Multi-level and Multi-component Bitmap Encoding for Efficient Search Operations

Madhu BHAN[1] , Dr.RAJANIKANTH K [2], Dr. Suresh KUMAR T.V[3]
[1,2]M.S.Ramaiah Institute of Technology, Department of Computer Applications, India.
[3]Visvesvaraya Technological University, India
madhoobhan@yahoo.co.in

The growing interest in data warehousing for decision makers is becoming more and more crucial to make faster and efficient decisions. On-line decision needs short response times. Many indexing techniques have been created to achieve this goal in read only environments. Indexing technique that has attracted attention in multidimensional databases is Bitmap Indexing. The paper discusses the various existing bitmap indexing techniques along with their performance characteristics. The paper proposes two new bitmap indexing techniques in the class of multi-level and multi-component encoding schemes and prove that the two techniques have better space–time performance than some of the existing techniques used for range queries. We provide an analytical model for comparing the performance of our proposed encoding schemes with that of the existing ones.

Keywords: *Bitmap encoding, Datawarehouse, multi-level indexing, multi-component indexing, On-Line Analytical Processing.*

1 Introduction

While the query performance issues of on-line transaction processing (OLTP) systems have been extensively studied and are pretty much well-understood, the state-of-the-art for data warehouse systems is still evolving as indicated by the growing active research in this area [1]. In particular, Data warehouse systems operate in read-mostly environments, which are dominated by complex adhoc queries that have high selectivity factors [2]. Due to large size of the data warehouse and the complexity of queries, quick response time plays an important role as timely access to information is the basic challenge to match the pace of the query results with the speed of thought of the user. From various methods available to improve performance, indexing ranks very high [3]. Indexes are database objects associated with database tables and created to speed up access to data within table. Index space and access time play an important role in choosing an indexing technique in data warehouse. If the space used by an index is large then the results are achieved in short time on the other hand if the space used by the index space is small then the results are achieved in greater amount of time. So there is a trade-off between the time consumed and the space used by a particular index. A committed approach to answer complex queries swiftly in Data warehouse systems is the use of bitmap indexing [4], [5] and [6]. Bitmap manipulation techniques have already been used in some commercial products [7] to speed up query processing. The basic bitmap index uses each distinct value of the indexed attribute as a key, and generates one bitmap containing as many bits as the number of records in the data set for each key [8]. The advantage of bitmap index is that complex selection predicates can be computed very quickly by performing bit-wise AND, OR and NOT operations on bitmap indices. Bitmaps are

well supported by hardware and are easy to compress. Each individual bitmap is small and frequently used ones can be cached in memory. This property of bitmap has led to considerable interest in their use in Decision Support systems. The size of a basic bitmap index is relatively small for low-cardinality attributes, such as "gender," "types of cars sold per month," or "airplane models produced by Airbus and Boeing." However, for high-cardinality attributes such as "temperature values in a supernova explosion," the index sizes may be too large to be of any practical use [9]. In the literature, there are three basic strategies to reduce the sizes of bitmap indices: (1) using more complex bitmap ***encoding*** methods to reduce the number of bitmaps or improve query efficiency, (2) ***compressing*** each individual bitmap, and (3) using ***binning*** or other mapping strategies to reduce the number of keys. Various bitmap indexes have been designed for different query types, including range queries, aggregation queries, and OLAP-style queries. However, as there is no overall best bitmap index over all kinds of queries, maintaining multiple types of bitmap indexes for an attribute may be necessary in order to achieve the desired level of performance. While the gains in query performance using a multiple-index approach might be offset by the high update cost in OLTP applications, this is not an issue in the read-mostly environment of data warehouse applications. In the remaining of this paper, we first present in Section 2 a review of different bitmap indexing strategies. We discuss the three basic encoding techniques namely Equality encoding, Range encoding and Interval encoding along with their performance characteristics in Section 3. In Section 4 and 5, the proposed multi-level encoding and multi-component encoding technique is defined with an analytical model. Section 5 concludes the paper with future enhancements of the proposed techniques.

2 Related work.

Various bitmap indexes have been demonstrated to significantly speed up searching operations in data warehousing, On-Line Analytical Processing (OLAP), and many scientific data management tasks [10] and [11 . This has led a number of commercial database management systems (DBMS) to support bitmap indexes [7]. However, most of the bitmap index implementations in commercial DBMS are relatively simple, such as the basic bitmap index or the bit-sliced index. There is a significant number of promising techniques proposed in the research literature that have not gained wide acceptance yet. A bitmap index typically uses a combination of three types of strategies namely encoding, binning and compression, though it is common to omit one or two. For example, the first commercial implementation of a bitmap in Model 204 uses equality encoding without binning or compression [8].

- **Encoding**: In the simplest encoding, a bitmap corresponds to exactly one attribute value. This encoding is known as Equality encoding where i-th bit is set to 1 if the i-th row of the base table has a value for the indexed column. It is possible to reduce the number of bitmaps by using a different encoding method. Fig 1(a) shows an example of encoded bitmap index. Assume that the attribute domain given by the table T is{a, b, c}. A simple bitmap index uses four bitmap vectors whereas an encoded bitmap index uses $\lceil \log_2 3 \rceil = 2$ bitmap vectors plus a mapping table. It encodes the values from a simple bitmap index by means of Huffman encoding. Thus we see that for an attribute with C distinct values we use only $\log_2 C$ encoded bitmap vectors instead of C bitmap vectors. We assume that we have a fact table SALES with N tuples and a dimension table PRODUCT with 12,000

different products. If we build a simple bitmap index on PRODUCT, It will require 12,000 bitmap vectors of N bits in length. However, if we use encoded bitmap indexing we only need $\lceil \log_2 12{,}000 \rceil = 14$ bitmap vectors plus a mapping table. It is a very significant reduction of the space complexity. Other common encoding schemes include range encoding and interval encoding [2]. More sophisticated encoding schemes can be generated from the above three basic encoding schemes, Equality encoding, Range encoding and Interval encoding. One approach of extending the basic encoding schemes is the multi-level encoding which can be viewed as using hierarchy of levels with different encoding techniques. Another strategy is to decompose the attribute value into several components and encode each component using a basic encoding scheme. These are called as multi-component encodings [12] and [13]. The best known example of such an encoding is the binary encoding which is also known as the bit-sliced index. Finding the optimal encoding method that balances query performance and index size remains an interesting challenge.

