Intelligent Data Compression Approach in Multidimensional Data Warehouse

Walid MOUDANI[1], Mohammad HUSSEIN[1], Mirna MOUKHTAR[1], and Félix MORA-CAMINO[2]
[1] Dept. of Business Information System, Faculty of Business, Lebanese University, Lebanon
[2] Air Transportation Department, ENAC, 31055 Toulouse, France

***Abstract*—The problem with MOLAP is that large tables should be loaded in main memory, which can slow the system, even saturate the memory. In this work, we present a new compression method, called BTC, for multidimensional data warehouses. Several methods have been proposed in the literature that can compress the data such as the Bitmap method. The main purpose of BTC is to improve the access time on data. This technique is based on the concept of B-Tree. We implemented this new method, and then we compared and analyzed it with Bitmap method used in the literature. Numerical results are obtained and presented in this paper.**

Keywords- Data Warehouse, Multi-dimensional databases, Data Compression, B-Tree component.

I. INTRODUCTION

Decision Support Systems have become a key contributor to mark a significant competitive advantage for businesses. Many companies have adopted data warehouses from operational databases. Thus, data warehouse analysis tools, On-Line Analytical Processing (OLAP), represent an effective solution for business intelligence. These systems are based on the multidimensional paradigm, based on the concepts of dimension, fact, measure and OLAP operators. These systems provide a multidimensional analysis of large amounts of data. Ever since the OLAP industry started, there has been an ongoing debate about the best way to store data for OLAP. One school of thought advocates storing and analyzing data using relational databases, which have long been known for their ability to scale to large amounts of data. This is known as Relational OLAP (ROLAP). In this case, analysis of data is done using SQL queries. The other school of thought says that multidimensional data processing should be done using a specialized storage format (called a multidimensional database, or MDDB) designed to quickly answer OLAP queries. This is known as Multidimensional OLAP (MOLAP) ([9], [16], [23]). The major benefit of MOLAP is that data is presented to the users in an intuitive multidimensional fashion that they can very easily access without needing to write complex and lengthy SQL. Further, because the storage format is optimized for multi-dimensional analysis, it may be possible to obtain much better performance than using SQL. The MOLAP is optimized for OLAP multidimensional analysis. This is a form of multidimensional hypercube that can represent data as an intersection of n dimensions; these dimensions may be more or less dense, thus characterizing the density or sparsity of the cube. In this work, we focus on the latter. Multidimensional data cubes are composed of dimensions and measures. The dimension values are treated as indices of multidimensional arrays. The position of the measured value in multidimensional arrays can be calculated by the size values: $R\left(D_1, \ldots, D_n \;;\; M_1, \ldots, M_k\right)$, where D_i represents the dimensions and M_j represents the measure. The large size of most data warehouses (typically hundreds of gigabytes to terabytes) results in significant storage costs and causes us to find an efficient compression technique for reducing the size of these warehouses. The problem associated to MOLAP is that large tables should be loaded in main memory, which can slow the system or saturate the memory. Thus, one solution is to use a compact structure for representing data in tables. Because of the existence of null values in these data, the result can be compressed with less disk space. On another critical issue in these schemes is how to quickly access data to answer queries.

Many compression methods have been adopted ([3], [7], [13], [19], [23]). Most of these methods take both the principle of *forward and backward mapping*. Thus, they provide access to data in its compressed form and perform operations on it directly without decompression. Similarly, many algorithms are developed to allow the use of aggregation operations directly on the compressed form. In general, compression methods used in the database are classic, such as: Header Compression method, and the method Run-Length Encoding bitmap compression method, etc.

The goal of this paper is to describe, new compression method named BTC (BTreeCube), for multidimensional data warehouses. This method aims to compress data in large relational databases by improving the access performance in the compressed cube. The remainder of the paper is organized as follows. In Section 2, we introduce the data wharehouse. Section 3 describes the related works on data compression. In section 4, we present the BTC compression method for multidimensional data warehouse. Section 5 discusses the performance evaluation. Finally, in section 6 we conclude and we describe future works.

II. DATA WAREHOUSE

A data warehouse is a centralized and universal of all corporate information ([6], [25]). This was to be a subject-oriented database, integrated and containing historical information from multiple data sources, non-volatile and exclusively for the processes of strategic decision support. In general, the data warehouse is maintained separately from operational databases of the organization for several reasons. The data warehouse supports online analytical processing (OLAP) and online transaction processing (OLTP). A data warehouse is based on a multidimensional data model which shows the data as a data cube. A data cube, such as sales, allows modeling and seeing the sales data in multiple dimensions ([12]). The representation of the fact table is in the form of a cube, where each axis represents a dimension.
To cope with new challenges, the company must collect, process, analyze information about its environment to anticipate. But this information produced by the company is superabundant, unorganized and scattered across multiple heterogeneous operating systems and can come from all marketplaces. It is essential to collect and standardize data to allow analysis of indicators to aid decision making. The purpose of the data warehouse is to define and incorporate an architecture that serves as the foundation for business intelligence applications. The technical infrastructure implementation is able to integrate, organize, store and intelligibly coordinate data generated within the information system (from production applications) or imported from outside the information system in which the end users derive relevant information using tools restitution and analysis.

