Analysis of Basic Data Reordering Techniques

Tan Apaydin[1], Ali Şaman Tosun[2,*], and Hakan Ferhatosmanoglu[1,**]

[1] The Ohio State University, Computer Science and Engineering
{apaydin,hakan}@cse.ohio-state.edu
[2] University of Texas at San Antonio, Computer Science
tosun@cs.utsa.edu

Abstract. Data reordering techniques are applied to improve the space and time efficiency of storage and query systems in various scientific and commercial applications. Run-length encoding is a prominent approach of compression in many areas, whose performance is significantly enhanced by achieving longer and fewer "runs" through data reordering. In this paper we theoretically study two reordering techniques, namely lexicographical order and Gray code order. We analyze these two methods in the context of bitmap indexes, which are known to have high query performances. We take into account the two commonly used bitmap encodings: equality and range. Our analysis indicates that, when we have all the possible data tuples, both ordering methods perform the same with equality encoding. However, Gray code achieves better compression with range encoding. Experimental results are provided to validate the theoretical analysis.

1 Introduction

Scientific data is mostly read-only and its volume can reach to the order of petabytes, e.g., astrophysics, genomic and proteomics, high energy physics. The techniques for maintaining the conventional databases usually do not apply to these applications, which brings up the need for effective indexing methods for efficient storage and retrieval. Bitmap indexes are compact index structures and they have been successfully applied to data warehouses and scientific databases by exploiting the property that scientific data are enumerated or numerical [6,11]. These structures have also been implemented in commercial Database Management Systems such as Oracle [1,2], Informix [4,7].

These indexes handle partial match and range queries very efficiently since they utilize the fast bitwise logical operations, which are directly supported by computer hardware. However, in order to maintain these advantages in large domains, effective compression schemes are applied. The most suited and widely used techniques are the adaptations of run-length based compression [10]. Besides reducing the data size, run-length compression has the benefit of partially avoiding the overhead of decompression in query processing for bitmap indexes [1,12]. Since run-length based compression schemes pack together the consecutive same-value-symbols, the compression ratio depends heavily on the occurrences of such patterns. Data can be reorganized/reordered to increase the length of the *runs* and improve the compression performance.

* Partially supported by US National Science Foundation (NSF) Grant CCF-0702728.
** Partially supported by US National Science Foundation (NSF) Grant IIS-0546713.

B. Ludäscher and Nikos Mamoulis (Eds.): SSDBM 2008, LNCS 5069, pp. 517–524, 2008.

A common approach is lexicographical sorting, which is used in traditional databases to preserve locality and avoid disk seeks. Several other reordering techniques have been successfully applied to increase the performance of compression in different domains as well [5,9]. For a boolean matrix, the objective of finding an order of data that minimizes the total number of runs in the columns of the matrix was shown to be NP-hard through a reduction to Traveling Salesperson Problem (TSP) [5]. Gray codes have been proposed as an efficient alternative to simple lexicographic or expensive TSP heuristics, and shown to achieve comparable compression to TSP, while running significantly faster than these heuristics [9].

In this paper we examine the effectiveness of (re-)ordering methods on the data compression performances. We provide the theoretical foundations and performance analysis of lexicographic and Gray code order in the context of bitmap indexes. Comparatively, lexicographic order and Gray code order are investigated for two encoding techniques that are commonly used in bitmap indices, namely equality and range encodings, and their relative performances are studied with data reordering in consideration.

Compared to their own equality encoding versions, our study reveals that both Gray code and lexicographic order achieve greater compression performances for range encoding. On the other hand, comparison of the two ordering methods leads to the following outcomes. With equality encoding, when we have all the possible data tuples, lexicographic order and Gray code order perform the same. However, Gray code order achieves better compression than lexicographic order when range encoding is used. We also provide experimental results to validate the theoretical analysis.

The organization of the paper is as follows. In Section 2 we briefly cover the background information about the impact of reordering schemes on the compression performance, and provide the preliminaries for the rest of the paper. Section 3 provides the analysis for the equality encoding bitmap model. We provide the theoretical study on the range encoding model in Section 4. Experimental results are presented in Section 5, and finally we conclude in Section 6.

2 Background

In this section, the background information for the data ordering and compression approaches are provided, and the related work is discussed.