- **Compression:**

Compressing each bitmap in a bitmap index can save space. A lossless compression method can be used for this purpose. There has been considerable amount of work done on this subject [14] and [15] . For example, most generic text compression methods, such as LZ77, are effective in reducing the index size on disk, but they can also significantly increase the time required to answer a query, because the compressed bitmaps have to decompressed before being used in logical operations. Bitmap compression algorithms typically employ run-length encoding such as the Byte-aligned Bitmap Code and the Word-Aligned Hybrid code [16]. These compression methods require very little effort to compress and decompress. The Byte-Aligned Bitmap Code (BBC) can compress bitmaps and at the same time it also reduces the query response time. The BBC compressed basic bitmap index is implemented in ORACLE DBMS. The Word-Aligned Hybrid (WAH) code has been shown to outperform BBC in most cases[18]. This method trades some space for more efficient CPU operations. In one set of tests, it was shown to use about 50% more space than BBC, but answered queries 10 times faster on average More importantly, bitmaps compressed with WAH, BBC, PLWAH and CONCISE can directly participate in bitwise operations without decompression. This gives them considerable advantages over generic compression techniques such as LZ77. Fig1 (b) shows an WAH bit vector representing 128 bits. Assuming that computer word length is 32 bits, each literal word stores 31 bitmaps from the bitmap and each fill word represents a fill with a multiple of 31 bits. The second line in Figure shows how the bitmap is divided into 31-bit groups and the third line shows the hexadecimal representation of the groups. The last line shows the values of the WAH words. The logical operations can be directly performed on the compressed bitmaps and the time needed by one such operation on two operands is related to the sizes of the compressed bitmaps. Extended work on WAH compression has shown further improvements in performance of query processing. Different compression methods can be used and each may have a drastically different query processing costs.

- **Binning:**

In Binning, bitmap indices are built on attribute ranges rather than on distinct attribute values. For high-cardinality columns, it is useful to bin the values, where each bin covers multiple values and build the bitmaps to represent the values in each

bin [17] .The advantage of this approach is that a lower number of bitmap vectors is required. On the other hand, parts of the original data (candidates) have to be read from disk in order to answer the queries correctly. This process is called candidate check . This approach reduces the number of bitmaps used regardless of encoding method. However, binned indexes can only answer some queries without examining the base data. An example of a bitmap index with bins is given in Figure 1(c). Assume that we want to evaluate the query 37 <= x < 63. Bins 1, 2 and 3 contain the relevant data values. The bin in which a query boundary falls is known as an edge bin. Thus bins 1 and 3 are edge bins since they contain also irrelevant values, answering this query involves checking the values on disk corresponding to the four "1-bits" in these two columns. In this example only one of the four values qualifies, namely, 61. We call this additional step the candidate check. As we can see from this example, the cost of performing a candidate check on an edge bin is related to the number of "1-bits" in that bin. The process of checking the base data is known as the candidate check. In most cases, the time used by the candidate check is significantly longer than the time needed to work with the bitmap index [18] and [19]. Therefore, binned indexes exhibit irregular performance. They can be very fast for some queries, but much slower if the query does not exactly match a bin. The key advantage of binning is that it may reduce index sizes

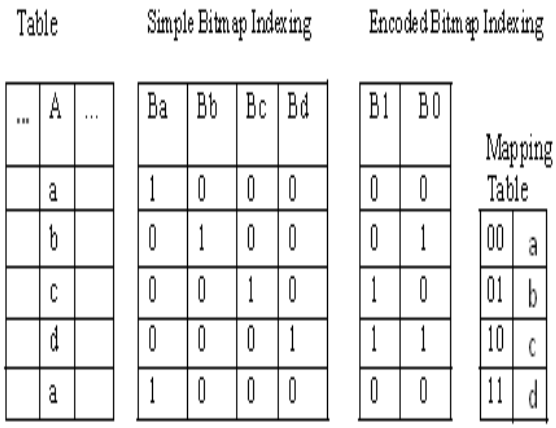
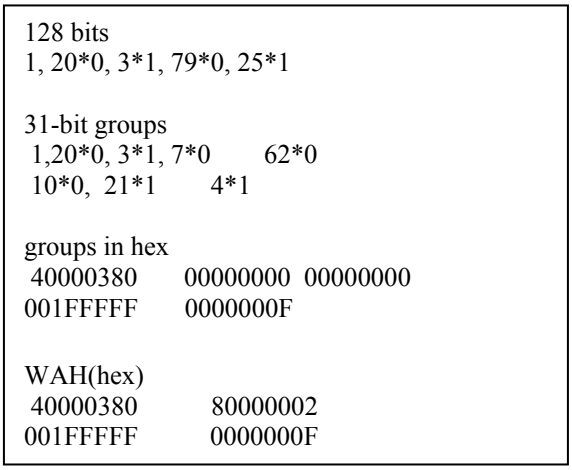


Fig 1(a)



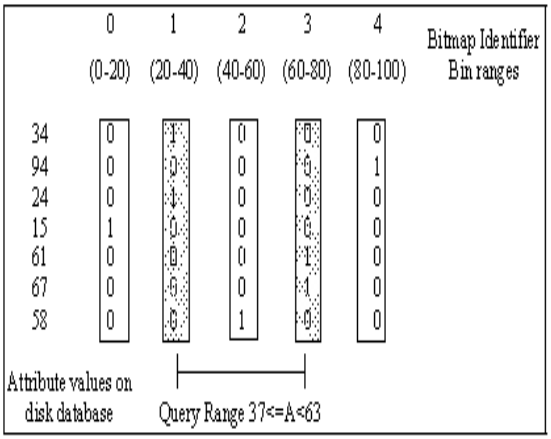


Fig. 1(c)

however, the disadvantage is that the index is no longer able to fully resolve all queries Among the three, encoding is by far the largest category, however their impacts on the overall index performance are less

studied than those of binning or compression. For this reason, studying the encoding methods is more likely to lead to the best bitmap index method.