In data warehouses, data are divided into facts and dimensions. The facts are the important entity (i.e. sales) and include measures that can be grouped (i.e. Price). The dimensions describe the facts; such a sale has the dimensions: product and Time magazine. Multidimensional data cubes are composed of dimension and measures. The dimension values are treated as indices of multidimensional arrays. The position of the measured value in multidimensional arrays can be calculated by the size values: $R(D_1, \ldots, D_n \ ; \ M_1, \ldots, M_k)$, where D_i represents the dimensions and M_j represents the measure (example: sales of products (Product, Time, Store)) ([23]). The cube can have several dimensions, theoretically no limit to the number of dimensions, typically from April to December cubic dimensions, more than three dimensions of the hypercube term is sometimes used. A cell is given combinations of dimension values, so that a cell can be empty (no data for a specific combination). A cube called "sparse" is a cube with a few cells non-empty and dense cube is a cube with many non-empty cells. Thus, it is useful to compress the cubes "sparse" while eliminating empty cells, which results in less disk space ([5]).

III. RELATED WORKS ON DATA COMPRESSION

The compression concept is to reduce the physical size of information ([2], [18], [26]).The compression method is intrinsically depends on the type of data to be compressed, For example, a method compression of an image is different of a method compression of audio file. It leads to use appropriate algorithm in order to optimize the data by using the considerations specific to the type of data to compress. A decompression is needed to reconstruct the original data using the inverse algorithm of the one used in compression. Data compression presents several advantages such as: reducing the storage capacity, reducing the transfer time, and enhancing the data security, etc.

The multidimensional data cube inherits the idea of compression from relational databases because it shares the same properties of sparsity, the numeric data types, and the large amounts of data. Many articles have shown the importance of data compression in particular for large scientific databases and statistics ([11], [24]). The advantages and disadvantages of data compression are discussed as well. Many compression methods have been presented ([1], [4], [7], [14], [15], [17], [19], [20], [21], [23]). In the remainder of this section, we present several compression methods described in the literature. An analysis of these methods is done while specifying their advantages and disadvantages. Also, we present a numerical example illustrating the functioning of each method.

III.1 Bitmap compression method

The bitmap compression method is to divide the logical database in two parts ([7], [8], [10]). The first part is a string of data sources that consists of a sequence of bits in which the bits “1” are references to a non-constant data from the database logic and bits “0” are references to constant data. The second part consists of non-constant values, i.e. values that cannot be deleted. This compression method is easy and simple to program. However, its mechanism based on *forward and backward mapping* must check all the bits in the bitmap string. So the complexity of access is of the order $O(N)$ where N is the number of bits in bitmap equivalent to the number of items in the database logic.

Applying this method in a multidimensional case, the compression process begins by creating a multidimensional cube of bits with the same size as the original cube and one-dimensional array to store non-empty values, called the value vector. Then we start reading sequentially each cell in the data cube: if the cell is

empty, the corresponding value in the multi-dimensional array of bits will be set to 0, otherwise it will be set to 1 and the value of measuring the origin value will be added to the vector value.

III.2 Run Length Encoding compression method

The compression method Run Length Encoding (RLE) is based on the repetition of consecutive elements ([19]). It is based on the mechanism of *forward and backward mapping* and it depend on the number of distinct values. The basic principle is to encode a first element giving the number of repetitions of a value, and then complete it by the value to repeat. This method is very simple to program but it is useful only in cases where there are many consecutive identical data.

III.3 Header compression method

The compression method based on Header Compression is a combination of constant values and non-constant. The vector L represents the logical database which consists of two values (0: constant to remove and V_i is non-constant) ([7]), and it leads to two vectors such as:

- The vector H contains the accumulated score of alternative values of non-constant and constant.
- The vector P contains the non-constant values.

The number of access in the "forward and backward mapping" mechanisms is of the order of $O(\log S)$ where S is the size of the vector H.

III.4 Bit compression method

This Bit compression method consists in compressing the attributes of numerical values ([21]). If the domain of values of an attribute is k (values ranging from 0 to k-1), the recommended number of ASCII characters to represent each attribute value is $[\log_{10} k]$. This requires $8\times[\log_{10} k]$ bit, is that each character is represented by one byte of 8 bits. In this method, each attribute is represented by $[\log_2 k]$ bits so there is a simplification of order $8\times[\log_{10} k]-[\log_2 k]$ bits.

