HyBiX: Hybrid Encoding Bitmap Index for Efficient Space and Query Processing Time

Naphat Keawpibal

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Science (International Program)
Prince of Songkla University
2018

HyBiX: Hybrid Encoding Bitmap Index for Efficient Space and Query Processing Time

Naphat Keawpibal

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Science (International Program)
Prince of Songkla University
2018

Thesis Title	HyBiX: Hybrid Encoding Bitmap Index for Efficient Space and Query Processing Time
Author	Mr. Naphat Keawpibal
Major Program	Computer Science (International Program)

Major Advisor

..... (Asst. Prof. Dr. Sirirut Vanichayobon)

Co-advisor

..... (Asst. Prof. Dr. Ladda Preechaveerakul)

Examining Committee:

..... Chairperson (Assoc. Prof. Dr. Ohm Sornil)

..... Committee (Asst. Prof. Dr. Sirirut Vanichayobon)

..... Committee (Asst. Prof. Dr. Ladda Preechaveerakul)

..... Committee (Asst. Prof. Dr. Wiphada Wettayaprasit)

The Graduate School, Prince of Songkla University, has approved this thesis as partial fulfillment of the requirements for the Doctor of Philosophy Degree in Computer Science (International Program).

.....
(Prof. Dr. Damrongsak Faroongsarng)
Dean of Graduate School

This is to certify that the work here submitted is the result of the candidate's own investigations. Due acknowledgement has been made of any assistance received.

.....Signature
(Asst. Prof. Dr. Sirirut Vanichayobon)
Major Advisor

.....Signature
(Asst. Prof. Dr. Ladda Preechaveerakul)
Co-advisor

.....Signature
(Mr. Naphat Keawpibal)
Candidate

I hereby certify that this work has not been accepted in substance for any degree, and is not being currently submitted in candidature for any degree.

.....Signature
(Mr. Naphat Keawpibal)
Candidate

ชื่อวิทยานิพนธ์ HyBiX: การลงรหัสดัชนีบิตแมปแบบไฮบริด สำหรับประสิทธิภาพด้านพื้นที่และเวลาการประมวลผลสอบถาม
ผู้เขียน นายณภัทร แก้วภิบาล
สาขาวิชา วิทยาการคอมพิวเตอร์ (นานาชาติ)
ปีการศึกษา 2561

บทคัดย่อ

การพัฒนาของเทคโนโลยีในปัจจุบันได้สร้างข้อมูลจำนวนมหาศาลขึ้นมา ซึ่งได้ก่อให้เกิดปัญหาในการจัดเก็บและเข้าถึงข้อมูลจำนวนมหาศาลดังกล่าว ดังนั้นเทคนิคในการจัดเก็บข้อมูลและการเข้าถึงข้อมูลจึงได้รับความสนใจและศึกษาเพื่อให้มีการจัดเก็บและเข้าถึงข้อมูลอย่างมีประสิทธภาพ ดัชนีบิตแมปเป็นวิธีการจัดทำดัชนีที่มีประสิทธิภาพและประสิทธิผลในการเรียกดูข้อมูลบนระบบที่มีสภาวะแวดล้อมแบบอ่านอย่างเดียว เนื่องจากสามารถดำเนินการการค้นหาได้รวดเร็วโดยใช้ตัวดำเนินการบูลีนต้นทุนต่ำบนดัชนีได้โดยตรงก่อนเข้าถึงข้อมูลจริง อย่างไรก็ตามข้อเสียของดัชนีบิตแมปคือขนาดของดัชนีที่มีขนาดใหญ่ขึ้นเมื่อสร้างบนแอตทริบิวต์ที่มีคาร์ดินอลิตี้สูง วิทยานิพนธ์นี้เสนอดัชนีบิตแมปที่มีการลงรหัสรูปแบบใหม่ซึ่งเรียกว่า ดัชนีบิตแมปแบบไฮบริด (ดัชนีบิตแมป HyBiX) แนวคิดพื้นฐานของการสร้างดัชนีบิตแมปแบบไฮบริดคือการจัดกลุ่มค่าของแอตทริบิวต์ และการใช้แนวคิดพื้นฐานการลงรหัสของดัชนีบิตแมปรูปแบบอื่น ๆ ที่มีอยู่ เพื่อปรับปรุงประสิทธิภาพทั้งด้านเนื้อที่และเวลาที่ใช้ในการประมวลผลสำหรับการสืบค้นข้อมูลในลักษณะต่าง ๆ การจัดกลุ่มค่าข้อมูลของแอตทริบิวต์ช่วยอำนวยความสะดวกในการตอบแบบสอบถามที่มีการค้นหาช่วงของค่าข้อมูลที่ต่อเนื่องกัน จากผลการวิเคราะห์และทดลองเปรียบเทียบระหว่างดัชนีบิตแมปแบบไฮบริดกับดัชนีบิตแมปอื่น ๆ แสดงให้เห็นว่า เวลาที่ใช้ในการตอบแบบสอบถามแบบค่าเท่ากันเร็วขึ้น 79% และการตอบแบบสอบถามแบบช่วงเร็วขึ้น 82% นอกจากนี้ประสิทธิภาพของดัชนีบิตแมปแบบไฮบริดในแง่ของการแลกเปลี่ยนระหว่างประสิทธิภาพของพื้นที่กับเวลา (Space vs. time trade-off) อยู่ในลำดับที่สามที่ดีที่สุดสำหรับการสอบถามแบบค่าเท่ากัน และลำดับแรกที่ดีที่สุดสำหรับการสอบถามแบบช่วง เมื่อเปรียบเทียบกับดัชนีบิตแมปแบบอื่น ๆ