3 Comparison of basic Encoding Schemes

We first review the three existing bitmap encoding schemes, equality encoding, denoted by E, range encoding denoted by R and interval encoding denoted by I. These schemes have been described in several papers under different names [1], 2] and [13]. Equality encoding is the most fundamental and common bitmap encoding scheme. It consists of C bitmaps E = {EO, El,. . . ,EC-1}, where each bitmap Ev = {v}. Consider an attribute A of a relation R, where the attribute cardinality is c. For simplicity and without loss of generality, the domain of A is assumed to be a set of consecutive integers from 0 to C - 1. This allows us to use set operators and logical operators interchangeably. The logical operators AND, OR, and XOR are denoted by Λ, V, and ⊖ respectively. For example, Figure 2(a) shows the Equality-encoded bitmap index. The leftmost column shows the row ids (RID) for the data values represented by the projection index on an attribute A with cardinality C = 10 of a 12 record relation R. Figure 2(b) shows the Equality encoded bitmap index for the data in Figure 2(a), where each column represents an equality-encoded bitmap Ev associated with an attribute value v. This strategy is the most efficient for equality queries such as "A = 3 which needs only one bitmap E2 to be accessed. The Range encoding and Interval encoding techniques are optimized for one-sided and two-sided range queries, respectively. An example of a one-sided range query (1RQ) is "A <= 3". A two-sided range query (2RQ), for instance, is "6 < A < 8". A comparison of an Equality encoding, Range encoding and Interval encoding is given in Figure 2 . Let us look at the encoding of value 2, in Equality encoding and Range encoding first, which is highlighted in the Figure2(a) and Figure 2(b). For Equality encoding, the third bitmap is set to "1" (E2), whereas all other bits on the same horizontal line are set to "0". For the Range-encoded bitmap index, all bits between bitmap R2 and R8 are set to "1", the remaining bits are set to "0". Range encoding is very efficient for evaluating range queries. Consider, for instance, the query "A <= 4". In this case, at most one bitmap, namely bitmap R4, has to be accessed (scanned) for processing the query. All bits that are set to "1" in this bitmap fulfil the query constraint. On the other hand, for the Equality encoded bitmap index, the bitmaps E0 to E4 have to be ORed together (via the Boolean operator OR). This means that, Range encoding requires at most one bitmap scan for evaluating range queries, whereas Equality encoding requires in the worst case C/2 bitmap scans, where C corresponds to the number of bitmaps. Since one bitmap in Range encoding contains only "1"s, this bitmap is usually not stored. Therefore, there are C-1 bitmaps in a range-encoded index. The Interval encoding, I, is optimal for the two sided Range queries. In Range encoding, each bitmap Ri = [0, i], and each 2RQ -query is evaluated by

	$\pi_A(R)$	E^9	E^8	E^7	E^6	E^5	E^4	E^3	E^2	E^1	E^0
1	3	0	0	0	0	0	0	1	0	0	0
2	2	0	0	0	0	0	0	0	1	0	0
3	1	0	0	0	0	0	0	0	0	1	0
4	2	0	0	0	0	0	0	0	1	0	0
5	8	0	1	0	0	0	0	0	0	0	0
6	(2)	0	0	0	0	0	0	0	(1)	0	0
7	9	1	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1
9	7	0	0	1	0	0	0	0	0	0	0
10	5	0	0	0	0	1	0	0	0	0	0
11	6	0	0	0	1	0	0	0	0	0	0
12	4	0	0	0	0	0	1	0	0	0	0

Fig 2 (a)

	$\pi_A(R)$	R^8	R^7	R^6	R^5	R^4	R^3	R^2	R^1	R^0
1	3	1	1	1	1	1	1	0	0	0
2	2	1	1	1	1	1	1	1	0	0
3	1	1	1	1	1	1	1	1	1	0
4	2	1	1	1	1	1	1	1	0	0
5	8	1	0	0	0	0	0	0	0	0
6	(2)	1	1	1	1	1	1	0	0	0
7	9	0	0	0	0	0	0	0	0	0
8	0	1	1	1	1	1	1	1	1	1
9	7	1	1	0	0	0	0	0	0	0
10	5	1	1	1	1	0	0	0	0	0
11	6	1	1	1	0	0	0	0	0	0
12	4	1	1	1	1	1	0	0	0	0

Fig 2(b)

	$\pi_A(R)$	I^4	I^3	I^2	I^1	I^0
1	3	0	1	1	1	1
2	2	0	0	1	1	1
3	1	0	0	0	1	1
4	2	0	0	1	1	1
5	8	1	0	0	0	0
6	2	0	0	1	1	1
7	9	0	0	0	0	0
8	0	0	0	0	0	1
9	7	1	1	0	0	0
10	5	1	1	1	1	0
11	6	1	1	1	0	0
12	4	1	1	1	1	1