III.5 Range Cube compression method

The Range cube method involves constructing a tree from an initial table. Two types of information are stored in a node of this tree: the values of dimensions and measures ([23]). These measures are an aggregation of tuples descendants of a node. Each leaf represents a distinct tuple. Each node can contain several common dimensions that constitute a key.
This method uses the correlation in the datasets to reduce the cost of aggregation. However, it is not an optimal compression method.

III.6 BAP compression method

A compression method such as BAP technique is presented in literature. Its principle is to provide an efficient compression mechanism where the "forward and backward mapping" is considered efficient. In this method, the physical database consists of three parts:

- Physics Vector (PV)
- Bit Vector (BV)
- Address Vector (AV)

The search for an item using the mechanism of "forward and backward mapping" requires a single disk access.

Part of PV:
Consider the following logic database: $DB=(x_1, x_2, \ldots, x_N)$. Consider the constant c that must be removed. The Physics Vector PV is the vector of non-distinct constants in BD. Thus the vector is represented as $PV=(y_1, y_2, \ldots, y_n)$ where y_j belongs to DB, y_j differs from c where $1\le j\le n, \quad n\le N$

Part of BV:
The Bit Vector, BV, is composed of N bits. It is represented as follows:

$$BV=(b_1, b_2, \ldots, b_N) \text{ where } b_i=1 \quad if \quad x_i\neq c \quad \forall\, 1\le i\le N$$
$$b_i=0 \quad if \quad x_i=c \quad \forall\, 1\le i\le N$$

This vector is compressed by applying the Golomb method which consists of two phases:

* *First phase:* It consists of dividing *BV* into sub-vectors, each containing *D* bits where *D* is a parameter chosen by the user.

* *Second phase:* Each sub-vector is stored, after compression, in a separate block on the disk. We choose an integer parameter m (defined so that the probability of occurrence of 0-bit is close to 0.5) and then we divide each sequence of "*r*" zeros consecutive bits into $\lceil r/m \rceil$ groups so that each group contains "m" bits except the last one that can contain a number less than "m" bits. Each group "m" is encoded by a single bit of value "1" except the last group that is encoded as $\lceil \log_2 m \rceil$ bits having value equal to: $r - m \times \lfloor r/m \rfloor$. The latter group is separated from antecedent groups by a bit having value "0". To increase the effectiveness of this method, "m" is an integer chosen so that the probability of finding "m" consecutive bits of value 0 must be close to 0.5.

Part of *AV*:

Dividing *BV* into sub-vectors leads to divide the database *DB* into "d" sections of size *D* where $d = \lceil N/D \rceil$. We define the Vector *AV* as:

$AV = (a_1, a_2, \ldots, a_d)$ such as:

- $a_1 = 0$
- a_i : represents the position in *PV* relative to the last non-constant found into the previous section (i-1) of *DB*
- $a_i = a_{i-1}$ in case where the previous section (i-1) of *DB* does not have non-constant.

This method allows for quick access when searching for data while preserving acceptable compression ratio. However, it does not work (for example, search data, application of aggregation operators) directly on compressed data without carrying out a decompression stage of the database.

IV. BTC COMPRESSION METHOD

In this section, the new method proposed to tackle with the problem of data compression is presented. It leads to improve the complexity of direct access to the compressed data set. Thus, the methodology adopted is described in details for the compression of multidimensional data cube, the implementation of the compression method and how to access data in a compressed data cube. In the remainder of this section we describe the B-tree and we present the concept of BTC method.

IV.1 Description and structure of B-Tree

A multi-way tree of order m is an ordered tree where each node has at most m children. For each node, if k is the actual number of children in the node, then (k-1) is the number of keys in the node. A B-tree is designed to branch out in this large number of directions and to contain a lot of keys in each node so that the height of the tree is relatively small. This means that only a small number of nodes must be read from disk to retrieve an item. The goal is to get fast access to the data, and implicitly this means reading a very small number of records.
Unlike a binary-tree, each node of a B-tree may have a variable number of keys and children. The keys are stored in non-decreasing order. Each key has an associated child that is the root of a sub tree containing all nodes with keys less than or equal to the key but greater than the preceding key. A node also has an additional right most children are the root for a sub tree containing all keys greater than any keys in the node. Each non-leaf node is of the form:

$$(p_1, k_1, p_2, k_2, \cdots k_{m-1}, p_m)$$

where

p_i is a pointer to the ith child, $1 \le i \le m$

k_i is a key value, $1 \le i \le m-1$, which are in the sorted order, $k_1 \prec k_2 \prec \cdots \prec k_{m-1}$, such that:

- all keys in the subtree pointed to by p_1 are less than k_1
- For $2 \le i \le m-1$, all keys in the subtree pointed to by p_i are greater than or equal to k_{i-1} and less than k_i
- All keys in the subtree pointed to by p_m are greater than (or equal) to k_{m-1}

VI.2 Concept of BTC method

Our proposed compression method, called, is based, somehow, on the bitmap compression method with a modification that improves the complexity of direct access to the compressed data set. In this method, the measure value and the associated indexes are represented in a specific structure which is a node. The data will be arranged in, almost, balanced B-tree which leads to a logarithmic complexity of access.