Thesis Title	HyBiX: Hybrid Encoding Bitmap Index for Efficient Space and Query Processing Time
Author	Mr. Naphat Keawpibal
Major Program	Computer Science (International Program)
Academic Year	2018

ABSTRACT

With an increasing availability of technology, an enormous amount of data has been generated. The problems in the storage and access have emerged. The consequent need for efficient techniques to store and access the information has been a strong resurgence of interest in the area of information retrieval. A bitmap-based index is an effective and efficient indexing method for operating information retrieval 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. However, a drawback of the bitmap index is that the index size increases with the cardinality of indexed attributes. This dissertation then proposes a new encoding bitmap index, called HyBiX bitmap index. The basic concept of HyBiX bitmap index is the use of grouping idea with attribute values and the encoding design of existing encoding bitmap indexes in order to improve both storage demanded and execution time consumed with various queries. Particularly, the grouping of attribute values facilitates in answering a continuous range of query values. The experiment show that the HyBiX bitmap index takes 79% and 82% faster execution times than the Encoded bitmap index, for equality and range queries, respectively. Furthermore, the performance of HyBiX bitmap index in terms of space and time trade-off achieves the third-best and first-best as compare to existing encoding bitmap index, for equality and range queries, respectively.

ACKNOWLEDGEMENTS

First of all, I would like to express my deep gratitude to my advisor, Assistant Prof. Dr. Sirirut Vanichayobon, for her support and guidance in the discussion on different opinions, for her patient and generously spent hours in correcting both my stylistic and scientific mistakes, and for a completed final research. She has helped me become better in conducting scientific research and scientific writing.

I would like to express my sincere thanks to my co-advisor, Assistant Prof. Dr. Ladda Preechaveerakul. The door to Assistant Prof. Dr. Ladda Preechaveerakul office always opens whenever I ran into a trouble spot or had a question about my study.

Besides my advisor, I would like to thank the rest of my dissertation committee: Associate Prof. Dr. Ohm Sornil and Assistant Prof. Dr. Wiphada Wettayaprasit, for their insightful comments and encouragement to accomplish my study.

My thanks go to PSU Ph.D scholarship financially supported by the Graduate School, Prince of Songkla University (PSU). I also gratefully acknowledge the PSU.GS. Financial Support for Thesis, Fiscal Year 2017 from Graduate School, PSU.

I also thank all lecturers and officers at the Department of Computer Science, Faculty of Science, PSU, who gave access to the Ph.D. office and research facilities. Without their precious supports, it would not be possible to conduct this research.

I would like to thank my friends studying in Master and Ph.D. degree at Department of Computer Science, Faculty of Science, PSU, who helped and encouraged me to overcome difficulties during my study.