Fig 2(c)

operating on an appropriate pair of bitmaps: [x, y] = R^y (+) R^{x-l}. The Interval encoding scheme based on range encoding consists of [C/2] bitmaps I = $\{I^0, I^1, I.^2 \ . \ . \ , I^{[C/2-1]}$, where each bitmap I^j =[j, j + m], and m = [C/2] - 1. The Interval encoding, I, is optimal for the two sided range queries. Figure 2(c) shows the Interval encoded bitmap index for the data in Figure 2(a). A 2RQ, in general, is evaluated by operating on a pair of bitmaps: [x,y] = $I^x \Lambda I^{y-m}$. The Interval-encoding scheme[2] reduces the number of bitmaps only by a factor 2 while still guaranteeing at most a two-scan evaluation for any query . Thus, other techniques are needed to make bitmap indices practical for high cardinality attributes [9]. The encoding method that produces the least number of bitmaps is Binary encoding. This encoding method uses only $\log_2 C$ rather than C/2 bitmaps, where C is the attribute cardinality. The advantage of this encoding is that it requires much fewer bitmaps than Interval encoding. However, to answer a range query, using interval encoding one has to access only two bitmaps whereas using binary encoding one usually has to access all bitmaps. An Equality encoded index may access a large number of bitmaps to answer a Range query, but the bitmaps are usually relatively easy to compress, while the Range encoding and the Interval encoding access fewer bitmaps to answer a range query, but they produce bitmaps that are hard to compress. Equality encoding requires C bitmaps, Range encoding requires C-1 bitmaps and Interval encoding reduces the number of bitmaps only by a factor of 2. Controlling the size of bitmap indicies is crucial to make bitmap indicies practical for high cardinality attributes. We find that Equality encoded index accesses larger number of bitmaps to answer a query but here it is easy to compress the bitmaps, while Range encoding and Interval encoding access few bitmaps ,but the bitmaps produced are hard to compress. Therefore, it is worthwhile to explore strategies that combine the advantages of the three basic bitmap indexing techniques. Strategies like multi-level encoding and multi-component encoding have been proposed by authors to reduce the index size and bitmap scans.

4 Multi-level Encoding

We find that Equality encoded index accesses larger number of bitmaps to answer a query but it is easy to compress the bitmaps, while Range encoding and Interval encoding access few bitmap , but the bitmaps produced are hard to compress

[15],[16]. To combine the advantages of these encoding techniques we can explore multilevel encoding techniques. We can think of a multi-level encoding as encoding at multiple levels where each level can be encoded separately using any encoding method. In previous works [6] we have seen that the multi-level encoding, when used with binning methods require candidate checks, which resulted in performance measurements that do not truly represent the characteristics of the encoding schemes. In this paper, we study the multi-level encoding with binning, which removes the need for candidate checks. This allows a better understanding of the performance characteristics of these multi-level encodings. We take an analytical approach in our study which allows us to compare various parameters like number of bins and cardinality of attributes.

- **Methodology**

Conceptually we can think of our multilevel bitmap technique as formed of equality encoding with binning in the first level and followed by binary encoding at the second level. Figure 3 shows an example of our multi-level encoding techniques with the same attributes values as given in the above example(Figure 1). To answer a query we first scan the equality encoded bitmaps(E_{0-1},E_{2-3},E_{3-4}……,E_{8-9}). Based on these bitmaps we need to scan Binary bitmap vector B. Consider for instance the Query A = 3. Our multi-level coding first access bitmap E_{2-3}. Then corresponding to 1's in E_{2-3} it scans binary bitmap B. The 1's in B indicate 3 and 0's in B indicate 2 as shown in mapping table. Thus we see that with C =10 and bin size n =2 we require 5(C/2) + 1 bitmap vectors scans.

1. Equality Encoding

Case of equality queries(A=3) :-

TIME: One bitmap E^2 needs to be scanned.

SPACE: 10(C) bitmaps are formed.

Case of 1RQ(A<=3) :-

TIME: Four bitmaps E^0 to E^3 need to be scanned and ORed.

SPACE:10(C) bitmaps are formed.

2. Our Multilevel Encoding:

Case of Equality queries(A=3):-

TIME: One bitmap E_{2-3} + Binary bitmap B needs to be scanned.

SPACE: 5(C/n) bitmaps where n is the bin size + 1Binary Bitmap is formed + mapping table

Case of One sided Range Query(A<=3):-

TIME: Two bitmaps $E_{0-1,}$ E_{2-3} need to be scanned and ORed.

SPACE:5(C/n) bitmaps where n is the bin size +1 Binary bitmap is formed + mapping table.

From the above we analyse that our proposed multi-level encoding technique is suitable for 1-sided range queries where the index space as well as number of bitmaps scanned is reduced by a factor of n. The above analyses is true for one sided range queries where query condition contains the upper bound value of the specific bin. However if the query condition contains the lower value of the specific bin additional $\log_2 n$ binary vectors need to be scanned. In such cases also the number of scans for one sided range queries is less than that required in Equality Encoding. For the given example the comparison between no. of scans for different query conditions of range queries is given in Figure 4. For range query A <= 6, Equality encoding requires eight bitmaps to be scanned whereas new multi-level encoding requires only four equality encoded bitmaps $E_{0-1,}$ $E_{2-3,}$ $E_{4-5,}$ E_{6-7} and Binary bitmap B to check for corresponding 0's which indicate value 6 so that value 7 is eliminated.

RID	$\Pi_A(R)$	E_{8-9}	E_{6-7}	E_{4-5}	E_{2-3}	E_{0-1}	B
1	3	0	0	0	1	0	1
2	2	0	0	0	1	0	0
3	1	0	0	0	0	1	1
4	2	0	0	0	1	0	0
5	8	1	0	0	0	0	0
6	2	0	0	0	1	0	0
7	9	1	0	0	0	0	1
8	0	0	0	0	0	1	0
9	7	0	1	0	0	0	1
10	5	0	0	1	0	0	1
11	6	0	1	0	0	0	0
12	4	0	0	1	0	0	0

Mapping Table

0 2 4 6 8	0
1 3 5 7 9	1

Fig 3. Multi-level encoding techniques

- **Analytical Model**

This section will compare the space-time trade off of the aforementioned bitmap encoding schemes**.** Let C denote the Cardinality and n denote the bin size.