Efficient storage of large boolean tables are achieved by utilizing the run-length based compression [10], which is the process of replacing the consecutive occurrences of a symbol by a single instance and a count. We define a *run* as a sequence of 0's followed by (not including) a 1 or end symbol, or a sequence of 1's followed by (not including) a 0 or end symbol. For instance, in Figure 1(a) the first column has 2 runs and the second column has 4 runs. Variations of the run-length compression technique are utilized in different domains in the literature. For example, for bitmap indexes, the two most popular compression schemes are BBC [1] and WAH [12]. BBC stores the compressed data in bytes while WAH stores in words. They are designed not only to decrease the bitmap index size but also to speed up the query execution performance while running the queries over the compressed data.

$$\begin{array}{l} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \end{array} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{array}{l} t_1 \\ t_2 \\ t_4 \\ t_3 \\ t_7 \\ t_8 \\ t_6 \\ t_5 \end{array} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

(a) Lexicographic (b) Gray code

Fig. 1. Example of tuple reordering

For a boolean matrix, long runs of 0 or 1 blocks are necessary for run-length based compression to be effective. In other words, the performance depends on the number of runs, therefore reordering techniques are utilized for improvement by packing the same-value bits together. Adaptations of the traveling salesperson problem (TSP) solutions have been applied to the large boolean matrices in [5]. In order to improve the bitmap index compression, Gray code ordering (GCO) is proposed as a data reorganization technique in [9]. GCO based approaches are known to be faster than TSP-based solutions.

The original Gray code for binary numbers is an encoding such that two adjacent numbers differ only by one bit (Hamming distance is equal to 1). For instance (000, 001, 011, 010, 110, 111, 101, 100) is a *binary reflected* Gray code. One can achieve a Gray code recursively as follows: $i)$ Let $S = (s_1, s_2, ..., s_n)$ be a Gray code. $ii)$ First write S forwards and then reflect S by writing it backwards, so that we have $(s_1, s_2, ..., s_n, s_n, ..., s_2, s_1)$. $iii)$ Append 0 to the beginning of first n numbers, and 1 to the beginning of last n numbers. For instance, take the Gray code (0, 1). Write it forwards and backwards, and we get: (0, 1, 1, 0). Then we add 0's and 1's to get: (00, 01, 11, 10).

Figure 1 illustrates the effect of running the GCO algorithm. On the left is the numeric (or lexicographic) order of a boolean matrix with 3 columns. In the rest of the paper, we refer to the lexicographic order shortly as *Lexico order*. GCO of the same matrix is presented on the right. As the figure illustrates, the aim of GCO is to produce longer and thus fewer runs than Lexico order. Figure 1(a) produces 14 runs (2 on the first column, 4 and 8 on the following columns) whereas Figure 1(b) has 10 runs (2 on the first column, 3 and 5 on the following columns).

We call a data set that has all the possible combinations of tuples as *full*. Table 1 is an example of a full data. Recall that the aim of GCO is to reorder the data so that the Hamming distances between the consecutive tuples will be 1. For a set of tuples there can be more than one order that have such property. Therefore, Gray codes are not unique. In this paper, to simplify the analysis, only the reflected GCO is taken into account.

Equality Encoding is the basic encoding scheme for bitmap indices, which is also known as Value-List index [8]. For an equality encoded bitmap index, data is partitioned into several bins, where the number of bins for each attribute could vary. If a value falls into a bin, this bin is marked "1", otherwise "0". Since a value can only fall into a single bin, only a single "1" can exist for each row of each attribute. Table 1 shows a two-attribute example such that the first attribute has 2 bins and the second attribute has

Table 1. Encoding example for two attributes with 2 and 3 bins

Tuple	Equality Encoding					Range Encoding				
	Attribute 1		Attribute 2			Attribute 1		Attribute 2		
	a	b	1	2	3	a	b	1	2	3
t_1 = (b, 3)	0	1	0	0	1	0	1	0	0	1
t_2 = (a, 2)	1	0	0	1	0	1	1	0	1	1
t_3 = (a, 3)	1	0	0	0	1	1	0	0	0	1
t_4 = (b, 2)	0	1	0	1	0	0	1	0	1	1
t_5 = (b, 1)	0	1	1	0	0	0	1	1	1	1
t_6 = (a, 1)	1	0	1	0	0	1	1	1	1	1

3 bins. The first tuple t_1 falls into the second bin of the first attribute and the third bin of the second attribute.

Another prominent encoding scheme is called *Range encoding* [3], which is also presented in Table 1. In this encoding, if a value falls into a bin b_i, all the greater bins and also b_i are marked "1"; and "0" otherwise. Range encoding performs better especially for single-sided range queries compared to equality encoding. For details we refer the reader to [3].

