Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 1504 – 1522
© TÜBİTAK
doi:10.3906/elk-1807-277

HyBiX: A novel encoding bitmap index for space- and time-efficient query processing

Naphat KEAWPIBAL, Ladda PREECHAVEERAKUL, Sirirut VANICHAYOBON*
Information Systems Technology and Applied Research Laboratory (*i*Star), Department of Computer Science,
Faculty of Science, Prince of Songkla University, Songkhla, Thailand

Received: 26.07.2018 • **Accepted/Published Online:** 05.02.2019 • **Final Version:** 22.03.2019

Abstract: A bitmap-based index is an effective and efficient indexing method for answering selective queries in a read-only environment. It offers improved query execution time by applying low-cost Boolean operators on the index directly, before accessing raw data. A drawback of the bitmap index is that index size increases with the cardinality of indexed attributes, which additionally has an impact on a query execution time. This impact is related to an increase of query execution time due to the scanning of bitmap vectors to answer the queries. In this paper, we propose a new encoding bitmap index, called the HyBiX bitmap index. The HyBiX bitmap index was experimentally compared to existing encoding bitmap indexes in terms of space requirement, query execution time, and space and time trade-off for equality and range queries. As experimental results, the HyBiX bitmap index can reduce space requirements with high cardinality attributes with satisfactory execution times for both equality and range queries. The performance of the HyBiX bitmap index provides the second-best results for equality queries and the first-best for range queries in terms of space and time trade-off.

Key words: Data warehouse, bitmap index, encoding bitmap indexes, query processing, space and time trade-off

1. Introduction

The development of technologies has rapidly generated massive amounts of data from various sources [1–3], causing challenges in efficiently storing, searching, processing, analyzing, and managing such amounts of data. With increasing amounts of data, the efficiency of the query processing is a crucial issue when executing complex queries over large data by the traditional approaches [3, 4]. Several approaches have been demonstrated to speed up searching operations, such as parallel processing, materialized views, and indexing [4–9]. Among these three, indexing is an efficient approach to retrieve data without requiring additional hardware and to enable answering ad hoc and complex queries in reasonable times [7, 10–12]. One indexing technique for enabling fast query processing over large read-only repositories in data warehouses is bitmap-based indexes [5, 10, 11, 13, 14]. The common bitmap-based index is known as a basic bitmap index [15]. The basic bitmap index is easy to use for representing the data in binary format, with 0s and 1s, and allows fast query processing by low-cost Boolean operations (AND, OR, NOT, XOR) on the index directly before accessing the actual data. From this advantage, the basic bitmap index has been efficiently used in selection, aggregation, and iceberg queries, which exist in data warehouses and analytics applications [10, 16, 17].

The efficiency of the basic bitmap index is more suitable for attributes with low cardinality (the number

*Correspondence: sirirut.v@psu.ac.th

of unique values of that attribute) [14, 18]. Unfortunately, when the cardinality is increased, the index size used by the basic bitmap index significantly grows, causing problems with space requirements and query execution time. Therefore, there are four strategies to improve the performance of the basic bitmap index, including a compressing bitmap, binning bitmap, encoding bitmap, and multilevel and multicomponent bitmap [10, 19]. The compressing bitmap [20–27] offers a good storage requirement by compressing homogeneous bits (only bit 0s or only 1s) in each bitmap vector. Decompression and logical operations, however, dominate some execution time against the basic bitmap index without compression. The binning bitmap [10, 12, 28, 29] can accurately answer some range queries when they match the binning. For other queries, parts of original data have to be read from storage and checked against the specified conditions when the query does not match the binning. The encoding bitmap [30–33] minimizes the number of bitmap vectors created and allows us to use Boolean operations on the predicted bitmap vectors without decompression or additional processes. However, the encoding design is a critical issue, which has an impact on both space demanded and query execution times consumed. Lastly, the multilevel and multicomponent bitmap [19, 34] combines the preceding three strategies to reduce the storage requirement, but its complexity of query processing time is dramatically increased. Therefore, the design of an encoding scheme for the bitmap index is more likely to be comprehensively studied by improving the performance in terms of space and time trade-off (i.e. a trade-off between storage resources and efficiency).

The common encoding bitmap indexes have been implemented as a basic bitmap index [15], range bitmap index [30], interval bitmap index [31], encoded bitmap index [32], and dual bitmap index [33]. When dealing with high cardinality attributes, the basic, range, and interval bitmap indexes suffer from impractical storage requirements. In contrast, the limitation of the storage requirements for attributes having high cardinality can be efficiently managed by the encoded and dual bitmap indexes. For a processing of range query, it generally is in the form of $v_1 \leq A \leq v_2$, where v_1 and v_2 are values in the domain of attribute A. Obviously, the range query involves the continuous attribute values v_1 and v_2 between v_1 and v_2. For example, a domain of attribute A is $\{0, 1, 2, \ldots, 14\}$ and the range query can be in the form of $0 \leq A \leq 4$. The query involves the attribute values 0, 1, 2, 3, and 4. Especially for the encoded and dual bitmap indexes, they have suffered from massive scanning of relevant bitmap vectors, which has an impact on slow query processing with range queries. Clearly, the existing encoding bitmap indexes have not fully solved problems with range queries in terms of space and time trade-off. To achieve a good balance between space and time, a new encoding bitmap index, called HyBiX, short for hybrid encoding bitmap index, is introduced. The HyBiX bitmap index enables the small number of bitmap vectors created and requires a small number of bitmap vectors scanned for accurately answering equality and range queries.

In the remainder of this paper, we describe five encoding bitmap indexes in Section 2. The proposed encoding bitmap index, HyBiX, is presented in Section 3. In Section 4, a theoretical analysis in space requirement and execution time for six encoding bitmap indexes is presented. In Section 5, we present experimental results in which the space demanded and query processing time for six encoding bitmap indexes are compared. Finally, the major findings are summarized in Section 6.

2. Related works

A bitmap index is a well-known data structure for improving the speed of query processing in a data warehouse [4, 10, 14, 17]. The bitmap index contains a sequence of bits, one bitmap vector for a set of arbitrary attribute values, with each row representing an item. These bitmap vectors are primarily used in bitwise logical operations to answer queries [15, 17]. The bitwise logical operators AND, OR, and XOR are denoted by $\wedge$, $\vee$, and $\oplus$, respectively. Basically, an encoding bitmap index is an efficient strategy for representing the attribute values as well as maintaining query execution time. By a variety of representations to attribute values, different designs of encoding bitmap indexes can be obtained. This section presents the concept of five encoding bitmap indexes, including a basic bitmap index [15], range bitmap index [30], interval bitmap index [31], encoded bitmap index [32], and dual bitmap index [33], as described below.

2.1. Basic bitmap index

In the simplest scheme, each bitmap vector corresponds to one precise value of an indexed attribute, known as the basic bitmap index [15], applying an equality encoding. Let C be the attribute cardinality, which is the number of distinct values of that indexed attribute. Then the basic bitmap index consists of C bitmap vectors. To represent value v, the ith bit in bitmap vector for representing value v is set to 1 if the ith row of the indexed attribute contains value v. Otherwise, the bit is set to 0. The encoding function of the basic bitmap index can be written as in Eq. (1):

$$B^j = \begin{cases} 1 & j = v \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Assume that a domain of attribute A given by table T is $\{0, 1, 2, \ldots, 14\}$, as shown in Figure 1a. Then the basic bitmap index uses 15 bitmap vectors since the cardinality of the attribute A is 15, say $\{B^0$, B^1, B^2, $\ldots$, $B^{14}\}$, corresponding to the columns in Figure 1b. By using Eq. (1) to represent the attribute value 3, the 1st and 6th bits of B^3 are set to 1. The 1st and 6th bits of the remaining bitmap vectors are set to 0. When answering equality queries, only one bitmap vector associated with that query value is scanned. For example, to evaluate the equality query $A = 3$, only bitmap vector B^3 is scanned. The 1st and 6th rows are then returned as the final result for this query since the 1st and 6th bits in B^3 are set to 1, which represents the attribute value 3. For answering range queries, the bitmap vectors associated with each query value are scanned, and then bitwise-OR operators are used among those bitmap vectors to obtain the final result. For example, to evaluate the range query '$0 \le A \le 4$', five relevant bitmap vectors (i.e. bitmap vectors B^0, B^1, B^2, B^3, and B^4) are scanned and have bitwise-OR operators performed among them as $B^0 \vee B^1 \vee B^2 \vee B^3 \vee B^4$. Clearly, the basic bitmap index is recommended for efficiency of space usage with attributes having low cardinality as well as execution time for equality queries. In contrast, it has the expense of high space usage for attributes having