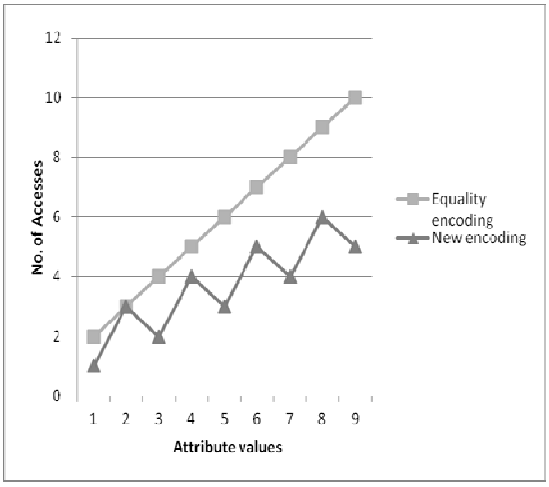


Fig 4. The comparison between no. of scans for different query conditions of range queries

Definition of cardinality in set theory refers to the number of members in the set. On database theory, the cardinality of a table refers to the number of rows contained in a particular table. In terms of OLAP system, cardinality refers to the number of rows in a table. On the other hand, on a data warehousing point of view, cardinality usually refers to the number of distinct values in a column. We compare the values of space and time at different values of C(10,100,1000). For an average case we have developed an analytical model for size and time comparisons of the two encoding schemes. Equations 1 and 2 are for index size and number of scans in Equality encoding and equations 3 and 4 are for size and number of scans in our Multilevel encoding scheme.

Size=C 1)

No. of scans=C/2 2)

Size=$C/n + \log_2 n + d$ 3)

No. of scans = $C/2n + \log_2 n + d$ 4)

The component d represents the size of mapping table. Since we assume that our mapping table will always reside in main memory we consider d to be zero. Figure 5 and Figure 6 show the graph between cardinality verses size for bin sizes 2 and 4 i.e for n=2 and n=4. Figure 7 and Figure 8 shows the graph between cardinality and no. of scans for bin sizes 2 and 4.