Last but not least, my deepest thanks are going to my parents who always encourage me, believe in me, and give me the willpower to keep walking, and to my brother who always cheers me up and stands by me.

Naphat Keawpibal

TABLE OF CONTENTS

Page

ABSTRACT (Thai) **V**

ABSTRACT (English) **VI**

ACKNOWLEDGEMENTS **VII**

TABLE OF CONTENTS **VIII**

LIST OF TABLES **X**

LIST OF FIGURES **XI**

LIST OF ALGORITHMS **XIII**

CHAPTER

1 Introduction **1**

1.1 Background and rationale 1

1.2 Research objectives 4

1.3 Research contributions 4

1.4 Dissertation organization 5

2 Bitmap Index Strategies **6**

2.1 Basic bitmap index 6

2.2 Compression 8

2.3 Binning 16

2.4 Encoding 16

2.5 Multi-level & Multi-component 17

3 Existing Encoding Bitmap Indexes **19**

3.1 Range bitmap index 19

3.2 Interval bitmap index 21

3.3 Encoded bitmap index 23

TABLE OF CONTENTS (Continued)

Page

3.4 Scatter bitmap index 25
3.5 Dual bitmap index 28

4 HyBiX: Hybrid Encoding Bitmap Index 35
4.1 The design concept of HyBiX bitmap index 35
4.2 Bitmap index creation for HyBiX bitmap index 42
4.3 Query processing for HyBiX bitmap index 44

5 Performance Study 52
5.1 Theoretical analysis 52
5.2 Experimental results 53
5.2.1 Data set used and experimental setting 53
5.2.2 The space-efficiency of seven encoding bitmap indexes . . . 55
5.2.3 The time-efficiency of seven encoding bitmap indexes 56
5.2.4 The trade-off between space and time of four encoding bitmap indexes 60

6 Conclusion and Future Work 65
6.1 Summary 65
6.2 Future work 67

BIBLIOGRAPHY 68

VITAE 76

LIST OF TABLES

Table **Page**

2.1 A summarization of compression algorithms to the Basic bitmap index . . . 10

3.1 A summarization of the number of bitmap vectors used for six encoding bitmap indexes 30

3.2 A summarization of encoding bitmap index algorithms 32

4.1 Notation used in creation algorithm for HyBiX bitmap index 42

5.1 A comparative study of seven encoding bitmap index algorithms 54

LIST OF FIGURES

Figure **Page**

2.1 An example of the Basic bitmap index: encoding of attribute A with cardinality 15. 7
2.2 Bitmap index strategies. 8

3.1 An example of the Range bitmap index: encoding of attribute A with $C = 15$. 20
3.2 An example of the Interval bitmap index: encoding of attribute A with $C = 15$. 21
3.3 An example of the Encoded bitmap index: encoding of attribute A with $C = 15$. 24
3.4 An example of the Scatter bitmap index: encoding of attribute A with $C = 15$. 26
3.5 An example of the Dual bitmap index: encoding of attribute A with $C = 15$. 28
3.6 Six encoding schemes with $C = 15$, (● represents bit 1). 30

4.1 A basic concept of HyBiX bitmap index with cardinality 15. 38
4.2 HyBiX scheme for attribute A with cardinality 15. 38
4.3 HyBiX assistant table, and an example of the HyBiX bitmap index for attribute A with $C = 15$. 44
4.4 The result of the equality query $A = 3$. 45
4.5 The result of the equality query $A = 8$. 46
4.6 The result of the range query $1 \leq A \leq 4$. 47
4.7 The result of the range query $6 \leq A \leq 8$. 48
4.8 The result of the range query $3 \leq A \leq 13$. 49
4.9 The result of the range query $6 \leq A \leq 10$. 50

5.1 The space requirement of seven alternative encoding bitmap indexes. 55
5.2 The query execution time of seven encoding bitmap indexes for equality queries: A query value falls in any group of HyBiX bitmap index. 56
5.3 The query execution time of seven encoding bitmap indexes for equality queries: Query values fall in group 0 of HyBiX bitmap index. 57
5.4 The query execution time of seven encoding bitmap indexes for range queries: A query value falls in any groups of HyBiX bitmap index. 58