	A
1	3
2	9
3	14
4	8
5	10
6	3
7	4
8	0
9	12
10	5
.	.
.	.
.	.
10,000	2

(a) Table T

B^0	B^1	B^2	B^3	B^4	…	B^{13}	B^{14}
0	0	0	1	0	…	0	0
0	0	0	0	0	…	0	0
0	0	0	0	0	…	0	1
0	0	0	0	0	…	0	0
0	0	0	0	0	…	0	0
0	0	0	1	0	…	0	0
0	0	0	0	1	…	0	0
1	0	0	0	0	…	0	0
0	0	0	0	0	…	0	0
0	0	0	0	0	…	0	0
.	.	.	.	.	…	.	.
.	.	.	.	.	…	.	.
.	.	.	.	.	…	.	.
0	0	1	0	0	…	0	0

(b) Basic bitmap index

R^0	R^1	R^2	R^3	R^4	…	R^{12}	R^{13}
0	0	0	1	1	…	1	1
0	0	0	0	0	…	1	1
0	0	0	0	0	…	0	0
0	0	0	0	0	…	1	1
0	0	0	0	0	…	1	1
0	0	0	1	1	…	1	1
0	0	0	0	1	…	1	1
1	1	1	1	1	…	1	1
0	0	0	0	0	…	1	1
0	0	0	0	0	…	1	1
.	.	.	.	.	…	.	.
.	.	.	.	.	…	.	.
.	.	.	.	.	…	.	.
0	0	1	1	1	…	1	1

(c) Range bitmap index

I^0	I^1	I^2	I^3	I^4	I^5	I^6	I^7
1	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1
0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1
1	1	1	1	1	1	0	0
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
1	1	1	0	0	0	0	0

(d) Interval bitmap index

Figure 1. An example of the basic, range, and interval bitmap indexes: encoding of attribute A with cardinality 15.

high cardinality and also execution time for range queries, which is a drawback of the basic bitmap index. Table T is used as an example in what follows.

2.2. Range bitmap index

To solve the problem of the basic bitmap index with range queries, the range bitmap index [30] was introduced for optimizing one-side range queries. The range bitmap index uses $C-1$ bitmap vectors, say $\{R^0, R^1, \ldots, R^{C-2}\}$. Each bitmap vector R^j represents the values between 0 and j. The encoding function for this bitmap index, for attribute value v, is given in Eq. (2) as follows:

$$R^j = \begin{cases} 1 & v \le j \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Considering table T, the range bitmap index consists of 14 bitmap vectors as shown in Figure 1c. Using Eq. (2) to represent the attribute value 3, all bits from R^3 to R^{13} are set to 1, while the remaining bits are set to 0. Querying the range bitmap index uses the retrieval function in Eq. (3), for both equality and range queries. Obviously, an equality query is executed if $v_1 = v_2$ and a range query is executed if $v_1 < v_2$.

For $0 \le v_1 < v_2 \le C-1$,

$$\text{“}v_1 \le A \le v_2\text{”} = \begin{cases} R^0 & v_1 = v_2 = 0 \\ R^{v_1} \oplus R^{v_1-1} & 0 < v_1 = v_2 < C-1 \\ \overline{R^{C-2}} & v_1 = v_2 = C-1 \\ \overline{R^{v_1-1}} & 0 < v_1 < C-1, v_2 = C-1 \\ R^{v_2} & v_1 = 0, 0 \le v_2 < C-1 \\ R^{v_2} \oplus R^{v_1-1} & \text{otherwise.} \end{cases} \tag{3}$$

By using Eq. (3) to evaluate the equality query $A = 3$, for example, the bitmap vectors R^3 and R^2 are scanned and a bitwise-XOR operator is performed to answer the query, yielding $R^3 \oplus R^2$. For evaluating the range query ‘$0 \le A \le 4$’, only bitmap vector R^4 is scanned, and then the rows containing bit value 1 are returned because the bitmap vector R^4 represents the attribute values ranging from 0 to 4. In this case, the range bitmap index commonly scans only one bitmap vector. Although the range bitmap index is optimizing the equality and range queries by accessing 2 bitmap vectors at most, its space usage decreases by just one bitmap vector from the basic bitmap index. Massive storage usage is a crucial issue for the range bitmap index when built on attributes having high cardinality.

2.3. Interval bitmap index

Improving the range bitmap index, the interval bitmap index [31] reduces the number of bitmap vectors by half to $\{I^0, I^1, \ldots, I^{\lceil \frac{C}{2} \rceil - 1}\}$. Each bitmap vector I^j represents the range of values between j and $j+m$, where $m = \lfloor \frac{C}{2} \rfloor - 1$. The encoding function for the interval bitmap index can be written as in Eq. (4):

$$I^j = \begin{cases} 1 & j \le v \le j+m \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Figure 1d shows an example of the interval bitmap index for the data in table T, with the 8 bitmap vectors, $\{I^0, I^1, \ldots, I^7\}$. On encoding the attribute value 3, all bits between I^0 and I^3 are set to 1 and the remaining bits are set to 0. The retrieval functions in Eqs. (5) and (6) check for equality and range queries, respectively.

$$\text{“}A = v\text{”} = \begin{cases} I^0 & v = 0, m = 0 \\ \overline{I^0} & v = 1, C = 2 \\ I^1 & v = 1, C = 3 \\ I^v \wedge \overline{I^{v+1}} & v < m \\ I^1 \wedge I^0 & v = m, m > 0 \\ I^{v-m} \wedge \overline{I^{v-m-1}} & m < v < C-1, m > 0 \\ \overline{I^{\lceil \frac{C}{2} \rceil - 1} \vee I^0} & v = C-1. \end{cases} \tag{5}$$

$$\text{“}v_1 \leq A \leq v_2\text{”} = \begin{cases} I^{v_1} \wedge \overline{I^{v_2}+1} & v_2 < m \\ I^{v_1} \wedge I^0 & v_2 = m \\ I^{v_1} \wedge I^{v_2-m} & v_2 < v_1 + m, v_1 < n \\ I^{v_1} & v_2 = v_1 + m, v < n \\ I^{v_1} \vee I^{v_2-m} & v_2 > v_1 + m, v_1 < m \\ I^{v_1} \vee I^{v_1+1} & v_2 = v_1 + m + 1, v_1 = m \\ I^{v_2-m} \wedge \overline{I^{v_1-m-1}} & v_1 \geq n. \end{cases} \tag{6}$$

By using Eq. (5) to evaluate the equality query $A = 3$, for example, the retrieval function of the query is completely generated as $I^3 \wedge \overline{I^4}$. By using Eq. (6) to evaluate the range query $0 \leq A \leq 4$, it creates the retrieval function for this query, yielding $I^0 \wedge \overline{I^5}$. Obviously, the interval bitmap index scans 2 bitmap vectors at most for both equality and range queries. Therefore, the interval bitmap index is optimizing the execution time for range queries, but it gives slower execution time than the basic bitmap index when answering equality queries, which scans only one bitmap vectors. In terms of space usage, the interval bitmap index requires less space than the basic bitmap index and range bitmap index. However, by an increasing cardinality of the indexed attribute, it still suffers from the large storage occupied, which is decreased by just half compared with the basic bitmap index.

Furthermore, the concept of hybrid encoding schemes was introduced in [31]. The concept attempted to combine two known encoding bitmap indexes on a single attribute. It is a good idea to take advantage of the basic bitmap index, which is optimized for equality queries, and the range bitmap index or the interval bitmap index, which are optimized for range queries. This method allows us to answer both equality and range queries efficiently by separately creating 2 encoding bitmap indexes on a single attribute. These hybrid encoding schemes occasionally offer a slightly faster execution time at the expense of massive space requirements. Therefore, they rarely provide a better performance against existing nonhybrid encoding bitmap indexes.

2.4. Encoded bitmap index

To our knowledge, the encoded bitmap index [32] produces the smallest number of bitmap vectors, which consists of $\lceil \log_2 C \rceil$ bitmap vectors, say $\{E^0, E^1, \ldots, E^{\lceil \log_2 C \rceil - 1}\}$, and a mapping table. The mapping table stores the binary patterns of all distinct attribute values. Based on the data in table T (see Figure 2a), an example of the encoded bitmap index for the attribute with cardinality 15 is shown in Figure 2b, which consists of 4 bitmap vectors, E^0, E^1, E^2, and E^3. Let us consider encoding the attribute value 3. The binary pattern for the attribute value 3 in the mapping table is 0011. Therefore, the bitmap vectors are set for this item as $E^0 = 0$, $E^1 = 0$, $E^2 = 1$, and $E^3 = 1$, respectively.