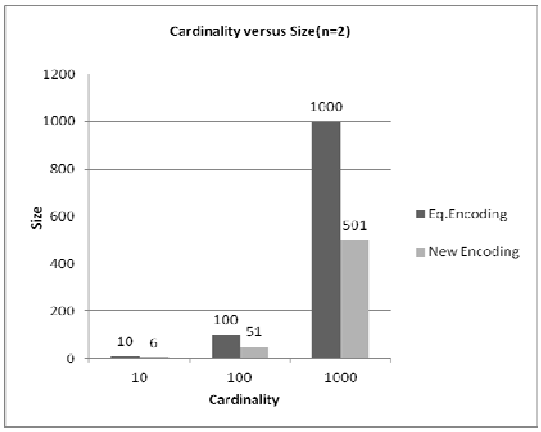


Fig 5. Cardinality vs Size

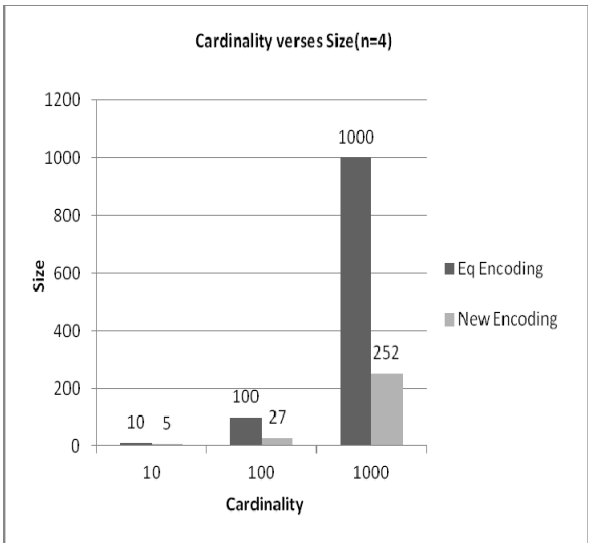


Fig 6. Cardinality vs Size

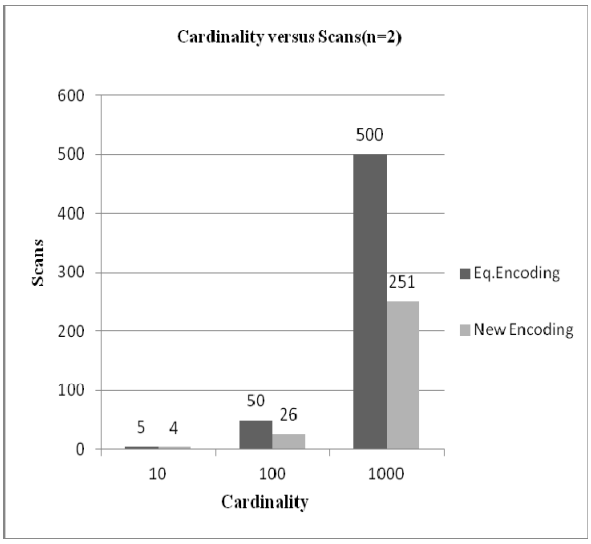


Fig. 7. Cardinality vs Scans

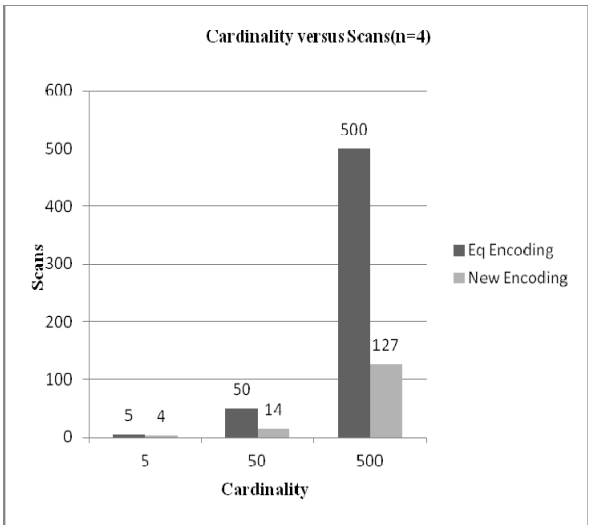
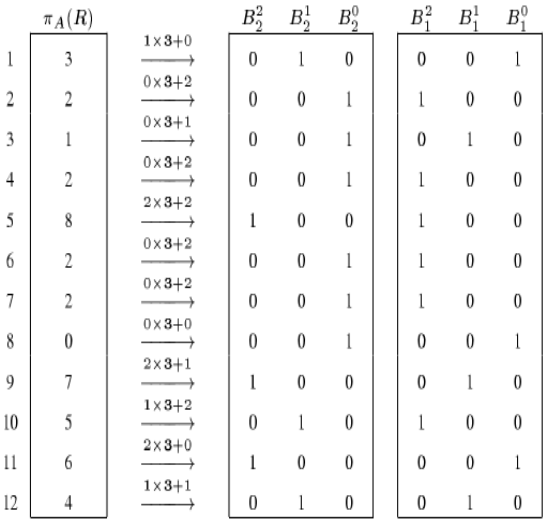


Fig 8. Cardinality vs Scans

5 Multi-component Encoding

The multi-component index are constructed from three basic encoding schemes by decomposing the attribute value into multiple components. The attribute value decomposition defines the arithmetic to represent the values of an attribute. It is the decomposition of an attribute's value in digits according to a chosen base. For example, consider an attribute with cardinality B = 50. An attribute value of 35 can be defined as a single base-50 digit (i.e., $35 = 35_{50}$), or as two base-8 digits (i.e., $35 = 4_8 3_8$), and so on. To make things clear we take the same 12-record relation used in Figure1.



	$\pi_A(R)$		B_2^2	B_2^1	B_2^0	B_1^2	B_1^1	B_1^0
1	3	1×3+0	0	1	0	0	0	1
2	2	0×3+2	0	0	1	1	0	0
3	1	0×3+1	0	0	1	0	1	0
4	2	0×3+2	0	0	1	1	0	0
5	8	2×3+2	1	0	0	1	0	0
6	2	0×3+2	0	0	1	1	0	0
7	2	0×3+2	0	0	1	1	0	0
8	0	0×3+0	0	0	1	0	0	1
9	7	2×3+1	1	0	0	0	1	0
10	5	1×3+2	0	1	0	1	0	0
11	6	2×3+0	1	0	0	0	0	1
12	4	1×3+1	0	1	0	0	1	0

Fig 9. A 2-Component index with base <3,3>

and transform it in a base <3,3> multi-component index .The attribute value 3 can be written in base-3 as $1_3 0_3$. By doing so the bitmaps have been reduced to 6. Figure 9 shows a 2-Component index with base <3,3>. The one-component encoding methods, such as the one used in the basic bitmap index (Figure 2a), requires the largest number of bitmaps. In contrast, the binary encoding produces the least number of bitmaps. This encoding method uses only $\log_2 B$ bitmaps for an attribute with cardinality B. However to answer a range query interval encoding requires accessing only two bitmaps whereas in binary encoding one has to access all the bitmaps. A number of authors have proposed strategies to find the balance between space and time requirements. One main purpose of studying multi-component encoding is to find whether any multi-component encoding can perform better than these two.

- ***Methodology***

Let attribute A have cardinality 1000, let its values range from 0 to 999. These values may be broken into three components of base size 10 each. Each of these components would be a digit of a 3 digit decimal number. Let i_1, i_2, and i_3 denote the values of three components ,the relation among them can be written as $i=i_1+10i_2+100i_3$. Such a three component index can be viewed as composed of three separate indexes on i_1, i_2, and i_3 .We propose BCD encoding for each of these three components. Figure 4 shows an example of our proposed multi-component technique. We observe that for B=0 to B=999 our encoding scheme requires 12 bitmaps. Let us look at the encoding of value 345 which is highlighted in the figure. The least significant four bitmap vectors (B0-B3) are BCD representation of least significant digit of the attribute value.

RID	$\Pi_A(R)$	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
1	136	0	0	0	1	0	0	1	1	0	1	1	0
2	345	0	0	1	1	0	1	0	0	0	1	0	1
3	789	0	1	1	1	1	0	0	0	1	0	0	1
4	069	0	0	0	0	0	1	1	0	1	0	0	1

Fig 10. Four bitmap vectors

The middle four vectors (B4-B7) are BCD representation of middle digit. The most significant four bitmap vectors BCD

representation of the most significant

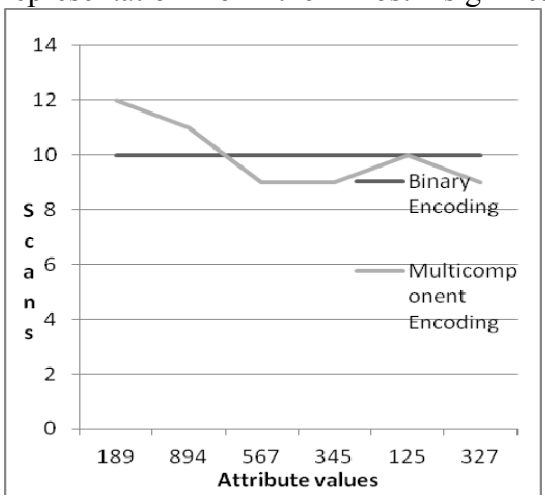


Fig 11. Binary Encoding vs. Multicomponent Encoding

digit of the attribute value. Based on the values of query condition the corresponding bitmap vectors are scanned based on their weights. The retrieval functions f_m for digits between 0 to 9 is given :