3 Equality Encoding

In this section, we investigate the behaviors of Lexico order and GCO schemes using equality encoding. Our main goal is to derive a formula for the total number of runs with full data. For the remaining of the paper, we use the terms *cardinality* and *number of bins* of an attribute interchangeably and they basically refer to the same value.

Define $F(x)$ as $F(x) = 3x - 2$. This function will be used to find the number of runs of an attribute as a function of its cardinality. The total number of attributes is denoted by A, and the cardinality of attribute i is denoted by C_i. For A attributes, where $A \geq 2$, following theorem presents the total number of runs for the full data using Lexico order.

Theorem 1. *For full data, number of runs in Lexico order using equality encoding is*

$$F(C_1) + \sum_{i=2}^{A} \left(F(C_i) \prod_{j=1}^{i-1} C_j - \left[(C_i - 2) \left((\prod_{j=1}^{i-1} C_j) - 1 \right) \right] \right)$$

Proof. Number of runs for the first attribute is $F(C_1)$. With full data i^{th} attribute can be considered as $\prod_{j=1}^{i-1} C_j$ separate chunks where the tuples in a chunk have the same value for the attributes $A_1, ..., A_{i-1}$. An example is given in Figure 2. In the example first attribute has a single chunk, second attribute has 2 chunks ($C_1 = 2$) and third attribute has 4 chunks ($C_1 \cdot C_2 = 2 \cdot 2 = 4$). Since there are $\prod_{j=1}^{i-1} C_j$ chunks and an attribute with C_i produces $F(C_i)$ runs, there are $F(C_i) \prod_{j=1}^{i-1} C_j$ runs in attribute i. However, this assumes that runs finish and start at chunk boundaries and can not be combined. Runs for the first and last columns of an attribute can not be combined. However, runs for other columns can be combined since runs are of the form: 0's followed by 1 followed by 0's. Trailing runs of 0's for one chunk can be combined with leading 0's of next chunk. There are $C_i - 2$ inner columns in each attribute and there are $\prod_{j=1}^{i-1} C_j - 1$ run

0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	0
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	0

Fig. 2. Example of chunks with 3 attributes each having 2 bins

merges for each inner column. Total number of run merges is $(C_i - 2)(\prod_{j=1}^{i-1} C_j - 1)$ and we subtract this to find the exact number of runs. □

For equality encoding with the full data, we next show that Lexico order and GCO produce the same number of runs.

Theorem 2. *For full data, the number of runs in GCO using equality encoding is equal to the number of runs in Lexico order, which is given in Theorem 1.*

Proof. Number of runs for the first attribute is $F(C_1)$. With full data, i^{th} attribute can be considered as $\prod_{j=1}^{i-1} C_j$ separate chunks where the tuples in a chunk have the same value for the attributes $A_1, ..., A_{i-1}$. Since there are $\prod_{j=1}^{i-1} C_j$ chunks and an attribute with C_i produces $F(C_i)$ runs, there are $F(C_i) \prod_{j=1}^{i-1} C_j$ runs in attribute i. For odd numbered attributes, binary numbers in a chunk appear in increasing order and for even numbered attributes, binary numbers in a chunk appear in decreasing order. In either case, number of runs for a column is the same. Above analysis assumes that runs finish and start at chunk boundaries and can not be combined. Rest of the proof is similar to the proof of Theorem 1. □

4 Range Encoding

In this section, we focus on range encoding and discuss the behaviors of Lexico order and GCO. Our main goal again includes deriving the total number of runs. In addition, we compare the compression performances of Lexico order and GCO both for equality and range encodings. Note that conversion of an equality encoded tuple T_i to its range encoded version $R(T_i)$ is a 1-1 transformation (see Table 1).

Total Runs for Lexico: Define function $E(x)$ as $E(x) = 2x - 1$, which will help deriving the number of runs of range encoding for both Lexico order and GCO. The formula for lexicographic order is given by the following theorem.