To answer equality queries, the binary pattern of the query value is retrieved from the mapping table, and then the $\lceil \log_2 C \rceil$ bitmap vectors are jointly checked for this pattern in each row. Rows matching the target pattern across the $\lceil \log_2 C \rceil$ bits are returned as the answer to the query. To evaluate the equality query $A = 3$, for example, the binary of the attribute value 3 is retrieved from a mapping table as 0011 corresponding to the position of bitmap vector E^0, E^1, E^2, and E^3, respectively. Therefore, the retrieval function of this query is created as $\overline{E^0} \wedge \overline{E^1} \wedge E^2 \wedge E^3$. The complement of bitmap vector E^0 and E^1 is required before performing bitwise-AND operations with bitmap vector E^2 and E^3. Indeed, the encoded bitmap index definitely scans all bitmap vectors for answering equality queries. This has a critical impact on the increase of execution time for the encoded bitmap index.

For evaluating range queries, the retrieval function for the query is formed in Boolean expressions that apply bitwise-OR operators to get the final result. For example, to evaluate the range query $0 \leq A \leq 4$, we first look up the binary pattern of each item from a mapping table and then transform to Boolean expression. Then we get $\overline{E^0E^1E^2E^3}$ for the attribute value 0, $\overline{E^0E^1E^2}E^3$ for the attribute value 1, $\overline{E^0E^1}E^2\overline{E^3}$ for the attribute value 2, $\overline{E^0E^1}E^2E^3$ for the attribute value 3, and $\overline{E^0}E^1\overline{E^2E^3}$ for the attribute value 4. Onward, the bitwise-OR operators are used to get the final of the query, yielding $\overline{E^0E^1E^2E^3} \vee \overline{E^0E^1E^2}E^3 \vee \overline{E^0E^1}E^2\overline{E^3} \vee \overline{E^0E^1}E^2E^3 \vee \overline{E^0}E^1\overline{E^2E^3}$. To optimize the range query processing, the retrieval function can be further reduced. From our example, we can reduce the retrieval function to $\overline{E^0E^1} \vee \overline{E^0}E^1\overline{E^2E^3}$. Although the encoded bitmap index provides an effective usage of space requirement, its query processing still takes long execution times for both equality and range queries. Furthermore, the data mining techniques and parallel processing are applied to optimize execution time, especially equality and membership queries [35–38].

2.5. Dual bitmap index

The encoded bitmap index has an efficient storage requirement for attributes having high cardinality, but it still suffers from query execution time, especially equality queries due to accessing all bitmap vectors. To solve that obstacle, the dual bitmap index [33] gives an encoding design that is suitable for the equality queries. The dual bitmap index efficiently represents the attribute values by using two bitmap vectors. The dual bitmap index consists of $n = \lceil \sqrt{2C + 0.25} + 0.5 \rceil$ bitmap vectors, say $\{D^0, D^1, \ldots, D^{\lceil \sqrt{2C+0.25}+0.5 \rceil - 1}\}$. The dual encoding function is given in Eq. (7):

$$D^j = \begin{cases} 1 & j = r \text{ and } j = s \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where $hiC = \frac{n(n-1)}{2}$, $r = \left\lceil \sqrt{2(hiC - v) + 0.25} + 0.5 \right\rceil$, and $s = \left\lceil r - 1 - \left[\left(v - \frac{(n-r)(n-r-1)}{2} \right) \bmod r \right] \right\rceil$.

Figure 2c depicts an example of the dual bitmap index for an attribute with cardinality 15, with 6 bitmap vectors, say $\{D^0, D^1, D^2, D^3, D^4, D^5\}$. Using Eq. (7) to represent the attribute value 3, the bits in D^1 and D^5 are set to 1, while the remaining bits are set to 0.

Evaluation of equality queries with the dual bitmap index uses the retrieval function in Eq. (8) directly. For example, to evaluate the equality query $A = 3$, by using Eq. (8), the retrieval function of this query can

	A
1	3
2	9
3	14
4	8
5	10
6	3
7	4
8	0
9	12
10	5
.	.
.	.
.	.
10,000	2

(a) Table T

E 0	E 1	E 2	E 3
0	0	1	1
1	0	0	1
1	1	1	0
1	0	0	0
1	0	1	0
0	0	1	1
0	1	0	0
0	0	0	0
1	1	0	0
0	1	0	1
.	.	.	.
.	.	.	.
.	.	.	.
0	0	1	0

Mapping table	
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110

(b) Encoded bitmap index and a mapping table

D 0	D 1	D 2	D 3	D 4	D 5
0	1	0	0	0	1
0	0	1	1	0	0
1	1	0	0	0	0
1	0	0	0	1	0
0	1	0	1	0	0
0	1	0	0	0	1
1	0	0	0	0	1
0	0	0	0	1	1
0	1	1	0	0	0
0	0	0	1	1	0
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
0	0	1	0	0	1

(c) Dual bitmap index

Figure 2. An example of the encoded and dual bitmap indexes: encoding of attribute A with cardinality 15.

be $D^5 \wedge D^1$:

$$\text{“}A = v\text{”} = D^r \wedge D^s. \tag{8}$$

To answer range queries, the retrieval function can be dynamically created by using Eq. (8) for each query value, and then the bitwise-OR operators are used. For example, to evaluate the range query $0 \leq A \leq 4$, the retrieval function for each item is dynamically created by using Eq. (8). Then we get $D^5 \wedge D^4$ for the attribute value 0, $D^5 \wedge D^3$ for the attribute value 1, $D^5 \wedge D^2$ for the attribute value 2, $D^5 \wedge D^1$ for the attribute value 3, and $D^5 \wedge D^0$ for the attribute value 4. Subsequently, the bitwise-OR operators are applied to find the final result of this query, yielding $(D^5 \wedge D^4) \vee (D^5 \wedge D^3) \vee (D^5 \wedge D^2) \vee (D^5 \wedge D^1) \vee (D^5 \wedge D^0)$. Definitely, the retrieval function generated by the dual bitmap index provides more complexity with the increasing number of query values. To improve the range queries, the generated retrieval function can probably be minimized to reduce the scanning of bitmap vectors as well as the numbers of Boolean operations, which has an impact on the query execution time [39]. In fact, the dual bitmap index was introduced to deal with attributes having high cardinality and to improve equality query processing. Therefore, the performance of the dual bitmap index is degraded when answering range queries.

Figure 3 illustrates the five encoding schemes with cardinality 15. Note that the black dots denote bit value 1. The basic bitmap index generates the largest amount of bitmap vectors, which consists of 15 bitmap vectors, as depicted in Figure 3a. Each bitmap vector corresponds to only one precise attribute value. The range bitmap index uses 14 bitmap vectors while the interval bitmap index uses 8 bitmap vectors, as shown in Figures 3b and 3c, respectively. The encoded and dual bitmap indexes use 4 and 6 bitmap vectors, as shown in Figures 3d and 3e, respectively.

In summary, we would like to highlight some issues of the existing encoding bitmap indexes in the literature. The basic, range, and interval bitmap indexes produce a massive number of bitmap vectors for attributes with high cardinality. Meanwhile, the encoded and dual bitmap indexes consume some proportional storage requirements, but their execution time with range queries is still unsatisfactory. Therefore, the existing encoding bitmap indexes are insufficient regarding the overall performance with equality and range queries, considering both space requirements and query execution times.