This BTC method, based on the concept of B-Tree has a complexity somehow better than the Bitmap method. As every node has multiple children (*m-ary*), then the depth decreases as branching increases, and the number of access also decreases. Therefore, the complexity related to the access time is estimated such as $O(\log_m n)$. This access time consumed is significantly reduced in BTC while comparing it to the one presented in Bitmap method.

A key value can be represented as a structure composed as follows:

```
struct vtree_node {
    long m; // measure value
    long [] tabindex ;//Array of size relative to the # of dimensions of the data cube
    };
```

Each key value is represented by a node in the B-tree, having the following structure:
- A nonzero value of the measure "m" in the fact table.
- A table "tabindex" of size relative to the number of the dimensions in the cube. It contains the associated indexes with a measure.

The keys' values are arranged by applying an adequate transformed function which dealt with the indexes. An integer value is generated to a given position of a measure in the data cube. For example, the following transformation function (f_1) is applied to deal with the case of a data cube with two dimensions $(D_i, i = 1\ldots 2)$, d_1 represents the relative index on dimension D_1 and d_2 represents the relative index on dimension D_2.

$$f_1:\ f(d_1, d_2) = (d_1 - 1) * |D_2| + d_2$$

The transformation function is generalized in the function f_n. This function is defined on n-dimensions data cube. A recursive function is performed in order to deal with any relative index $(D_i, i = 1 \cdots n)$.

$$fn:\ f(d_1, d_2, \ldots d_n) = \begin{cases} f(d_1, d_2, d_3, \ldots d_{n-1}) & if\ d_n = 1 \\ \prod_{i=1}^{n-1} |D_i| * (d_n - 1) + & if\ d_n \succ 1 \\ \qquad f(d_1, d_2, d_3, \ldots d_{n-1}) & \end{cases}$$

V. PERFORMANCE EVALUATION

The purpose of this section is to present the results of analysis performance of the BTC method proposed by comparison with the results of Bitmap method proposed in the literature. To test our BTC method, we considered a numerical example of a MOLAP data warehouse ("CubeSales") composed of three dimensions (dimPart, dimCustomer, and dimStoreLocation) and a single measurement value "Salesamount". In order to analyze the performance of BTC method, we have performed a numerical comparison with the Bitmap compression method used in literature. We are interested in our comparison to analyze a few key criteria such as:

- Time compression of a data cube
- Access time in a data cube
- Compression ratio of a data cube

V.1. Analysis of the compression time

The following table contains a comparison of the compression time that results from the two compression methods, BTC and (table 1). The results obtained by the BTC method are very encouraging compared to those obtained by the method of "Bitmap". Note that the compression time depends on the rate of "sparsity" considered in different cases. The compression time increases when the sparsity level decreases.

The above figures (1, 2, and 3) show a comparison of the compression time while taking into consideration different sparsity levels and also different size of dimensions. The results obtained from applying BTC method show significantly that this latter method is much better than Bitmap method.

Table 1: Summary of the compression time in the data warehouse “CubeSales”

DataSet	Dimension	Sparsity (%)	Bitmap	BTC
D1	***50 x 50 x 50***	*40*	*4.1*	*1.43*
D2		*30*	*4.34*	*1.53*
D3		*20*	*4.34*	*1.82*
D4		*10*	*4.79*	*2.07*
D5	***100 x 100 x 100***	*40*	*23.86*	*4.03*
D6		*30*	*24.78*	*5.56*
D7		*20*	*24.88*	*6.26*
D8		*10*	*25.42*	*7.05*
D9	***200 x 200 x 200***	*40*	*44.18*	*9.12*
D10		*30*	*45.08*	*9.39*
D11		*20*	*45.62*	*9.12*
D12		*10*	*46.78*	*10.24*

Figure 1: Summary of the compression time in a time in a data warehouse (50x50x50)

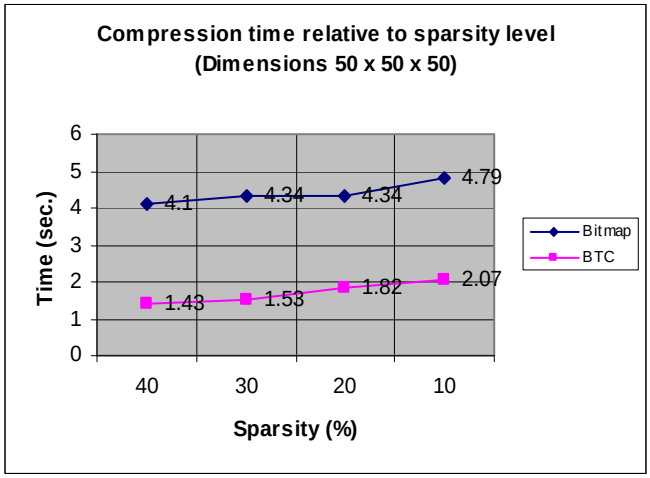


Figure 2: Summary of the compression data warehouse (100x100x100)