$F_0 = \mathbf{B_n}' \cdot \mathbf{B}_{n+1}' \cdot \mathbf{B}_{n+2}' \cdot \mathbf{B}_{n+3}'$;

$F_1 = \mathbf{B_n} \cdot \mathbf{B}_{n+1}' \cdot \mathbf{B}_{n+2}' \cdot \mathbf{B}_{n+3}'$;

$F_2 = \mathbf{B_n}' \cdot \mathbf{B}_{n+1} \cdot \mathbf{B}_{n+2}'$;

$F_3 = \mathbf{B_n}' \cdot \mathbf{B}_{n+1} \cdot \mathbf{B}_{n+2}$;

$F_4 = \mathbf{B_n}' \cdot \mathbf{B}_{n+1}' \cdot \mathbf{B}_{n+2}$;

$F_5 = \mathbf{B_n} \cdot \mathbf{B}_{n+1}' \cdot \mathbf{B}_{n+2}$;

$F_6 = \mathbf{B_n}' \cdot \mathbf{B}_{n+1} \cdot \mathbf{B}_{n+2}$;

$F_7 = \mathbf{B_n} \cdot \mathbf{B}_{n+1} \cdot \mathbf{B}_{n+2}$;

$F_8 = \mathbf{B_n}' \cdot \mathbf{B}_{n+1} \cdot \mathbf{B}_{n+2}' \cdot \mathbf{B}_{n+3}$;

$F_9 = \mathbf{B_n}' \cdot \mathbf{B}_{n+1}' \cdot \mathbf{B}_{n+2}' \cdot \mathbf{B}_{n+3}$;

Where the n can take values 0, 4 and 9.

Consider for instance the Query condition A=345. The multi-component encoding scheme B0, B1 and B2 are scanned for digit 5, B4 B5 and B6 are scanned for digit 4 and B8, B9 and B10 are scanned for digit 3. Thus we see that less bitmap vectors are scanned as compared to all bitmap scans in case of binary encoding.

1. Binary Encoding

Case of equality queries(A=345) :-

TIME:10(B) bitmaps needs to be scanned

SPACE: 10(B) bitmaps are formed.

Case of Range queries(A<=345):-

TIME: 10(B) bitmaps need to be scanned.

SPACE: 10 bitmaps

2. Multi-component encoding:

Case of Equality queries(A=345):-

TIME: 9 bitmap need to be scanned.

SPACE:12 bitmaps are formed.

Case of Range queries(A<=345):-

TIME:9 bitmaps need to be scanned.

SPACE:12 bitmaps.

From the above we analyze that our proposed multi-component indexing technique uses more space than the binary encoded index. However it accesses lesser bitmaps to answer the query Therefore it is possible that that the multi-component index may actually require less I/O time than a binary encoded index. For the given set of attribute values the comparison between no. of scans for different conditions of range queries is given in Figure 11.

A multi-component encoding is usually constructed with some user input parameters. For example if a user chooses the number of components, then it is possible to automatically decide the size of each component to minimize the number of bitmaps generated. For instance ,if a user specified to use a two component encoding for attribute cardinality of 100 ,then each component of size 10 is a good option . An alternative to fixing the number of components is fixing the base size of each component and use as many components as necessary to represent all the attribute values.

- ***Analytical model.***

Let B denote the cardinality of the attribute. For a given attribute value a_i the multi-component encoding decomposes i into a set of integers(i_1, i_2, i_3,…i_k). LetC_1, C_2…C_K denote the sizes of a k-component encoding basis sizes. Using BCD encoding, each component has 4 bitmaps. Thus The total number of bitmaps is D^E=4k .Based on the above assumptions we have developed an

analytical model for size and time comparison of two encoding schemes. Equations 5 and 6 represent index size and number of scans in binary encoding and equations 7 and 8 for size and number of scans in multi-component encoding scheme for range queries

Space= $\log_2 B$ 5)

No. of scans= $\log_2 B$ 6)

Space= 4k 7)

4k>=No. of scans>=k[($\sqrt[k]{B}$)/2 -2] 8)

Thus we conclude that a multi-component index with a base size greater than 2 uses more space than the binary encoded index, however, it may only accesses some of the bitmaps in order to answer a query. Figure 12 and Figure 13 shows the graph between cardinality verses size and the graph between cardinality and number of scans for the binary encoding and our proposed multi-component encoding. The dark shaded portion of bar representing multi-component encoding in Figure 12 tells us about the range over which it may vary compared to Binary encoding. For cardinality B=100,a binary encoding requires seven bitmaps to be scanned whereas our multi-component encoding may require between six(three bitmaps for each digit) and eight(four bitmaps for each digit). Number of scans as per equation 8) will be between 8(4k) and 6

(k[($\sqrt[k]{B}$)/2 -2]). Therefore, it is possible

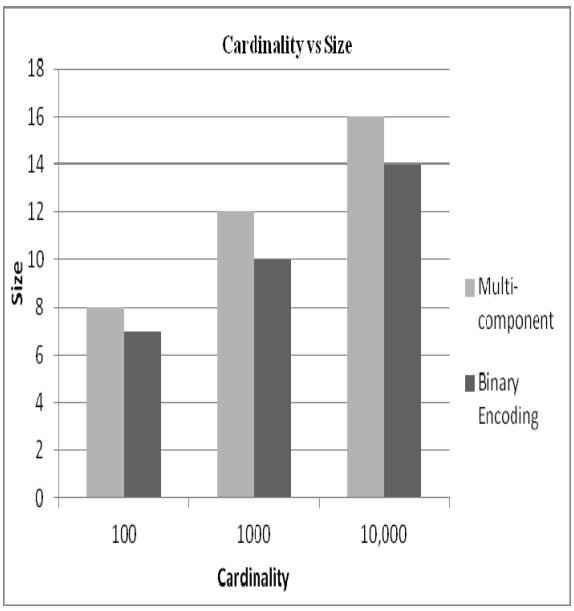


Fig 12. Cardinality vs. Size

that a multi-component index may actually require less I/O time than a Binary encoded index.