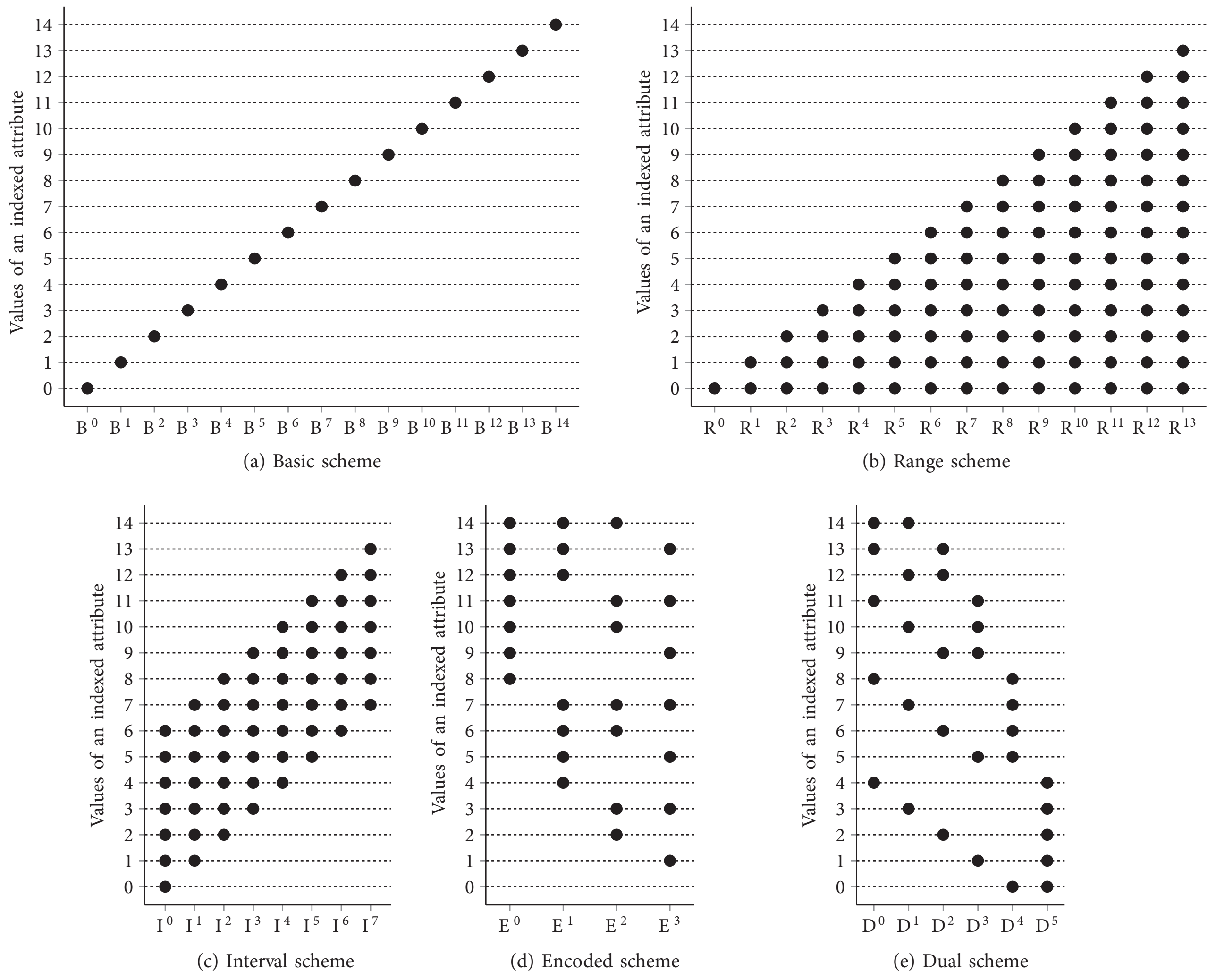


Figure 3. Five encoding schemes with cardinality 15 (• represents bit 1).

3. HyBiX: Hybrid encoding bitmap index

The HyBiX bitmap index [40] is a new encoding scheme that takes the strengths of the design concepts in existing encoding bitmap indexes (i.e. range and dual bitmap indexes) and the grouping of indexed attribute values to reduce the storage requirements as well as to improve query execution times with high cardinality attributes. It is important to note that our HyBiX bitmap index is different from the concept of hybrid encoding introduced in [31]. The basic concept of the HyBiX bitmap index is to divide the distinct attribute values into several groups, where the number of values contained in each group is unequal. Therefore, the number of values in each group decreases by 1 to efficiently represent attribute values as much as possible. In consequence, the number of groups and the numbers of values in each group can be figured out, as described below.

Let n denote a possible number of groups and a total number of bitmap vectors. Regarding the amounts of values in each group, group i has $n-i$ values, where $0 \leq i \leq n-1$. Therefore, the first group of the HyBiX bitmap index (group 0) has n values, the next group (group 1) has $n-1$ values, and so on until the last group (group $n-1$) has 1 value. Then the total number of values from every group must be at least the cardinality of the indexed attribute. Consequently, the number of bitmap vectors used by the HyBiX bitmap index can be calculated by $n = \lceil \sqrt{2C + 0.25} - 0.5 \rceil$, where C is the cardinality of the indexed attribute (see the proof in [40]).

3.1. Bitmap index creation for HyBiX bitmap index

Let H denote a set of bitmap vectors for the HyBiX bitmap index and $H = \{H^0, H^1, \ldots, H^{n-1}\}$. Table 1 describes the notation used in the bitmap index creation algorithm for the HyBiX bitmap index. Algorithm 1 shows the creation algorithm of the HyBiX bitmap index in 6 steps, described below.

Table 1. Notation used in the creation algorithm for the HyBiX bitmap index.

Symbol	Description
n	The total number of bitmap vectors used (i.e. the total number of groups)
C	The number of distinct values of the indexed attribute (i.e. cardinality)
C_{max}	The maximum value of C that can be represented by n bitmap vectors, $C_{max} = \frac{n(n+1)}{2}$
H^i	The bitmap vectors created for HyBiX bitmap index, where $0 \leq i \leq n-1$
v	A distinct value of indexed attribute
g_v	A group containing the value v, $g_v = n - \lceil \sqrt{2(C_{max} - v) + 0.25} - 0.5 \rceil$
s_v	Starting value in the group that contains the value v, $s_v = (\frac{g_v}{2})(2n - g_v + 1)$
e_v	Ending value in the group that contains the value v, $e_v = \left[(\frac{g_v+1}{2})(2n - g_v)\right] - 1$
l_v	The sequence of value v inside the group (level), $l_v = g_v + (v - s_v)$

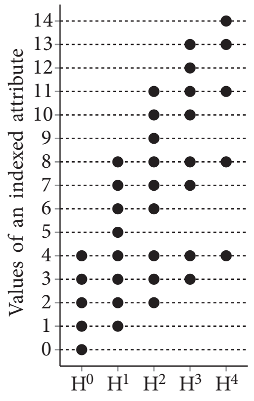
Algorithm 1 The creation of the HyBiX bitmap index.

```
Input: The cardinality and the values of the indexed attribute
Output: The HyBiX bitmap index
1: Assign an increasing sequence of numbers to each of distinct values of the indexed attribute (i.e.
   $0, 1, 2, \ldots, C-1$) and calculate $n = \lceil \sqrt{2C + 0.25} - 0.5 \rceil$
2: Create the HyBiX assistant table
3: for a value $v$ in each row do
4:     Get the group ($g_v$) and the level ($l_v$) corresponding to the value $v$ from the HyBiX assistant table
5:     Set bitmap vectors according to the following equation: $H^i = \begin{cases} 1 & g_v \leq i \leq l_v \\ 0 & \text{otherwise.} \end{cases}$
6: end for
```

First of all, each value of the indexed attribute A is assigned by a number in an increasing sequence (i.e. $0, 1, 2, \ldots, C-1$) and the number of bitmap vector used (n) is calculated. Next, a HyBiX assistant table is created once and stored in the main memory to provide preliminary information on each distinct attribute value. The benefits of the HyBiX assistant table are to eliminate a redundant computation, to facilitate creating the HyBiX bitmap index, and to assist with the eventual queries. The HyBiX assistant table contains five elements: v, g_v, s_v, e_v, and l_v, characterizing the attribute values. The values of attribute A are then encoded in the 3rd–6th steps. For each attribute value v, the associated values of g_v and l_v are retrieved from the HyBiX assistant table in the 4th step. Then, in the 5th step, the bits in H^i are set to 1 if i is between g_v and l_v; otherwise, the bits remain set to 0. Therefore, the encoding function of the HyBiX bitmap index is written as in Eq. (9):

$$H^i = \begin{cases} 1 & g_v \le i \le l_v \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Figure 4a depicts the HyBiX encoding scheme for attribute A with cardinality 15. To create the HyBiX bitmap index for attribute A with cardinality 15 (given in Figure 4b), the HyBiX bitmap index has 5 bitmap vectors, say $\{H^0, H^1, H^2, H^3, H^4\}$, with a HyBiX assistant table, as shown in Figure 4c. For example, on encoding the attribute value 3, the values of g_3 and l_3 are 0 and 3, respectively. Then the bits in H^0 to H^3 are set to 1, while the bit in H^4 is set to 0, and these are highlighted in Figure 4c.



(a) HyBiX scheme

	A
1	3
2	9
3	14
4	8
5	10
6	3
7	4
8	0
9	12
10	5
.	.
.	.
.	.
10,000	2

(b) Table T

H^0	H^1	H^2	H^3	H^4
1	1	1	1	0
0	0	1	0	0
0	0	0	0	1
0	1	1	1	1
0	0	1	1	0
1	1	1	1	0
1	1	1	1	1
1	0	0	0	0
0	0	0	1	0
0	1	0	0	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
1	1	1	0	0

v	g_v	s_v	e_v	l_v
0	0	0	4	0
1	0	0	4	1
2	0	0	4	2
3	0	0	4	3
4	0	0	4	4
5	1	5	8	1
6	1	5	8	2
7	1	5	8	3
8	1	5	8	4
9	2	9	11	2
10	2	9	11	3
11	2	9	11	4
12	3	12	13	3
13	3	12	13	4
14	4	14	14	4

(c) HyBiX bitmap index and HyBiX assistant table

Figure 4. An example of the HyBiX bitmap index for attribute A with cardinality 15.