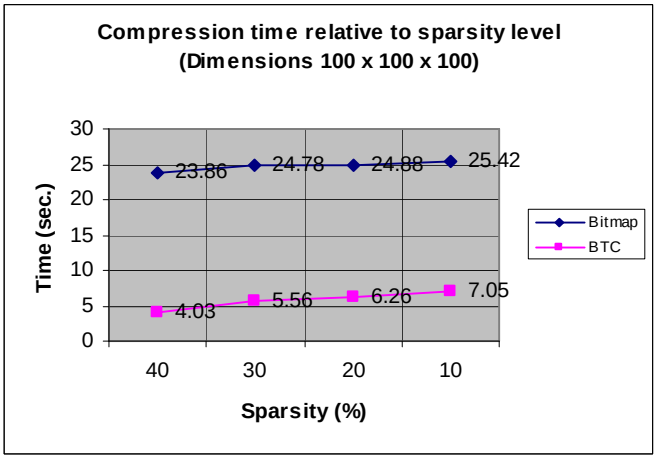
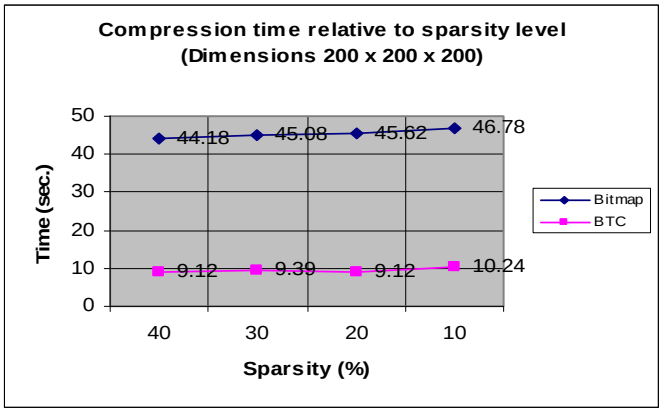


Figure 3: Summary of the compression time in a data warehouse (200x200x200)



V.2 Analysis of access times after compression

One of the main performance factors that should be considered seriously is related to the consuming time taken in order to access data after compression is made. The following table (Table 2) contains a comparison of performance results of two compressions method BTC and Bitmap. We notice that (Figure 4-7) show the access time in BTC method is better than Bitmap. We notice again that the access time (Table 2) in Bitmap decreases when the level of sparsity increases while this is not the case in the BTC method. This leads to distinguish that the B-tree is highly directed to balance the reduction much better the access time.

Table 2: Summarize the acces time in a data warehouse

DataSet	Dimension	Sparsity (%)	Bitmap	BTC
D1	***50 x 50 x 50***	*40*	*3.422*	*1.497*
D2		*30*	*3.453*	*1.387*
D3		*20*	*3.353*	*1.372*
D4		*10*	*3.151*	*1.361*
D5	***100 x 100 x 100***	*40*	*9.015*	*2.912*
D6		*30*	*9.161*	*2.835*
D7		*20*	*10.687*	*2.833*
D8		*10*	*11.11*	*2.784*
D9	***200 x 200 x 200***	*40*	*22.52*	*4.47*
D10		*30*	*22.91*	*4.65*
D11		*20*	*23.46*	*4.98*
D12		*10*	*24.05*	*5.18*