LIST OF FIGURES (Continued)

Figure **Page**

5.5 The query execution time of seven encoding bitmap indexes for range queries: Query values fall in group 0 of HyBiX bitmap index. 59

5.6 The space vs. time trade-off of four encoding bitmap indexes for equality queries in scale factor 100: A query value falls in any group of HyBiX bitmap index. 60

5.7 The space vs. time trade-off of four encoding bitmap indexes for equality queries in scale factor 100: A query value falls in group 0 of HyBiX bitmap index. 61

5.8 The space vs. time trade-off of four encoding bitmap indexes for range queries in scale factor 100: Query values fall in any groups of HyBiX bitmap index. 62

5.9 The space vs. time trade-off of four encoding bitmap indexes for range queries in scale factor 100: Query values fall in group 0 of HyBiX bitmap index. 63

6.1 Performances of seven encoding bitmap indexes for small and large datasets. 66

LIST OF ALGORITHMS

Algorithm **Page**

3.1 The creation of Scatter bitmap index 26
4.1 The creation of HyBiX bitmap index 43
4.2 The query processing of HyBiX bitmap index 50

CHAPTER 1

INTRODUCTION

1.1 Background and rationale

Advances in technology have enabled massive volumes of data with various types of data formats generated by various sources, such as online transactions, social networks, sensor networks, and so on [1–4]. Over the decades, many organizations have been acknowledging the importance of the increasing data and its critical role in providing a useful and valuable information. Such data is potentially employed to support forecasting, and decision-making applications in many areas, for example, education, healthcare, violent reduction, and other areas [2, 3, 5, 6]. Most large-scale data is deployed in a read-only environment, which the data is not updated later but the insertion of new data is frequent. The main goal is to provide the ability to access the data efficiently and to facilitate the discovery of patterns and useful information from the data [1, 4, 6–8]. Actually, storage limits and unsatisfactory time of the query processing have been accounted a key issue in traditional data management systems with tremendous volumes of data. These have posed challenges and opportunities in efficiently storing and retrieving such volumes of data. In order to retrieve the data, various forms of the queries are submitted, for example, an equality query, and a range query. Those queries can take up to days to execute and return the results depending upon the complexity of queries because the entire data needs to be scanned and verified the query conditions [7, 9]. The efficiency of retrieval by query processing is a crucial issue when executing complex queries on large data by the traditional approaches [1, 7].

The query performance has been extensively studied, especially in a data warehouse environment, scientific applications, and many application domains with respect to query execution time. Several approaches have been introduced and demonstrated to achieve faster query processing, such as a parallel processing, materialized views, and indexing [7, 10]. Parallel processing plays a significant role in the processing of massive data [11–15]. This approach partitions the data into smaller pieces, and

assigns each piece to different machines or processors for processing data simultaneously, to speed up the execution. However, a disadvantage of parallel processing is that it requires additional hardware resources. For example, parallel processing can utilize all the cores of a CPU rather than just one. Furthermore, in some cases, parallel processing is not worthwhile due to interdependences that prevent splitting data to independently processing. Therefore, the design and implementation of parallel processing should be carefully considered. A materialized view contains aggregated and summarized results of the predicated queries [7, 10, 16], and helps optimize the performance of query processing when the predicated queries are known. However, the drawbacks of materialized views include the need to access and retrieve the actual data when unpredicted queries are submitted, and the need to update when data source is changed. Furthermore, it is quite difficult to build materialized views for all possible queries, and choosing which ones to build is an important problem.

Indexing is an effective technique and data structure to speed up the searches from storage, without requiring additional hardware resources [7, 9, 10, 17]. This approach can be categorized into three groups, namely hash-based indexes, tree-based indexes, and bitmap-based indexes [10, 16, 18–24]. The performance of query based on these three types of indexes differs for different forms of queries and for different data characteristics, for example, cardinality of indexed attributes (the number of unique values of that attribute). Typically, hash-based indexes [19, 20] provide a good query performance for equality queries, but it is not efficient for range queries due to the number of computation used by hash function considerably increases with the range of query values. Tree-based indexes provide a good query performance when formed on attributes with high cardinality and when used for optimizing queries that retrieve small numbers of rows [9, 15, 19, 25]. However, a lot of memory overhead exists on the tree-based indexes and the indexes cannot support logical operations on the index before accessing the actual data, resulting in high I/O operations required. Bitmap-based indexes take the advantage of supporting low-cost Boolean operations (AND, OR, NOT, and XOR) on the index [9, 10, 16]. The bitmap-based indexes are easy to represent the data in binary format, with 0's and 1's. Therefore, they offer a good query performance when