3.2. Query processing for HyBiX bitmap index

A query of the form $v_1 \le A \le v_2$ is able to contribute both classes of equality and range queries. The query is an equality query if $v_1 = v_2$ and it is a range query if $0 \le v_1 < v_2 \le C-1$. The query processing algorithm for the HyBiX bitmap index is described in Algorithm 2. The equality query processing is executed in the 1st–3rd steps, while the range query processing is executed in the 4th–19th steps.

3.2.1. Equality query processing for HyBiX bitmap index

Normally, the form of an equality query is $A = v$, where $v = v_1 = v_2$. Therefore, the bitmap vector related to the group containing the value v_1 (or the value v_2) and the bitmap vector related to the level of the value v_1 (or the value v_2) are identified. The retrieval function for an equality query is created by Eq. (10):

$$\text{“}A = v\text{”} = P \wedge Q, \tag{10}$$

where

$$P = \begin{cases} H^{g_{v_1}} & g_{v_1} = 0 \\ \overline{H^{g_{v_1}-1}} \wedge H^{g_{v_1}} & \text{otherwise.} \end{cases},$$

Algorithm 2 The query processing of HyBiX bitmap index.

```
Input: Query values: $v_1$ and $v_2$
Output: A retrieval function
 1: if $v_1 = v_2$ then                                         ▷ Answer equality query
 2:     Get information of $v_1$ from HyBiX assistant table
 3:     return $P \wedge Q$
 4: else                                                        ▷ Answer range query
 5:     Get information of $v_1$ and $v_2$ from HyBiX assistant table
 6:     if $g_{v_1} = g_{v_2}$ then                              ▷ Cover one group
 7:         if $v_1 = s_{v_1}$ and $v_2 = e_{v_2}$ then
 8:             return $P$
 9:         else
10:             return $P \wedge Q$
11:         end if
12:     else                                                    ▷ Cover more than one group
13:         if $g_{v_1} = 0$ then
14:             return $T \vee RV1$
15:         else
16:             return $T \vee (\overline{H^{g_{v_1}-1}} \wedge RV1)$
17:         end if
18:     end if
19: end if
```

$$Q = \begin{cases} \overline{H^{l_{v_2}+1}} & v_1 = s_{v_1}, v_2 \neq e_{v_2} \\ H^{l_{v_1}} & v_1 \neq s_{v_1}, v_2 = e_{v_2} \\ H^{l_{v_1}} \oplus H^{l_{v_2}+1} & v_1 \neq s_{v_1}, v_2 \neq e_{v_2} \\ 1 & \text{otherwise.} \end{cases}$$

Example 1 *To evaluate the equality query $A = 3$, the information of 3 consists of $g_3 = 0$, $s_3 = 0$, $e_3 = 4$, and $l_3 = 3$, given by the HyBiX assistant table. Using Eq. (10), the retrieval function of this query is $H^0 \wedge (H^3 \oplus H^4)$.*

3.2.2. Range query processing for HyBiX bitmap index

For the HyBiX bitmap index, the range query can be divided into two cases: either $g_{v_1} = g_{v_2}$ (i.e. the values v_1 and v_2 are placed in the same group) or $g_{v_1} < g_{v_2}$ (i.e. the values v_1 and v_2 are placed in a different group). In the first case (the 6th–11th steps), the bitmap vectors related to the group containing the value v_1 are identified, and then the bitmap vectors related to the level of the value v_1 (and the value v_2, if necessary) are identified. The retrieval function for this case is given in Eq. (11) by utilizing the function P and Q of an equality query processing.

For $g_{v_1} = g_{v_2}$,

$$\text{“}v_1 \leq A \leq v_2\text{”} = \begin{cases} P & v_1 = s_{v_1}, v_2 = e_{v_2} \\ P \wedge Q & \text{otherwise.} \end{cases} \tag{11}$$

Example 2 *To evaluate the range query in the form of $0 \leq A \leq 4$, the information of 0 and 4 is provided by the HyBiX assistant table. By using Eq. (11), the retrieval function of this query is H^0. Therefore, the HyBiX bitmap index accesses only one bitmap vector to answer this query.*

In the second case (the 12th–18th steps), the query values cover many groups of the HyBiX bitmap index if $g_{v_1} < g_{v_2}$. Ideally, there are three parts separately considered. The bitmap vectors related to the group containing the values v_1 and v_2 are identified, respectively. Next, the bitmap vectors related to the groups containing relevant query values, except the groups containing the values v_1 and v_2, are identified. Then the bitwise-OR operations are used among those three parts to find the final result. The retrieval function is given in Eq. (12).

For $g_{v_1} < g_{v_2}$,

$$``v_1 \le A \le v_2" = T \vee \begin{cases} RV1 & g_{v_1} = 0 \\ \overline{H^{g_{v_1}-1}} \wedge RV1 & \text{otherwise,} \end{cases} \tag{12}$$

where

$$T = \overline{H^{g_{v_1}}} \wedge \begin{cases} \bigvee_{i=g_{v_1}+1}^{g_{v_2}-1} H^i \vee RV2 & g_{v_2} - g_{v_1} \ge 2 \\ RV2 & \text{Otherwise.} \end{cases},$$

$$RV1 = \begin{cases} H^{g_{v_1}} & v_1 = s_{v_1} \\ H^{g_{v_1}} \wedge H^{l_{v_1}} & v_1 \neq s_{v_1}. \end{cases}, \quad RV2 = \begin{cases} H^{g_{v_2}} & v_2 = e_{v_2} \\ H^{g_{v_2}} \wedge \overline{H^{l_{v_2}+1}} & v_2 \neq e_{v_2}. \end{cases}$$

Example 3 *To evaluate the range query in the form of* $6 \le A \le 11$*, the information of 6 and 11 is provided by the HyBiX assistant table. By using Eq.* (12)*, the retrieval function of this query is* $\left(\overline{H^1} \wedge H^2\right) \vee \left(\overline{H^0} \wedge H^1 \wedge H^2\right)$.

4. Theoretical analysis

Table 2 shows a theoretical analysis of six encoding bitmap indexes, comparing space requirements and the numbers of bitmap vectors to be scanned for the equality and the range queries. The encoded bitmap index offers the best space requirement while the HyBiX bitmap index provides the second-best space requirement, which is better than the dual, interval, range, and basic bitmap indexes, respectively.

In order to answer the equality queries, the basic bitmap index has the fastest execution time because of the scan of 1 bitmap vector. The scans of the range, interval, and dual bitmap indexes are 2 bitmap vectors at most, followed by the HyBiX bitmap index, which ranges from 2 to 4 bitmap vectors. The encoded bitmap index is the slowest execution time due to the scans of $\lceil \log_2 C \rceil$ bitmap vectors. For answering the range queries, the range and interval bitmap indexes have satisfactory execution times due to the scans of 2 bitmap vectors at most. The HyBiX bitmap index provides the third fastest execution time with the range queries because it scans the relevant bitmap vectors as needed, corresponding to the groups of the query values rather than the individual query value. The numbers of bitmap vectors scanned used by the encoded and dual bitmap indexes are increasing with the ranges of query values. Additionally, the basic bitmap index has the slowest execution time with the range queries since many related bitmap vectors are accessed.

Table 2. A comparative study of six encoding bitmap index algorithms.

Bitmap index	Space requirement	Query time for equality queries	Query time for range queries
	#bitmap vectors created	#bitmap vectors scanned	#bitmap vectors scanned
Basic	C	1	$v_2 - v_1 + 1$
Range	$C - 1$	1 to 2	1 to 2
Interval	$\lceil \frac{C}{2} \rceil$	2	2
Encoded	$\lceil \log_2 C \rceil$	$\lceil \log_2 C \rceil$	1 to $(v_2 - v_1 + 1)(\lceil \log_2 C \rceil)$
Dual	$\lceil \sqrt{2C + 0.25} + 0.5 \rceil$	2	3 to $2(v_2 - v_1 + 1)$
HyBiX	$\lceil \sqrt{2C + 0.25} - 0.5 \rceil$	2 to 4	1 to $g_{v_2} - g_{v_1} + 4$

C is cardinality of the indexed attribute.
g_{v_1} is the group in HyBiX bitmap index containing value v_1.
g_{v_2} is the group in HyBiX bitmap index containing value v_2.
$0 \leq g_{v_1} \leq g_{v_2} \leq \lceil \sqrt{2C + 0.25} - 0.5 \rceil - 1$.

