cells t1: (1,8,7:6) and t2: (1,3,7:12), could be stored once and the cells in the column (a) of the table 2 are referred to them. The reason for all sub-groups can be coalesced, focuses on the involvement between the dimensions values and the operation that eliminates the suffix redundancy commonly called as “coalescison”.

Table 2 – Examples of the suffix redundancy.

### 3 Reduction techniques

Traditionally, researchers in the data reduction field are interested to obtain accurate answers to the queries, and minimize the response time of these queries. Several studies have been conducted to reduce the data with the initial objective to estimate and select queries, so as to get quick answers to queries without significant concerns on the quality of information obtained. This was initially achieved using the approaches of responses that, in samples of data it was possible to obtain the information as soon as possible. With the evolution of the studies the concern also focused on obtaining quick answers, but became an important the accuracy of the responses to queries with the high issue levels of quality. With the emergence of the Data Cube several studies have focused initially on improving the performance in relation to the Data Cube processing while others focuses on ways to get small structures (summary data [2, 3, 4]) to reduce the storage space. Some
studies have emerged which began to focus in particular on the processing and compression of the Data Cube. In the first case, several of them have been proposed to process efficiently the Data Cube. Zhao et al. [6] presented the "Array Cube" algorithm, which is based on creating multidimensional arrays like structures following the guidance Top-down to select the dimensions to process.

Beyer & Ramakrishan [7] presented the Bottom-Up Computation (BUC) algorithm outlining a strategy to select the partitions of the Data Cube to be processed following the Bottom-up orientation. This last algorithm was very useful and used in several studies in the processing and compression of the Data Cube and has its origins in the processing of queries such as the "Iceberg" [10]. These types of queries had the particularity to return only partitions that meet the aggregation conditions specified by the user. Thus, it is possible with this algorithm to "prune" partitions that do not meet a particular condition (e.g. when there is a minimum number of cells in a partition), allowing only the processing of a part of the Data Cube. Another study oriented to the Data Cube processing was the "Star – Cube" [8] where there is a star structure to store the processed values extending the aggregation method.

To study the problem of selecting views to materialize the data of the Data Cube based on space restrictions some studies are summarized in [9]. Fang et al. [10] presented the algorithm "Iceberg-Cube", which allows to select only a subset of the Data Cube containing only cells in which the measure satisfies certain restrictions. By this way it will be processed only an excerpt of the information from the total Data Cube. Another study oriented to the Data Cube processing was the "Star – Cube" [8] where there is a star structure to store the processed values extending the aggregation method.

Several studies examine the problems and limitations of these algorithms and try to improve them. Feng et al. [11] improved the Suppress Cube detecting that it still does not completely eliminate the redundancy prefix. It proposes a new structure for storing and sharing the prefixes between tuples, naming this new algorithm by "Prefix Cube" (figure 5).

Longgang et al. [12] note that although the Dwarf structure eliminates the redundancy as much as the Prefix Suffix in fact in some cases it can’t completely eliminate the suffix redundancy creating additional redundant nodes. They proposed the replacement of the suffix redundancy elimination by a new algorithm called Dwarf schemes (figure 4) to execute the partitioning and the insertion of tuples. The same authors that proposed the QC algorithm in [4] present a structure called QC-Trees [5] in order to implement the algorithm QC (figure 6).
The incremental maintenance of the Data Cube is one of the problems that influences the efficiency of the processing and reduction of the Data Cube. Several studies [13, 14, 15] are oriented for this problem and for the treatment of groups, views or for the Data Cube.

Also the problem of the incremental maintenance is discussed with particular attention in the majority of the last algorithms and they have structures for indexing the data in order to facilitate the invocation of queries and deal with the incremental update of these structures. Is the case of the Cube Forest [16], the Statistics Trees [17], the Cubetrees [18], the QCTrees [5] and the DC-Trees [19]. Feng et al. [11] in order to incrementally update the Supress Cube presented the "CuboidTree" [20]. The Dwarf structure uses its own structure and mechanism for indexing and did not require extra mechanisms to address the problem of maintaining the information. Li et al. [21] presented a new solution for the incremental maintenance of the QC algorithm.