formed on attributes with low cardinality and for queries that retrieve large numbers of rows. From this fact, bitmap-based indexes are commonly used for optimizing queries in read-only environments due to the majority of complex queries usually require logical operations to obtain the result [22, 26, 27].

Basic bitmap index is efficient and effective for attributes with low cardinality [16]. Unfortunately, the size of the Basic bitmap index significantly increases with the cardinality of the indexed attribute, causing problems with space requirement [28, 29]. There are four strategies to improve the performance of the Basic bitmap index, including compression, binning, encoding, and multi-level and multi-component approach [9, 21, 25, 30–32]. Among these four, encoding plays an important role in improving the space requirements of the Basic bitmap index without sacrificing query execution time. The encoding bitmap indexes allow bitwise operators on the index directly, without decompression or additional processing. Furthermore, more studying of the encoding is more likely to be important and lead to better performance of the bitmap index, in terms of space and time trade-off with various types of queries.

Various encoding bitmap indexes have been studied in the past ten years, such as the Range bitmap index [33], Interval bitmap index [34], Encoded bitmap index [28], Scatter bitmap index [35], and Dual bitmap index [36]. When the cardinality of indexed attributes is high, some bitmap indexes suffer from impractical storage requirements. Other encoding bitmap indexes can solve this problem, but they still suffer from slow query processing for some types of queries, especially range queries. In the analytics applications, submitted types of queries are unpredictable, sometimes be equality or range queries [1, 21–23, 25, 26, 37]. Obviously, the existing encoding bitmap indexes have not fully solved problems with for example equality and range queries, in terms of space and time trade-off (i.e., trade-off between resources and efficiency).

To achieve good performance with various queries in terms of space and time trade-off, we introduce a new encoding bitmap index, namely HyBiX for Hybrid Encoding Bitmap Index, and its query processing with unpredictable queries. The results of an experiment in which space-efficiency, time-efficiency, and trade-off between space and time for existing encoding bitmap index on the benchmark dataset were compared.

The query execution time of HyBiX bitmap index for both equality and range queries is satisfactory, and the performance of HyBiX bitmap index in terms of space and time trade-off is good.

1.2 Research objectives

1.2.1 To study and analyze bitmap index strategies, including compressing, binning, encoding, and multi-level and multi-component, in order to reduce the bitmap index size as well as to improve query execution time.

1.2.2 To design and develop a new encoding scheme of the bitmap Index for efficient both index size and query processing.

1.2.3 To evaluate the efficiency of a new encoding bitmap index and the other existing encoding bitmap index by comparing:

- Space
- Query execution time
- Trade-off between space and time

under TPC-H, which is a decision support benchmark.

1.3 Research contributions

The proposed bitmap index is based on encoding bitmap index, called HyBiX for Hybrid Encoding Bitmap Index. The preliminary concept of the HyBiX bitmap index combines the grouping of attribute values and the concept of existing encoding bitmap indexes to enable efficiently answering both equality and range queries. The proposed bitmap index offers less storage requirements and facilitates a good performance in regard to query execution times when predicted queries are unknown. The performance of all encoding bitmap indexes, including the Basic, Range, Interval, Encoded, Scatter, Dual, and HyBiX bitmap indexes, was evaluated experimentally. The HyBiX bitmap index achieves a good performance for both equality and range queries, in points of view space-efficiency, time-efficiency, and space and time trade-off, especially when the query values fall in the first group of HyBiX bitmap index.

1.4 Dissertation organization

The rest of the dissertation is organized as follows. The next chapter describes the fundamental of the Basic bitmap index, and the strategies for the reduction of bitmap index size. In Chapter 3, algorithms to encoding bitmap indexes are investigated. Chapter 4 presents the proposed encoding bitmap index, named HyBiX for Hybrid Encoding Bitmap Index. In Chapter 5, the performance of the encoding bitmap indexes is experimentally evaluated, for both equality and range queries, in terms of space-efficiency, time-efficiency, and space and time trade-off. Finally, the major findings and the directions of further research are summarized in Chapter 6.