5. Experimental results

5.1. Dataset used and experimental setting

The experiments were conducted on 64-bit Windows 10 and 3.20 GHz Intel Core i5-4570 with 4.00 GB main memory. We used the TPC(H) benchmark dataset.[1] The benchmark data are generated along with a scale factor (SF), which indicates the size of data. We experimented with four different scale factors, including scale factors 25, 50, 75, and 100. The data along these scale factors are approximately 18 GB, 37 GB, 56 GB, and 75 GB in size, which store over 150, 300, 450, and 600 million rows, respectively. In our experiments, we used the table LINEITEM, consisting of 16 attributes as follows: ORDERKEY, PARTKEY, SUPPKEY, LINENUMBER, QUANTITY, EXTENDEDPRICE, DISCOUNT, TAX, RETURNFLAG, LINESTATUS, SHIPDATE, COMMITDATE, RECEIPTDATE, SHIPINSTRUCT, SHIPMODE, and COMMENT.

As a bitmap index is thoroughly suitable for attributes having fixed cardinality, we then discarded attributes having flexible cardinalities (ORDERKEY, PARTKEY, SUPPKEY, EXTENDEDPRICE, and COMMENT). Furthermore, it is a difficult comparison of query execution time for range queries with low cardinality. The ranges of query values are too small, hardly providing significant insights of a query execution time. Among attributes having fixed cardinality, we then neglected some attributes having low cardinality (the values in parentheses indicate cardinalities of the corresponding attribute), consisting of LINENUMBER (7), DISCOUNT (11), TAX (9), RETURNFLAG (3), LINESTATUS (2), SHIPINSTRUCT (4), and SHIPMODE (7). In our study, we intentionally emphasized the attributes having fixed and high cardinality, which can certainly have an impact on the index size and query execution time. Therefore, we considered four attributes having high cardinality (i.e. QUANTITY (50), SHIPDATE (2526), COMMITDATE (2466), and RECEIPTDATE (2,555)). Since the cardinalities of SHIPDATE, COMMITDATE, and RECEIPTDATE attributes are slightly different, we presented the result of SHIPDATE attribute as the results of COMMITDATE and RECEIPTDATE attributes. Consequently, we selected 2 representative attributes: a small dataset (QUANTITY attribute with 50 cardinalities) and a large dataset (SHIPDATE attribute with 2526 cardinalities).

The space efficiency of an encoding bitmap index is measured in terms of space requirements for storing

[1]http://www.tpc.org/tpch/

all its bitmap vectors. The time efficiency of an encoding bitmap index is measured in terms of average query execution time over all 15 queries in each query set for equality and range queries. The query execution time includes a disk I/O time for reading relevant bitmap vectors as well as a CPU time for Boolean operations on them. For some bitmap indexes, the Boolean simplification is used with range queries, but it does not significantly impact the overall query execution time. The trade-off between space and time of an encoding bitmap index is measured in terms of the overall performance calculated by the rectangular area under the coordinates of space demanded and time consumed.

5.2. The space efficiency of six encoding bitmap indexes

Figure 5 affirms the above theoretical analysis of space requirements for the six encoding bitmap indexes for the two selected attributes. The basic bitmap index requires the largest space while the encoded bitmap index requires the smallest for both QUANTITY and SHIPDATE attributes. The HyBiX bitmap index requires less space than the other remaining bitmap indexes for both attributes. As the cardinality increases, the sizes of basic, range, and interval bitmap indexes exceed the size of the raw data. This is an undesirable characteristic of indexing; the index size should be smaller than the raw data size to gain a benefit from the encoding.

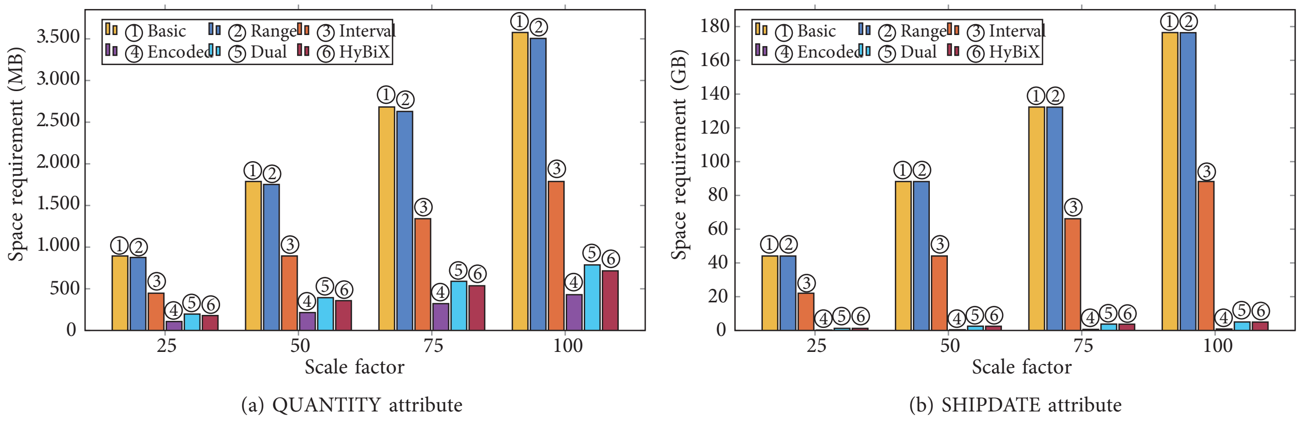
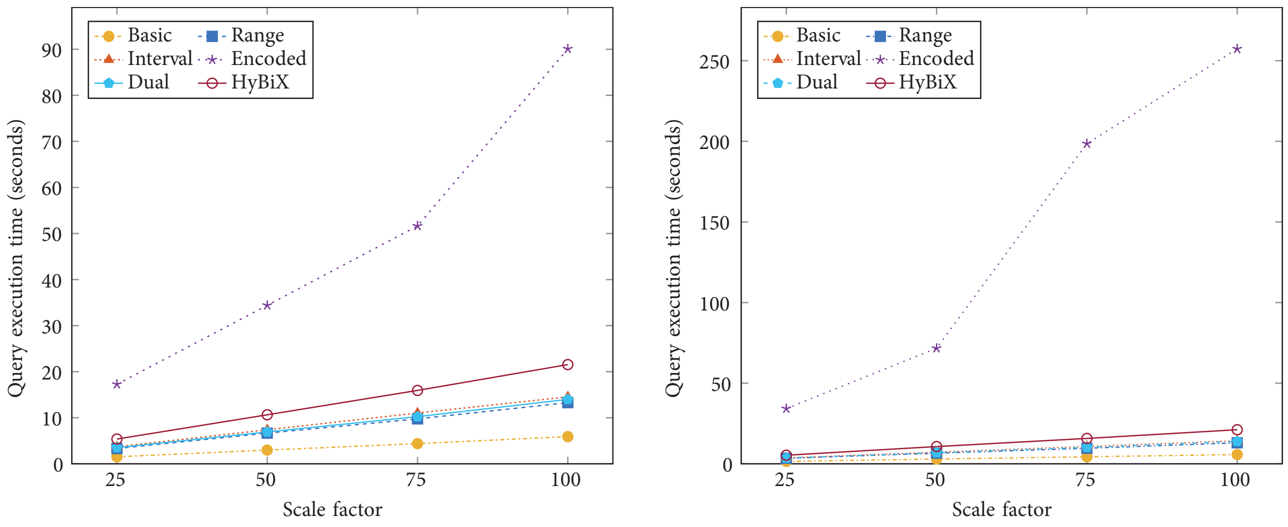


Figure 5. The space requirement of six encoding bitmap indexes.