Figure 4: Summary of the access time relative to sparsity level 40%

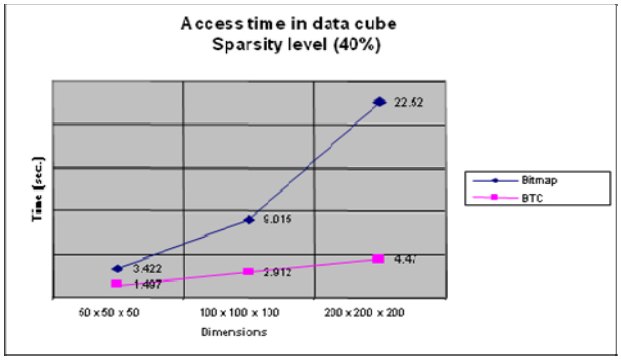


Figure 5: Summary of the access time relative to sparsity level 30%

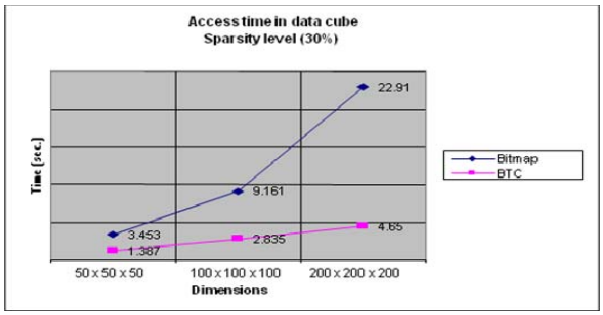


Figure 6: Summary of the access time relative to sparsity level 20%

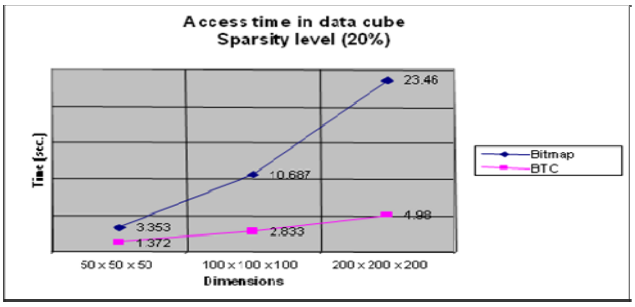
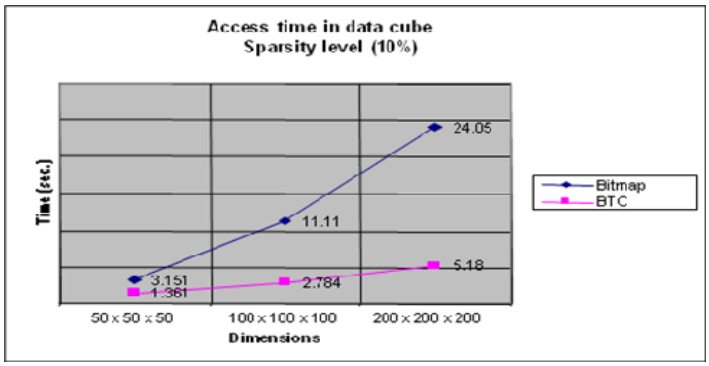


Figure 7: Summary of the access time relative to sparsity level 10%



The above figures show that the access time increases smoothly by using the BTC method while it increases so significantly by using the Bitmap method.

V.3. Analysis of compression ratio

Compression can be defined by the ratio of compression, i.e. the ratio's number of bits compressed data by the number of bits uncompressed data, this is called "compression ratio".

$$Compression\ Ratio = \frac{Compressed\ Size}{Uncompressed\ Size}$$

Sometimes the space savings is given instead, which is defined as the reduction in size relative to the uncompressed size:

$$Space\ Savings = 1 - \frac{Compressed\ Size}{Uncompressed\ Size}$$

The following tables (Tables 3 and 4) contain a comparison of performance results of two compressions BTC and Bitmap. Table 4 (respectively table 5) shows the compression ratio (resp. space savings) obtained by applying the method BTC with comparison to Bitmap method.

Note that (Figures 8-9) the "compression ratio" (respectively space savings) method in Bitmap is better than in BTC when we deal with a low level of sparsity. However, a data cube is more real or less dense; this leads to the conclusion that the application of the method Bitmap in these cases is more effective than the proposed one.

The figure 8 shows that the compression ratio in BTC method is better than the one in Bitmap method, especially when the sparsity is high. However, this advantage is switched toward Bitmap method when the data cube is dense. This statement is shown differently in figure 9 by displaying the savings space associated to the

Figure 8: Summary of the compression ratio relative to sparsity levels

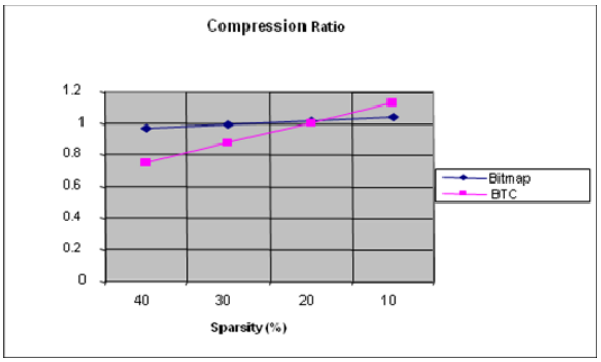


Figure 9: Summary of the space savings relative to sparsity levels

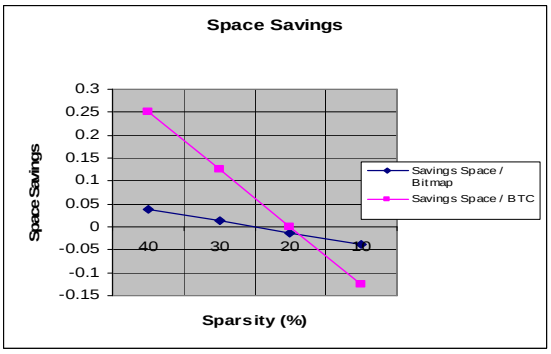


Table 3: Summary of the compression ratio relative to sparsity levels and dimensions