Theorem 3. *For full data, the number of runs in Lexico order of A attributes, where $A \geq 2$, using range encoding is*

$$E(C_1) + \sum_{i=2}^{A} \left(E(C_i) \prod_{j=1}^{i-1} C_j - \left[(\prod_{j=1}^{i-1} C_j) - 1 \right] \right)$$

Proof. The number of runs for the first attribute is $E(C_1)$. With full data, i^{th} attribute can be considered as $\prod_{j=1}^{i-1} C_j$ separate chunks where the tuples in a chunk have the

same value for the attributes $A_1, \ldots, A_{i-1}$. Since there are $\prod_{j=1}^{i-1} C_j$ chunks and an attribute with C_i produces $E(C_i)$ runs, there are $E(C_i) \prod_{j=1}^{i-1} C_j$ runs in attribute i. However, this assumes that runs finish and start at chunk boundaries and can not be combined. Runs for the last column can be combined since all the entries are 1's. Runs for other columns can not be combined since they all have a number of 0's followed by a number of 1's. There are $\prod_{j=1}^{i-1} C_j - 1$ run merges for the last column of the attribute. We subtract the number of run merges $(\prod_{j=1}^{i-1} C_j - 1)$ to find the exact number of runs. □

Equality Lexico vs. Range Lexico: For Lexico order, range encoding achieves better compression than equality encoding as shown by the following corollary.

Corollary 1. *For Lexico order of full data, range encoding produces fewer runs than equality encoding.*

Proof. Follows from the comparison of Theorems 1 and 3 using $E(C_i) < F(C_i)$ and $\left[(\prod_{j=1}^{i-1} C_j) - 1\right] < \left[(C_i - 2)\left((\prod_{j=1}^{i-1} C_j) - 1\right)\right]$. □

Total Runs for GCO: The total number of runs is given below. Tricky part of the derivation is again to find out how many of the runs merge. Since runs can cross the chunk boundaries (see Figure 2), we should avoid overcounting.

Theorem 4. *For full data, the number of runs in GCO of A attributes, where $A \geq 2$, in range encoding is*

$$E(C_1) + \sum_{i=2}^{A} \left(E(C_i) \prod_{j=1}^{i-1} C_j - C_i \left[(\prod_{j=1}^{i-1} C_j) - 1 \right] \right)$$

Proof. Number of runs for the first attribute is $E(C_1)$. With full data, i^{th} attribute can be considered as $\prod_{j=1}^{i-1} C_j$ separate chunks where the tuples in a chunk have the same value for the attributes $A_1, \ldots, A_{i-1}$. Since there are $\prod_{j=1}^{i-1} C_j$ chunks and an attribute with C_i produces $E(C_i)$ runs, there are $E(C_i) \prod_{j=1}^{i-1} C_j$ runs in attribute i. This analysis assumes that runs finish and start at chunk boundaries and can not be combined. However, the runs for all the bins of an attribute can be combined. Since there are C_i bins in the attribute and there are $\prod_{j=1}^{i-1} C_j - 1$ run-merges for each inner column, we subtract total number of run-merges of the attribute given by $C_i(\prod_{j=1}^{i-1} C_j - 1)$ to find the exact number of runs. □

Equality GCO vs. Range GCO: Range encoding using GCO produces fewer runs. In other words, the conversion from equality encoding into range encoding reduces the number of runs for full data. Since the conversion is 1-1, range encoding can be used as a way to achieve further compression, which is an open research question.

Corollary 2. *For GCO of full data, range encoding produces fewer runs than equality encoding.*

Proof. Follows from comparison of Theorems 2 and 4 using $E(C_i) < F(C_i)$ and $\left[(\prod_{j=1}^{i-1} C_j) - 1\right] < \left[C_i\left((\prod_{j=1}^{i-1} C_j) - 1\right)\right]$. □

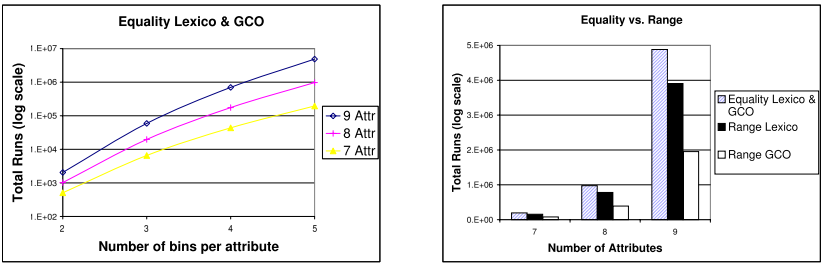
Range Lexico vs. Range GCO: For range encoding, finally we compare Lexico order and GCO in the following corollary.