CHAPTER 2

Bitmap Index Strategies

This chapter describes the fundamental of the Basic bitmap index and bitmap index strategies for reducing size of bitmap index.

2.1 Basic bitmap index

A bitmap index is a well-known data structure for improving the speed of query processing in a data storage [9, 38]. Typically, the bitmap index contains a sequence of bits, one bitmap vector for a set of arbitrary attribute values, with each row representing an item. Each bitmap vector has an identifier related to the specified set of arbitrary values. These bitmap vectors are primarily used in logical bitwise operations to answer queries. In the simplest scheme, each bitmap vector corresponds to one precise value of an indexed attribute. This scheme is known as the Basic bitmap index, 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 i^{th} bit in bitmap vector for representing value v is set to 1 if the i^{th} row of indexed attribute contains value v. Otherwise, the bit is set to 0. The encoding function of the Basic bitmap index can be written in Eq. (2.1) .

$$B^j = \begin{cases} 1 & j = v \\ 0 & \text{Otherwise.} \end{cases} \tag{2.1}$$

Assume that a domain of attribute A given by table T is $\{0, 1, 2, \ldots, 14\}$, as shown in Figure 2.1a. 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 2.1b. As seen in Figure 2.1b, bitmap vector B^3 represents attribute value '3', and the 1^{st} and 6^{th} bits in B^3 are set to 1. When answering equality queries, only one bitmap vector associated with that query value is scanned. For example, in order to evaluate the equality query '$A = 3$', only bitmap vector B^3 is scanned. Then,

	A
1	3
2	9
3	14
4	8
5	10
6	3
7	4
8	0
9	12
10	5
.	.
.	.
.	.
100,000	2

(a) Table T

B^0	B^1	B^2	B^3	B^4	B^5	B^6	B^7	B^8	B^9	B^{10}	B^{11}	B^{12}	B^{13}	B^{14}
0	0	0	1	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	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
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	1	0	0	0	0
0	0	0	1	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
1	0	0	0	0	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	0	0	0	1	0	0	0	0	0	0	0	0	0
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

(b) Basic bitmap index

Figure 2.1 An example of the Basic bitmap index: encoding of attribute A with cardinality 15.

the 1st and 6th rows are returned as the final result due to the 1st and 6th bits in B^3 are set to 1. 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, in order to evaluate the range query '$1 \leq A \leq 4$', the bitmap vector B^1, B^2, B^3, and B^4 are scanned and performed bitwise-OR operators among those bitmap vectors (i.e., $B^1 \vee B^2 \vee B^3 \vee B^4$). Table T in Figure 2.1a is used as an example in what follows.

Advantage

The Basic bitmap index is efficient in index size and query execution time with low cardinality attributes which is recommended.

Limitations

With increasing cardinality of attribute, the Basic bitmap index generates a massive number of bitmap vectors, which impacts on index size. Therefore, the Basic bitmap index is inappropriate with high cardinality in point of view space requirement.

Clearly, the Basic bitmap index consumes storage proportional to the cardinality of the indexed attribute. Approaches to reduce excessive storage requirements can be classified into four types of strategies, including 1) Compression, 2) Binning, 3) Encoding, and 4) Multi-level & Multi-component, shown in Figure 2.2.