DataSet	Dimension	Sparsity (%)	Initial size (KB)	Bitmap	Ratio / Bitmap	BTC	Ratio / BTC
D1	***50 x 50 x 50***	*40*	*2000*	*1925*	*0.9625*	*1500*	*0.75*
D2		*30*	*2000*	*1975*	*0.9875*	*1750*	*0.875*
D3		*20*	*2000*	*2025*	*1.0125*	*2000*	*1*
D4		*10*	*2000*	*2075*	*1.0375*	*2250*	*1.125*
D5	***100 x 100 x 100***	*40*	*16000*	*15400*	*0.9625*	*12000*	*0.75*
D6		*30*	*16000*	*15800*	*0.9875*	*14000*	*0.875*
D7		*20*	*16000*	*16200*	*1.0125*	*16000*	*1*
D8		*10*	*16000*	*16600*	*1.0375*	*18000*	*1.125*
D9	***200 x 200 x 200***	*40*	*128000*	*123200*	*0.9625*	*96000*	*0.75*
D10		*30*	*128000*	*126400*	*0.9875*	*112000*	*0.875*
D11		*20*	*128000*	*129600*	*1.0125*	*128000*	*1*
D12		*10*	*128000*	*132800*	*1.0375*	*144000*	*1.125*

Table 4: Summary of the space savings relative to sparsity levels and dimensions

DataSet	Dimension	Sparsity (%)	Initial size (KB)	Bitmap	Space Savings / Bitmap	BTC	Space Savings / BTC
D1	***50 x 50 x 50***	*40*	*2000*	*1925*	*0.0375*	*1500*	*0.25*
D2		*30*	*2000*	*1975*	*0.0125*	*1750*	*0.125*
D3		*20*	*2000*	*2025*	*-0.0125*	*2000*	*0*
D4		*10*	*2000*	*2075*	*-0.0375*	*2250*	*-0.125*
D5	***100 x 100 x 100***	*40*	*16000*	*15400*	*0.0375*	*12000*	*0.25*
D6		*30*	*16000*	*15800*	*0.0125*	*14000*	*0.125*
D7		*20*	*16000*	*16200*	*-0.0125*	*16000*	*0*
D8		*10*	*16000*	*16600*	*-0.0375*	*18000*	*-0.125*
D9	***200 x 200 x 200***	*40*	*128000*	*123200*	*0.0375*	*96000*	*0.25*
D10		*30*	*128000*	*126400*	*0.0125*	*112000*	*0.125*
D11		*20*	*128000*	*129600*	*-0.0125*	*128000*	*0*
D12		*10*	*128000*	*132800*	*-0.0375*	*144000*	*-0.125*

V.4 Disscusion

After testing the two methods under different scenarios, we should evaluate the performance of these algorithms compared to the compression ratio achieved and the access time of the cube. The Bitmap method has a good compression ratio compared to BTC method when dealing with dense data cube. Regarding the access time, the method of Bitmap shows an increased access time compared to the BTC method. This is due to the sequential access to binary cube, resulting in a complexity of O(N) where N is the number of cell in the cube. Similarly, it is obvious that the size of the cube affects significantly the access performance in the Bitmap method while we can mention here that the access performance is not significantly affected by the size of the cube in our method. Our proposed method consumes less time than the Bitmap method in accessing compressed data cubes. The complexity of access is of the order $O(\log_m n)$ in the best case, which leads to an almost constant access time regardless the size and the level of sparsity of data cubes. On the other hand, we observe that the compression performance was significantly affected by size increasing of the cubes in the Bitmap method whereas in our approach this performance has been slightly affected. We can conclude that BTC method was not as good as the Bitmap method in relation to the factor of compression performance, but it has largely overcome the Bitmap method with respect to the performance factor for direct access to the cubes compressed.

VI. SUMMARY AND CONCLUSIONS

In this work, we have presented a new idea of compression technique in MOLAP data warehouses. Then we compared it with the Bitmap method. Therefore, we have presented an overview about the data warehouse as their importance in large companies. Then we have introduced the importance of data compression and its benefits, followed by an illustration of the main compression methods found in the literature. In order to analyze the performance of BTC, the Bitmap and BTC methods were compared in different scenarios. This has led to the following results:

- The BTC method is more effective in access to compressed data cube than Bitmap. It is almost independent of the size of data cube and the level of sparstiy, which is not the case for the Bitmap method.
- The BTC method is also better in the time compression of data cubes than the method of Bitmap. The compression time varies slightly depending on the level of "sparsity" considered in the two methods. Contrariwise, the method of Bitmap is largely affected by the size of the cubes unlike the method of BTC.
- Concerning the compression ratio, we conclude that BTC method does not give a compression ratio as good as the Bitmap method in dense cubes.

In perspective, we propose to use the binary tree approach to deal with this problem and drive it in such a manner to get a balanced binary tree in order to have a reduced complexity of order O(log n), which can lead to a better solution. Also, we propose to improve the compression ratio of this new method.