Corollary 3. *GCO produces fewer number of runs than Lexico for range encoding.*

Proof. Follows from comparison of Theorems 3 and 4. □

5 Experimental Results

For our experiments, we used full data sets with varying number and cardinality of attributes. In Figure 3(a), we present the total runs in log scale as a function of the attribute cardinality. The larger the cardinality, the higher the number of runs. (Recall that Lexico and GCO have the same number of runs for equality encoding.) We also repeated the experiment with different number of attributes (7, 8 and 9). Again, increasing the number of attributes leads to higher number of runs. For example, 7 attributes each with 5 bins produce 195,331 number of runs, 8 attributes each with 5 bins produce 976,584 runs, and 9 attributes produce 4,882,837 runs.



(a) Equality Encoding (b) Equality vs. Range

Fig. 3. Experimental results with different number of attributes with different cardinalities

For a comparison between equality and range encodings, we present Figure 3(b) where each attribute has 5 bins. Since Lexico and GCO perform the same for equality encoding, we simply combined them and named that as *Equality Lexico & GCO*. Note that, among the three approaches (namely Equality Lexico & GCO, Range Lexico, and Range GCO), the best performance is achieved by Range GCO. For an example in Figure 3(b), the values for 9 attributes are as follows: Equality Lexico & GCO produces 4,882,837 runs. Range Lexico has 3,906,257 runs, and Range GCO produces 1,953,169 runs. For range encoding, note that the number of runs for Lexico is about twice the number of runs for GCO.

6 Conclusion

In this paper we studied the effectiveness of reordering methods that are applied for better compression performances in databases. High energy physics, astrophysics, genomic and proteomics are some of the applications that produce large data sets, which

bring up the need for effective indexing techniques for efficient storage and querying. Bitmap indexes are practical structures that are prominently used for querying scientific data. In the literature, in order to reduce the sizes of these indexes, run-length based compression schemes are developed whose performances are improved by data reordering approaches.

We provided the theoretical foundations and performance analysis of lexicographic order and Gray code order in the context of bitmap indexes. Comparatively, lexicographic order and Gray code order are investigated for two encoding techniques that are commonly used in bitmap indices, namely *equality* and *range* encodings, and their relative performances are studied with data reordering in consideration.

Our study reveals that both Gray code and lexicographic order achieve greater compression performances for range encoding compared to their own equality encoding versions. On the other hand, comparison of the two ordering methods leads to the following observations. With equality encoding, when we have all the possible data tuples, lexicographic order and Gray code order perform the same. However, Gray code order achieves better compression than lexicographic order when range encoding is used. We also provided experimental results to validate the theoretical analysis.

References

1. Antoshenkov, G.: Byte-aligned bitmap compression. In: Data Compression Conference, Nashua, NH. Oracle Corp. (1995)
2. Antoshenkov, G., Ziauddin, M.: Query processing and optimization in oracle rdb. The VLDB Journal 5(4), 229–237 (1996)
3. Chan, C.Y., Ioannidis, Y.E.: Bitmap index design and evaluation. In: Proceedings of the 1998 ACM SIGMOD international conference on Management of data, pp. 355–366. ACM Press, New York (1998)
4. Informix. Decision support indexing for enterprise datawarehouse, `http://www.informix.com/informix/corpinfo/-zines/whiteidx.htm`
5. Johnson, D., Krishnan, S., Chhugani, J., Kumar, S., Venkatasubramanian, S.: Compressing large boolean matrices using reordering techniques. In: VLDB 2004 (2004)
6. Chen, J., Wu, K., Koegler, W., Shoshani, A.: Using bitmap index for interactive exploration of large datasets. In: Proceedings of SSDBM (2003)
7. O'Neil, P.: Informix and Indexing Support for Data Warehouses. Database Programming and Design 10, 38–43 (1997)
8. O'Neil, P., Quass, D.: Improved query performance with variant indexes. In: Proceedings of the 1997 ACM SIGMOD international conference on Management of data, pp. 38–49. ACM Press, New York (1997)
9. Pinar, A., Tao, T., Ferhatosmanoglu, H.: Compressing bitmap indices by data reorganization. In: ICDE, pp. 310–321 (2005)
10. Salomon, D.: Data Compression: The Complete Reference, 3rd edn (2004)
11. Stockinger, K., Shalf, J., Bethel, W., Wu, K.: Dex: Increasing the capability of scientific data analysis pipelines by using efficient bitmap indices to accelerate scientific visualization. In: Proceedings of SSDBM (2005)
12. Wu, K., Otoo, E.J., Shoshani, A.: Optimizing bitmap indices with efficient compression. ACM Trans. Database Syst. 31(1), 1–38 (2006)