5.3. The time efficiency of six encoding bitmap indexes

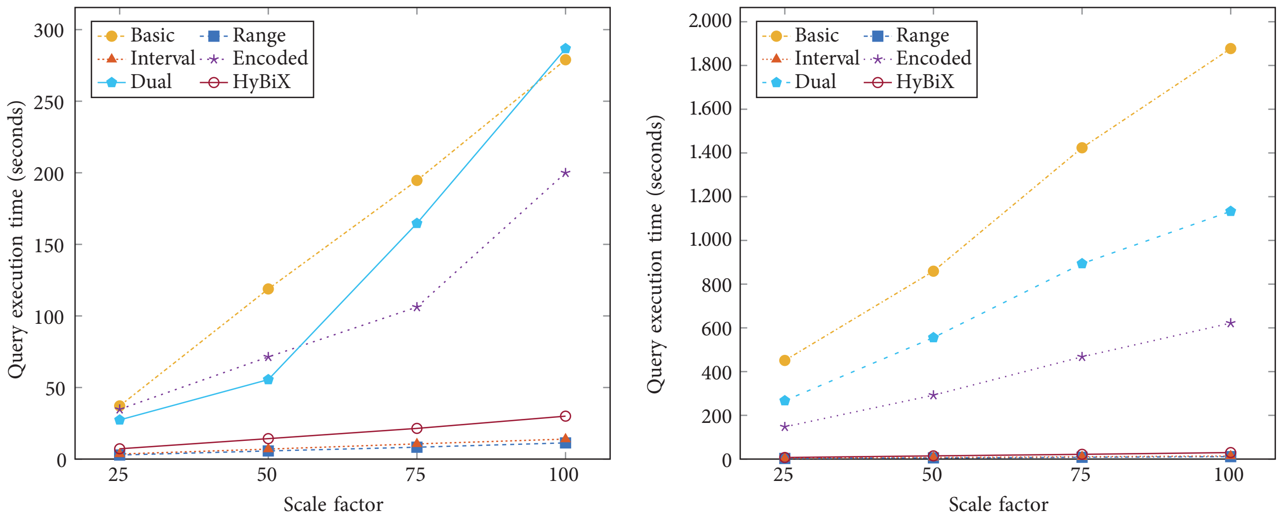
Figure 6 illustrates the query execution times of the six encoding bitmap indexes with equality queries on QUANTITY and SHIPDATE attributes. The basic bitmap index offered the fastest execution time, while the range, interval, and dual bitmap indexes had slower execution time than the basic bitmap index. The execution time used by the HyBiX bitmap index was slightly different from the range, interval, and dual bitmap indexes, but much faster than that used by the encoded bitmap index. Furthermore, the execution time used by the encoded bitmap index significantly increases with the cardinality of the indexed attribute.

Figure 7 shows a comparison of the query execution times with six encoding bitmap indexes for the range queries on two selected attributes. The range and interval bitmap indexes offered the fastest execution time, while the HyBiX bitmap index obtained the third fastest execution time. Clearly, the execution times with the encoded and dual bitmap indexes are undesirable because of the high complexity of their retrieval functions. The basic bitmap index has the slowest execution time. With increased cardinality, the ranges of query values are broadened. The execution times of the basic, encoded, and dual bitmap indexes increased exponentially.



(a) QUANTITY attribute (b) SHIPDATE attribute

Figure 6. The query execution time of six encoding bitmap indexes for equality queries.



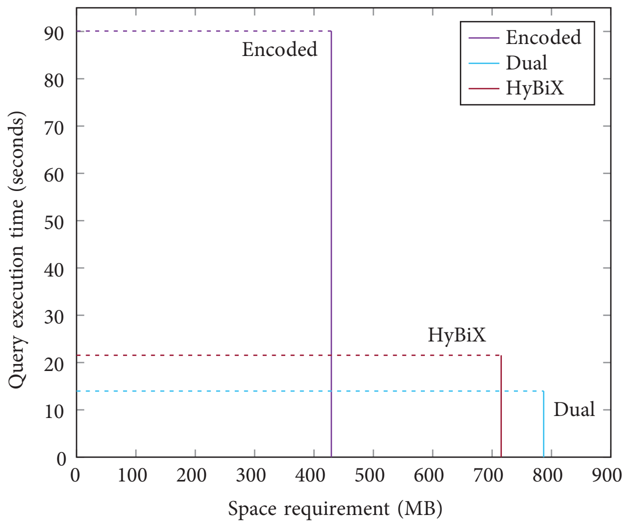
(a) QUANTITY attribute (b) SHIPDATE attribute

Figure 7. The query execution time of six encoding bitmap indexes for range queries.

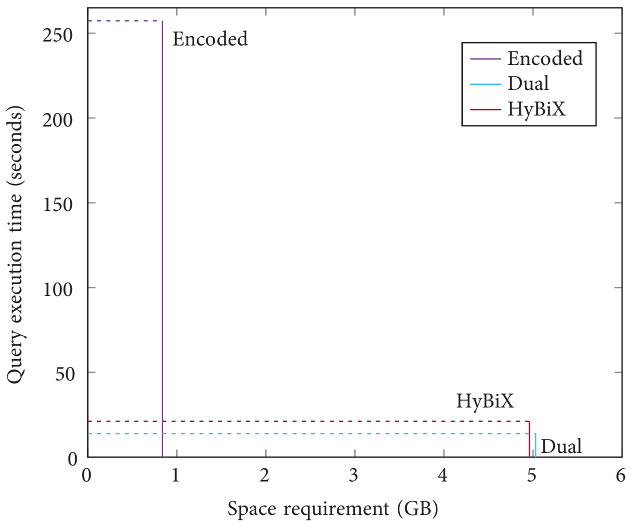
5.4. The trade-off between space and time of four encoding bitmap indexes

The space requirements used by basic, range, and interval bitmap indexes are undesirable, which certainly has an impact on the overall performance in terms of space and time trade-off. We present the results only for scale factor 100 as the result of any scale factors. Figure 8 depicts space and time trade-off for three encoding bitmap indexes for the equality queries on QUANTITY and SHIPDATE attributes. The dual bitmap index achieves the best performance in terms of space and time trade-off for the equality queries. However, the performance of the HyBiX bitmap index is better than that of the encoded bitmap index for both attributes.

Figures 9 shows space and time trade-off with three encoding bitmap indexes for the range queries on two selected attributes. The result confirms that the HyBiX bitmap index outperforms the other encoding bitmap indexes for the range queries regarding space and time trade-off with both attributes.

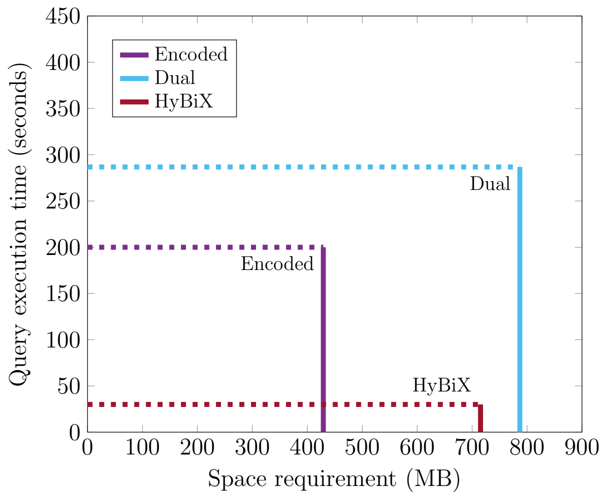


(a) QUANTITY attribute



(b) SHIPDATE attribute

Figure 8. Space and time trade-off of three encoding bitmap indexes for the equality queries in scale factor 100.



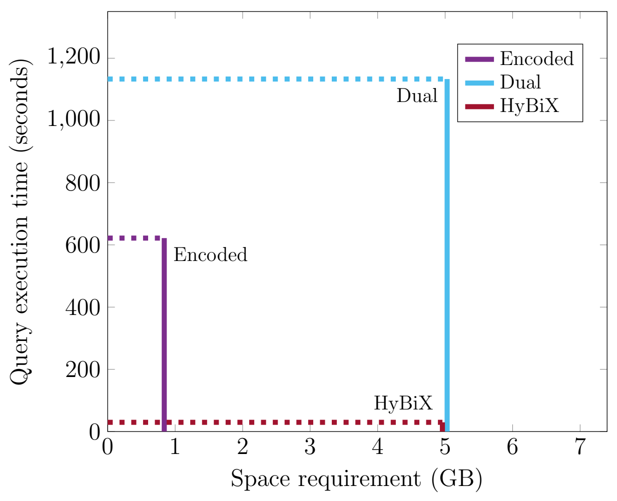


Figure 9. Space and time trade-off of three encoding bitmap indexes for the range queries in scale factor 100.

Obviously, the HyBiX bitmap index has the advantage in space requirements of the dual bitmap index and the advantage in query processing of the range bitmap index. Therefore, this provides the improved efficiency of the HyBiX bitmap index in the space and time trade-off point of views, especially for range queries.

6. Conclusions

In data warehouses, analytics applications require access to a massive amount of data, which will be used in further data analysis. Therefore, the retrieval techniques play an important role and provide convenience to the query processing with such amounts of data. Bitmap-based indexes have been successfully deployed in selection, aggregation, and iceberg queries. Several encoding bitmap indexes have been introduced to reduce index size and efficiently speed up the query processing. When dealing with the range queries on the indexed attributes having high cardinality, the performance of existing encoding bitmap indexes is degraded, both in terms of space required and time to process queries, and these are critical issues with encoding bitmap indexes. In this paper, we presented a new encoding bitmap index, called the HyBiX bitmap index. Our comparative study shows that the HyBiX bitmap index can reduce the space requirements with high cardinality of indexed attributes. Although the performance of the HyBiX bitmap index is worse than that of the dual bitmap index in terms of space and time trade-off for the equality queries, the performance of the HyBiX bitmap index outperforms the other existing encoding bitmap indexes in terms of space and time trade-off with the range queries. In future work, the performance of the HyBiX bitmap index can be improved by applying data mining techniques to select frequent query values submitted.