3. Experimental Results

All experiments were tested on machines with Pentium IV 3Ghz with 1024 MB DDR RAM and 320 GB disk with 72 RPM. These algorithms focus on the generation of structures in memory and then the storage in disk. The algorithms were implemented in Visual C++ 6.0 [22]. In this study we implemented the Data Cube processing, the BU-Cube and the Prefix Cube. For the evaluation of the processing time it was included the time of the algorithms initialization, the processing time and the time required for the storage of the data in the file. For that it was defined a variable "tstart" with the current time, and then executed the Data Cube processing and recorded the final time variable “finish”. The calculation of the total time corresponds to the difference between these two variables and divided by the number of the machine clock cycles (per second).

The data sets used were five with three types of characteristics: one to examine the dispersive data, another for the density and three for the analysis of real data. For the first set we generated different levels of density, fixing the number of dimensions to six, the cardinality of each dimension equal to one hundred, varying the number of tuples in each set of two hundred thousand to one million.

For the second data set and for to analyze the dispersion of data, generate six sets of data with the variation of the zipf factor between 0 and 3 with an interval of 0.5, fixing the number of dimensions equal to six, a total of one million of tuples and varying the cardinality in each dimension with the following values: 900, 800, 700, 600, 500 and 400. The third data set is the set of real data with the weather [Hahn et al. 1994] recorded at various stations throughout the planet in September 1985. The fourth and fifth data set contains 1.2 million and 1.4 million respectively, corresponding to a data from a commercial company.

3.1 Compression rate

In these experiments we explore the benefits of the compression and the processing time of the algorithms BU-Condensed Cube and Prefix Cube. We analyzed the compression ratio versus the main factors of the data sets in particular, the number of cells in the base table (fact table), the cardinality and the number of dimensions. To have a more concrete idea of the algorithms performance, there was a need to compare the size of the file processed by the Data Cube without making any kind of compression.

![Fig. 8 - Compression ratio for real data sets.](image-url)
In the figure 8 we present the ratio compression result of the real data sets. The efficiency of the Prefix Cube in comparison with the BU – Condensed Cube is about 25.5%. Thus, we obtained an efficiency in terms of compression rate of about 25.41% of the algorithm Prefix Cube against the BU – Condensed Cube. The figure 9 presents the values obtained for the ratio compression of the algorithms for the data set with density characteristic. We observe that with the increasing of the dataset density the compression rate of the algorithm BU – Condensed Cube increases, in opposition to the Prefix Cube.

Fig. 9 – Compression ratio with density datasets.

The efficiency of the compression ratio by the technique of sharing the prefixes (Prefix Cube) is substantially better than that obtained by the BU – Condensed Cube. Another observation that can be drawn is that for the Prefix Cube algorithm with the increasing of the density tends to decrease the compression ratio. The reason for this phenomenon is that the Data Cube contains a small set of data (two hundred thousand tuples), causing the majority of pages in the files of the Data Cube are not met. This situation leaves space on the stored page occupying however space in the file relating to these pages. With the increasing density of the pages it will be more satisfied when the number of tuples increases by one million.

Fig. 10 – Compression ratio for the dispersion data.

To analyze the dispersion of the data, we use a data distribution with zipf method, and the results are presented in the figure 10. As we can see, so the zipf factor increases, the Cube Prefix has better performance in ratio compression than the BU – Condensed Cube. In terms of compression rate, the Prefix Cube algorithm is 30% better than the BU-Condensed Cube. Another observation that can be drawn is that the curve of the Cube Prefix is regular (without significativelly variations), suggesting that the Cube Prefix is not sensitive to variations in the data dispersion.

Fig. 11 – Processing time for real datasets.