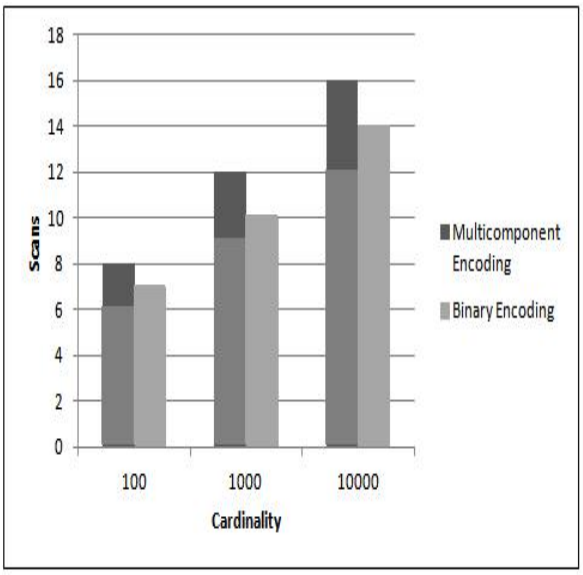


Fig 13. Cardinality vs. Size

Conclusion

The ability to extract data to answer complex, iterative, and ad hoc queries quickly is a critical issue for data warehouse applications. A good indexing technique is reduce I/O intensive table accesses against

large data warehouse tables. The challenge is to find an appropriate index type that would improve the queries' performance. Various Bitmap indexes have been demonstrated to significantly speed up searching operations in Data warehousing and On-Line Analytical Processing. All encoding methods proposed in the past can be categorized as either a Multi-component encoding or Multilevel-encoding. In this paper we have present novel variations in the class of Multi-level and Multi-component indexes and find that they answer range queries faster than some of the existing multi-level and multi-component indexes. We have formed an analytical model to predict the index size and access time of these encoding schemes for worst case scenario. The main contribution of this paper is the development of equations that predict the index size and number of scans which is a measure of I/O operations. Future work may include conducting a experimental evaluation of these two proposed encoding schemes on real application data.

References:

[1] Surajit Chaudhuri and Umeshwar Dayal. An Overview of Data Warehousing and OLAP Technology. In Proc. ACM SIGMOD Conf.1997, Pages 65-74.

[2] C.Y. Chan and Y.E. Ioannidis. An Efficient Bitmap Encoding Scheme for Selection Queries. Computer Sciences Department, University of Wisconsin-Madison,1998. http:/ /www.cs.wisc.edu/Ncychan/interval.ps

[3] Patrick O'Neil and Dallan Quass. Improved-Query-Performance with Variant Indexes. In Proc. ACM SIGMOD Conf. 1997, Pages 38-49.

[4] P. O'Neil and G. Graefe. Multi-Table Joins Through Bitmapped Join Indices. ACM SIGMOD Record, pages 8-11, September 1995.

[5] Morteza Z,Somnuk P.,Su-Cheng Haw. An adequate Design for Large Data Warehouse Systems: Bitmap index versus B-tree index. International Journal of Computers and Communications, Issue 2, Volume 2,2008.

[6] Stockinger, K and Wu, K. 2006. Bitmap Indicies for Data Warehouses. Idea Group, Inc., Chapter VII,179-202.LBNL-59952.

[7] J. Winchell. Rushmore's Bald Spot. *DBMS*, 4(10):58, September 1991.

[8] O'Neil, P. Model 204 Architecture and Performance. Workshop in High Performanc Transaction Systems, Asilomar, California, USA. Springer-Verlag.1987.

[9] Wu, K., & Otoo, E.J., & Shoshani, A. (2004). On the Performance of Bitmap Indices for High Cardinality Attributes. International Conference on VLDB,Toronto, Canada. Morgan Kaufmann.

[10] O'Neil.,E., O'Neil,P., and Wu.K., Bitmap index design choices and their performance impli-cations, Database Engineering and Applications Symposium. IDEAS 2007. 11th International, pp. 72-84.

[11] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. A performance comparison of bitmap indexes. In CIKM, pages 559–561. ACM, 2001.

[12] Kesheng Wu, Arie Shoshani, Kurt Stockinger. (2010) Analyses of Multi-Component Compressed Bitmap Indexes.ACM Transactions on Database Systems Vol . 35,No.1. Article2.

[13] Kesheng Wu, Kurt Stockinger and Arie Shoshani. Performance of Multi-Level and Multi – Component Compressed Bitmap Indexes. Lawrence Berkeley National Laboratory, June 11, 2007.

[14] Kesheng Wu, Ekow Otoo, and Arie Shoshani. Compressing bitmap indexes for faster search operations. In SSDBM'02, pages 99–108, Edinburgh, Scotland, 2002. LBNL-49627.

[15] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. Compressed bitmap indices for efficient query processing. Technical Report LBNL-47807, Lawrence Berkeley National Laboratory, Berkeley, CA, 2001.

[16] G. Antoshenkov. Byte-aligned bitmap compression. Technical report, Oracle Corp., 1994

[17] K.-L.Wu and P. Yu. Range-based bitmap indexing for high cardinality attributes with skew. Technical Report RC 20449, IBM Watson Research Division, Yorktown Heights, New York, May 1996

[18] Doron Rotem, Kurt Stockinger, Kesheng Wu. Optimizing I/O Costs of Multi-dimensional Queries using Bitmap Indices. DEXA 2005,220-229.

[19] Doron Rotem, Kurt Stockinger, and KeshengWu. Optimizing candidate check costs for bitmap indices.In CIKM 2005. ACM Press.

[20] Wu.K.,Otoo,E.,and hoshani, A.2006. Optimizing bitmap indices with efficient compression. ACM Trans. Datab. Syst 31,1-38