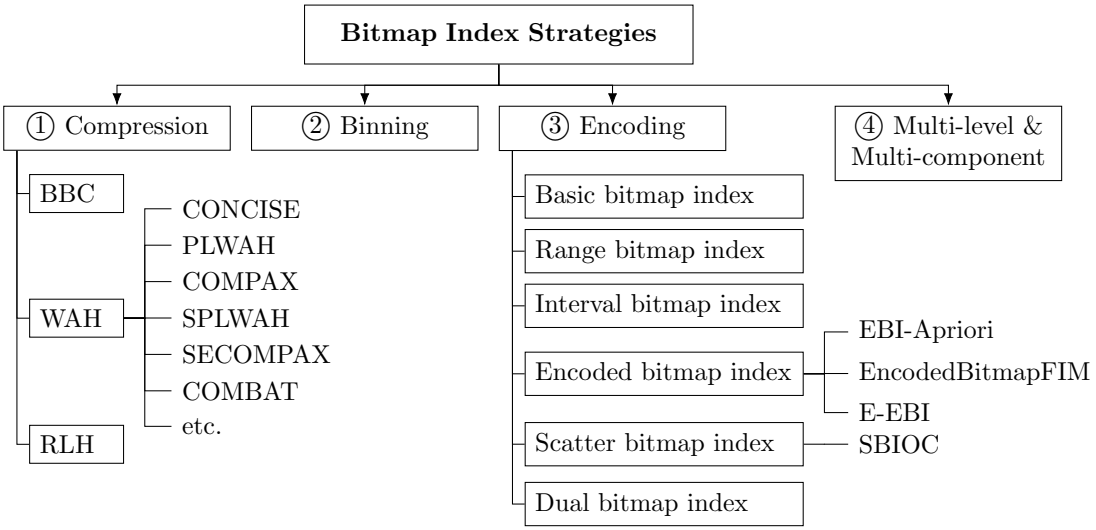


Figure 2.2 Bitmap index strategies.

2.2 Compression

Obviously, the Basic bitmap index generates a massive numbers of bitmap vector with high cardinality attributes. The long sequences of homogeneous bits (only bit 0s or only 1s) commonly exist in the bitmap vectors of the Basic bitmap index. Furthermore, each bitmap vector of the Basic bitmap index may be used separately from others. Then, bitmap compression is typically applied on each individual bitmap vector presented by long homogeneous segments. The notable bitmap compression algorithms used include Byte-aligned Bitmap Compression (BBC) [39], Word-aligned Hybrid (WAH) [17, 26, 40], Run-length Huffman Encoding (RLH) [31], which encodes the continuous sequences of bits as their value and count. Especially, the RLH combines run-length encoding and Huffman encoding to achieve higher compression ratio. Due to Huffman encoding constructs the Huffman tree for both encoding and decoding, the RLH requires some proportional of execution time to retrieve the Huffman tree and decode it into original sequences of bits. Therefore, the RLH is quite slow in execution time on both compression and decompression.

The BBC and WAH algorithms are a well-known compression algorithm in both space demanded and time consumed. The unit of compression used by BBC is 8 bits while that of compression used by WAH is 31 bits. The bitmap vectors compressed with BBC are slightly smaller in size than those compressed with WAH. However, the

WAH supports faster both building and query processing than BBC and RLH algorithms. A key concept of WAH is to allow compressed bitmap vectors to be stored in words which is the operational units of modern computer hardware.

Due to a literal word contains both bit value 0 and 1, there are few bits which are different from each other, called dirty bits, which degrades the compression ratios. From this fact, WAH algorithm cannot compress these literal words. Hereby, several algorithms based on WAH have been developed to improve compression ratios and deal with the dirty bits existing in the literal words, such as CONCISE [41], PLWAH [42], EWAH [43], COMPAX [44] and so on. In [42], Position List WAH (PLWAH) adapts the concept of Nearly identical (NI) in order to merge the Fill and Literal words into a single word by identifying one dirty bit in literal word. However, more than one dirty bit exists in the literal words, which the PLWAH confronts this problem in compression. Therefore, the COMPressed Adaptive indeX (COMPAX) [44] was introduced in order to consider a dirty byte, rather than a dirty bit. Both PLWAH and COMPAX are able to specify one dirty bit and dirty byte, respectively. They cannot compress the literal words, which contain more than one dirty bit or more than one dirty byte. To deal with this scenario, the Scope-Extended COMPressed Adaptive indeX (SECOMPAX) [45] was introduced in order to improve the compression ratios of COMPAX by dealing with more than one byte in the literal word, but not more than four bytes. This ability of SECOMPAX provides better compression ratios when many dirty bits spread in several bytes of the literal word. Furthermore, Switch Position List WAH (SPLWAH) [46] extended from PLWAH attempts to deal more than one dirty bit existing the literal words, but not more than four dirty bits. The compression ratios given by SPLWAH are definitely better than that used by PLWAH. Although these algorithms accomplish small index size, query processing time on the compressed bitmap vectors is considerably increasing due to decompression and logical operations dominate some proportional execution times, especially range queries. Table 2.1 shows a summarization of existing compression algorithms to the Basic bitmap index.