3.2 Processing Time

The results for the processing time and compression of each algorithm are presented in figures 11, 12 and 13. Figure 11 presents the results for the processing time when using a set of real data. How we can analyze, for sets with few dimensions, the processing time is much lower than that obtained for sets with more dimensions. Furthermore, although for most sets the Prefix Cube algorithm is more efficient in terms of processing time compared to the BU – Condensed Cube. For data sets with a fewer dimensions the algorithm BU – Condensed Cube is more efficient occurring the opposite with the increasing of the dimension number. The reason is that the algorithm Prefix Cube spends more time on the analysis of the common prefixes among tuples when the dataset has few dimensions. This analysis of the efficiency of sharing common prefixes in the storage time is more important as the number of dimensions increases in the data set. The performance of the algorithms processing time for dense data sets are shown in figure 12 where the results point to greater efficiency by the BU – Condensed Cube algorithm compared with the Prefix Cube.

Fig. 12 – Processing time for datasets with density data.

Indeed, the task of processing the Prefix Cube is more difficult than for the BU – Condensed Cube and than the Data Cube, since in each iteration is necessary to examine the prefixes sharing between cells. This is why the efficiency (in terms of processing time) of the
BU-Condensed Cube algorithm is lower for data sets with a dense distribution.

Fig. 13 – Processing time for data sets with dispersed data.

To analyze the performance of the processing time for the datasets with dispersed data (figure 13) we can analyze that the algorithms are not sensitive to data sets with dispersed data since the time of processing of both BU – Condensed Cube and Prefix Cube are set to increase the dispersion factor, resulting in an approximate value between them.

3.3 Discussion
In our study, we implemented three algorithms: the Data Cube processing, the BU – Condensed Cube and the Prefix Cube. For the analysis of the density impact and the dispersion of the datasets in the algorithms performance we used data sets with uniform dispersion and distribution zipf [zipf 1949] respectively. In each set we used various levels of dispersion and density. In the analysis of density, when the dataset becomes more dense the reduction ratio of the BU – Condensed Cube algorithm increases by the effect of sharing prefixes more efficient in dense data sets. As the dataset becomes more dense the Prefix Cube reduces the tendency of increasing the compression ratio making it more efficient than the BU – Condensed Cube and adapts better to a dense data sets.

Another conclusion is that the compression ratio algorithms in both data sets with two hundred thousand cells are greater than the compression ratio with a million cells. This happens usually when the base has few cells and most memory pages are empty when are stored in files. For this reason, much space is left free causing an increase in compression ratio compared with the pages filled. In these experiments we analyze various levels of dispersion in the data and the algorithm more efficient in our tests was the Prefix Cube. Thus, on average the Prefix Cube is the best around 30% in terms of compression rate than the BU – Condensed Cube for the compression ratio algorithms when subjected to a real datasets.

For real data sets and in terms of efficiency, the Prefix Cube algorithm is well above the BU – Condensed Cube and when the number of dimensions involved in data set. This is because it needs to find more cells with the possibility of sharing prefixes. In terms of processing time for dense datasets and dispersed, and increasing the number of cells (and the cardinality of the dimensions) there’s no increase in the processing time.

4 Conclusion
The Data Cube corresponds to the materialization of the group-by operator in all dimensions of the fact tables of the Data Warehouses. This type of operations requires a high computational cost in order to process the Data Cube and also a large space to store all group-bys. Many algorithms are oriented for the Data Cube processing and compression as the: Dwarf, BU-Condensed Cube, Min Cube, Prefix Cube and Quotient Cube. In this paper we present a study of the difficulties of the Data Cube processing and compressing task and present the benchmarking of the Prefix Cube and Condensed Cube to compress the Data Cube. The results were very good, obtaining compression rates at about 70% of the total of the Data Cube. It will be interested in the future to compare this algorithms with the implementation of others algorithms.

References:
[8] Dong Xin, J. Han, X. Li, B. Wah.” Star-cubing: computing iceberg cubes by top-down and bottom-up integrations”. In VLDB, 2003.