Acknowledgments

This research was financially supported by a PSU-PhD scholarship from Prince of Songkla University. The authors would like to thank Associate Professor Seppo Karrila, PhD (Chemical Engineering), from the Publication Clinic, Research and Development Office, Prince of Songkla University, for help in manuscript proofreading.

References

[1] Sagiroglu S, Sinanc D. Big data: a review. In: 2013 International Conference on Collaboration Technologies and Systems; San Diego, CA, USA; 2013. pp. 42-47.

[2] Kambatla K, Kollias G, Kumar V, Grama A. Trends in big data analytics. Journal of Parallel and Distributed Computing 2014; 74 (7): 2561-2573. doi: 10.1016/j.jpdc.2014.01.003

[3] Thanekar SA, Subrahmanyam K, Bagwan AB. Big data and mapreduce challenges, opportunities and trends. International Journal of Electrical and Computer Engineering 2016; 6 (6): 2911-2919.

[4] Gani A, Siddiqa A, Shamshirband S, Hanum F. A survey on indexing techniques for big data: taxonomy and performance evaluation. Knowledge and Information Systems 2016; 46 (2): 241-284.

[5] Chan CY, Ioannidis YE. Bitmap index design and evaluation. ACM SIGMOD Record 1998; 27 (2): 355-366.

[6] Sohrabi MK, Azgomi H. TSGV: A table-like structure-based greedy method for materialized view selection in data warehouses. Turkish Journal of Electrical Engineering & Computer Sciences 2017; 25 (4): 3175-3187. doi: 10.3906/elk-1608-112

[7] Silberschatz A, Korth HF, Sudarshan S. Database System Concepts. New York, NY, USA: McGraw-Hill, 2011.

[8] Stallings W. Computer Organization and Architecture Designing for Performance. Upper Saddle River, NJ, USA: Prentice Hall Press, 2012.

[9] Lu J, Feng J. A survey of MapReduce based parallel processing technologies. China Communications 2014; 11 (14): 146-155. doi: 10.1109/CC.2014.7085615

[10] Stockinger K, Wu K. Bitmap indices for data warehouses. In: Wrembel R, Koncilia C (editors). Data Warehouses and OLAP Concepts Architectures and Solutions. Hershey, PA, USA: IGI Global, 2007. pp. 157-178.

[11] Chen Z, Wen Y, Cao J, Zheng W, Chang J et al. A survey of bitmap index compression algorithms for big data. Tsinghua Science and Technology 2015; 20 (1): 100-115. doi: 10.1109/TST.2015.7040519

[12] Qin X. Performance comparison of index schemes for range query of big data. In: 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery; Changsha, China; 2016. pp. 1469-1473.

[13] Chaudhuri S, Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Record 1997; 26 (1): 65-74. doi: 10.1145/248603.248616

[14] Mei Y, Ji K, Wang F. A survey on bitmap index technologies for large-scale data retrieval. In: 2013 6th International Conference on Intelligent Networks and Intelligent Systems; Shenyang, China; 2013. pp. 316-319.

[15] O'Neil P, Quass D. Improved query performance with variant indexes. ACM SIGMOD Record 1997; 26 (2): 38-49. doi: 10.1145/253262.253268

[16] Guzun G, Canahuate G. Performance evaluation of word-aligned compression methods for bitmap indices. Knowledge and Information Systems 2016; 48: (2) 277-304. doi: 10.1007/s10115-015-0877-9

[17] Prakash KS, Prathap PMJ. Evaluating aggregate functions of iceberg query using priority based bitmap indexing strategy. International Journal of Electrical and Computer Engineering 2017; 7 (6): 3745-3752.

[18] Abdulhadi ZQ, Zuping Z, Housien HI. Bitmap index as effective indexing for low cardinality column in data warehouse. International Journal of Computer Applications 2013; 68 (24): 38-42.

[19] Wu K, Shoshani A, Stockinger K. Analyses of multi-level and multi-component compressed bitmap indexes. ACM Transactions on Database Systems 2010; 35 (1): 1-52. doi: 10.1145/1670243.1670245

[20] Antoshenkov G. Byte-aligned bitmap compression. In: DCC '95 Data Compression Conference; Snowbird, UT, USA; 1995. p. 476.

[21] Wu K, Otoo EJ, Shoshani A. Optimizing bitmap indices with efficient compression. ACM Transactions on Database Systems 2006; 31 (1): 1-38. doi: 10.1145/1132863.1132864

[22] Stabno M, Wrembel R. RLH: Bitmap compression technique based on run-length and Huffman encoding. Information Systems 2009; 34 (4): 400-414. doi: 10.1016/j.is.2008.11.002

[23] Wen Y, Chen Z, Ma G, Cao J, Zheng W et al. SECOMPAX: A bitmap index compression algorithm. In: 2014 23rd International Conference on Computer Communications and Networks; Shanghai, China; 2014. pp. 1-7.

[24] Chang J, Chen Z, Zheng W, Cao J, Wen Y et al. SPLWAH: A bitmap index compression scheme for searching in archival Internet traffic. In: 2015 IEEE International Conference on Communications; London, UK; 2015. pp. 7089-7094.

[25] Kim S, Lee J, Satti SR, Moon B. SBH: Super byte-aligned hybrid bitmap compression. Information Systems 2016; 62: 155-168. doi: 10.1016/j.is.2016.07.004

[26] Wu Y, Chen Z, Wen Y, Zheng W, Cao J. COMBAT: A new bitmap index coding algorithm for big data. Tsinghua Science and Technology 2016; 21 (2): 136-145. doi: 10.1109/TST.2016.7442497

[27] Li C, Chen Z, Zheng W, Wu Y, Cao J. BAH: A bitmap index compression algorithm for fast data retrieval. In: 2016 IEEE 41st Conference on Local Computer Networks; Dubai, United Arab Emirates; 2016. pp. 697-705.

[28] Goyal N, Sharma Y. New binning strategy for bitmap indices on high cardinality attributes. In: 2nd Bangalore Annual COMPUTE Conference; Bangalore, India; 2009. pp. 1-5.

[29] Kim JW. Binning strategy for hierarchical bitmap indices with large scale domain hierarchy. International Journal of Applied Engineering Research 2016; 11 (18): 9289-9296.

[30] Wu KL, Yu PS. Range-based bitmap indexing for high cardinality attributes with skew. In: Twenty-Second Annual International Computer Software and Applications Conference; Vienna, Austria; 1998. pp. 61-66.

[31] Chan CY, Ioannidis YE. An efficient bitmap encoding scheme for selection queries. ACM SIGMOD Record 1999; 28 (2): 215-226. doi: 10.1145/304181.304201

[32] Wu MC, Buchmann AP. Encoded bitmap indexing for data warehouses. In: 14th International Conference on Data Engineering; Orlando, FL, USA; 1998. pp. 220-230.

[33] Wattanakitrungroj N, Vanichayobon S. Dual bitmap index: Space-time efficient bitmap index for equality and membership queries. In: 2006 International Symposium on Communications and Information Technologies; Bangkok, Thailand; 2006. pp. 568-573.

[34] Bhan M, Rajanikanth K, and Kumar STV. Multi-level and multi-component bitmap encoding for efficient search operations. Database Systems Journal 2012; 3 (4): 47-60.

[35] Alam MGR, Arafat MY, Iftekhar MKU. A new approach of dynamic Encoded bitmap indexing technique based on query history. In: 2008 International Conference on Electrical and Computer Engineering; Dhaka, Bangladesh; 2008. pp. 974-979.

[36] Sainui J, Vanichayobon S, Wattanakitrungroj N. Optimizing encoded bitmap index using frequent itemsets mining. In: 2008 International Conference on Computer and Electrical Engineering; Phuket, Thailand; 2008. pp. 511-515.

[37] Keawpibal A, Wattanakitrungroj N, Vanichayobon S. Enhanced encoded bitmap index for equality query. In: 2012 8th International Conference on Computing Technology and Information Management; Seoul, South Korea; 2012. pp. 293-298.

[38] Keawpibal N, Duangsuwan J, Wettayaprasit W, Preechaveerakul L, Vanichayobon S. DistEQ: Distributed equality query processing on Encoded bitmap index. In: 2015 12th International Joint Conference on Computer Science and Software Engineering; Songkhla, Thailand; 2015. pp. 309-314.

[39] Keawpibal N, Preechaveerakul L, Vanichayobon S. Optimizing range query processing for Dual bitmap index. Walailak Journal of Science and Technology 2019; 16 (2): 133-142.

[40] Keawpibal N. HyBiX: Hybrid encoding bitmap index for efficient space and query processing time. PhD, Prince of Songkla University, Songkhla, Thailand, 2018.