REFERENCES

[1] Aberg, J., & Shtarkov, Y.M.: Text compression by context tree weighting. Proc. Data Compression Conference, pages 377–386, Snowbird, Utah, March 25 – 27, (1997)

[2] Balakrishna, R.I.: Data Compression Support in Databases. In Proceedings of the 20th International Conference on Very Large Data Bases, pp. 695--704 , (1994)

[3] Bassiouni, M.A.: Data Compression in Scientific and Statistical Databases. IEEE Transactions on Software Engineering, vol. 11, no. 10, pp. 1047-1058, (1985)

[4] Chee-Yong, C., & Yannis, E.: Bitmap Index Design and Evaluation. ACM SIGMOD. ISSN:0163-5808, Vol. 27(2), pp. 355 – 366, (1998)

[5] Fangling, L., Yubin, B., Ge, Y. , Daling, W., & Yuntao, L.: An Efficient Indexing Technique for Computing High Dimensional Data Cubes. Lecture Notes in Computer Science, Springer Berlin, Volume 4016, pp. 557-568, (2006)

[6] Imhoff, C., Galemmo, N., & Geiger, J.G.: Mastering Data Warehouse Design: Relational and Dimensional Techniques. Published by Wiley Publishing, Inc., Indianapolis, Indiana, (2003).

[7] Jian-zhong, Li: A New Compression Method with Fast Searching on Databases. Proc. of the 13th VLDB Conference, Brighton, (1987)

[8] Johnson, Th.: Performance Measurements of Compressed Bitmap Indices. VLDB, pp. 278-289, 1999.

[9] Larry, P.: Improving Data Warehouse and Business Information Quality. New York, NY: Wiley Publishing, Inc., (1999)

[10] Lemire, D., Kaser, O., & Aouiche, K. : Sorting improves word-aligned bitmap indexes. Data & Knowledge Engineering, 69(1), (2010)

[11] Li, J., Srivastava, J.: Efficient Aggregation Algorithms for Compressed Data Warehouses. IEEE Transactions on Knowledge and Data Engineering, v.14 n.3, p.515-529, (2002)

[12] Lo, E., Kao, B., Ho, W.S., Lee, S.D., Chui, C.K., & Cheung, D.W.: OLAP on sequence data. SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, NY, USA, ACM, pp. 649–660., (2008)

[13] Maier, M., Lennon, C., Britt, T., & Schaefer, S.: Lightning Detection and Ranging (LDAR) system performance analysis. Sixth Conference on Aviation Weather Systems, Dallas, TX, Amer. Meteor. Soc., (1995)

[14] Nagapadma, R., & Kaulgud, N.: Three Dimensional Motion Vectorless Compression. IJCSNS International Journal of Computer Science and Network Security, Vol.9 No.4, (2009)

[15] Pasco, R.C.: Source Coding Algorithms for Fast Data Compression. PhD thesis, Dept. of Electrical Eng., Stanford University, (1976).

[16] Pedersen, T.B., Jensen, C.S.: Multidimensional Database Technology. Computer, vol. 34, no. 12, pp. 40-46, (2001)

[17] Reghbati, H.K.: An overview of Data compression techniques. Computer, Vol. 14, No. 4, pp. 71-76, (1981)

[18] Moffat, A.: Implementing the PPM data compression scheme. IEEE Trans. on Communications, 38(11):1917–1921, (1990)

[19] Salomon, D., Motta, G., & Bryant, D.: Data compression: The complete Reference. 4th Edition, Publisher Springer-Verlag, London, (2006).

[20] Szepkuti, I.: Difference Sequence Compression Of Multidimensional Databases, PERIODICA POLYTECHNICA ELECTRICAL ENGINEERING, (2002)

[21] Wee Keong, N.G., Chinya, V.R.: Block-Oriented Compression Techniques for Large Statistical Databases. IEEE Trans. on Knowledge and Data Engineering, Vol. 9, pp. 314-328, (1997)

[22] Willems, F.M.J., & Volf, P.A.J.: Reducing model cost by weighted symbol decomposition. IEEE Int. Symp. on Information Theory, page 338, Washington, D.C., US, June 24-29, (2001)

[23] Ying, F.: Range CUBE: Efficient Cube Computation by Exploiting Data Correlation. Proc. of the 20th ICDE Conference, (2004)

[24] Ziv, J., & Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Info. Theory, vol. IT-23 (3), pp. 337-343, (1977)

[25] Davis, K. and Gupta, A., Data Warehouses and OLAP: Concepts, Architectures, and Solutions, IRM Press, (2007).

[26] Wu, K., Otoo, E.J., & Shoshani, A.: Optimizing bitmap indices with efficient compression, ACM Transactions on Database Systems (TODS), v.31 n.1, p.1-38, (2006).