Table 2.1 A summarization of compression algorithms to the Basic bitmap index

Algorithm	Year	Method	Pros.	Cons.
BBC [39]	1995	- Compress Bitmap in length of 8 bits - Use header to identify fill bytes and literal bytes	- Easy to store in CPU memory - Easy to implement - Index size is compact	- Spend many times to load compressed Bitmap to CPU - Query processing is very slow
WAH [40]	2002	- Compress bitmap in 31- bits length - The first bit is used to identify the fill and the literal word	- Faster than BBC - Perform logical operation over compressed Bitmap Index without fully decompressing	- Index size is larger than BBC - Doesn't deal with nearly identical
RLH [31]	2007	- Based on Huffman encoding - Count the distance between bits of 1 - Compute the frequency of distances - Build a Huffman tree - Replace distance with their Huffman code	- Huffman tree is stored in CPU memory - Index size is smaller than WAH	- Huffman tree is used for both compressing and decompressing - A new Huffman tree must be created when updating the index

Continued on next page

Algorithm	Year	Method	Pros.	Cons.
CONCISE [41]	2010	- Use the concept of mixed fill word - Introduce the concept of flipped bit - Use piggyback concept - Compress Literal-Fill together	- Be able to identify the position of one dirty bit within a literal word and compress it - Index size is smaller than WAH	- Cannot compress when more than 1 dirty bit appear in a literal word - Query processing time is slower than WAH
PLWAH [42]	2010	- Introduce Nearly Identical concept - Apply the concept of codebook - Compress Fill-Literal word together	- Identify 1 dirty bit in a literal word - Index size is smaller than WAH and CONCISE	- Cannot compress when more than 1 dirty bit appear in a literal word - Query processing time is slower than WAH
EWAH [43]	2010	- Divide bit sequence into 32-bit word - 3 parts of header - 1 bit for type of fill word - 16 bits for number of fill word - 15 bits for dirty word - Followed by dirty words	- Access dirty word at most once - Faster query processing time than WAH	- Cannot deal with the nearly identical - Needs to access header 3 times

Continued on next page

Algorithm	Year	Method	Pros.	Cons.
COMPAX [44]	2010	- Apply Nearly Identical concept - Apply the concept of codebook - Compress when the case of 0Fill-Literal-0Fill, and Literal-0Fill-Literal occur	- Identify 1 dirty byte in a literal word, i.e., more than 1 bit - Index size is smaller than WAH and CONCISE	- Need to check types of codebook - Omit the consideration of consecutive bit 1s - Cannot compress when more than 1 dirty bit appear in a literal word - Query processing time is slower than WAH
GPU-WAH [47]	2010	- Based on WAH - Utilize GPU to accelerate compression time and query processing time	- Faster compression and query processing time than the traditional WAH	- Few overheads of execution occur when transferring data between CPU memory and GPU memory
GPU-PLWAH [15]	2011	- Based on PLWAH - Utilize GPU to accelerate compression time and query processing time	- Faster compression and query processing time than the traditional PLWAH	- Few overheads of execution occur when transferring data between CPU memory and GPU memory

Continued on next page

Algorithm	Year	Method	Pros.	Cons.
PWAH [48]	2011	- Divide into to partitions - Use header to specify fill or literal word - Use extended fill concept	- More flexible to save space - Index size is smaller than WAH	- The optimal number of partitions is an important issue
PLWAH+ [49]	2014	- Extension of PLWAH - Use Nearly Identical concept - Consider Literal-Fill and Fill-Literal	- Can compress when more than 1 bit appears in a literal word but less 4 bits	- Slower than WAH - Cannot compress when more than 4 bits appear in a literal word
VAL-WAH [50]	2014	- Variable block size - Dividing a word into block with specific length - Add header to identify fill and literal word - Apply WAH to compress	- More flexible - The optimal block size gives smaller index size than WAH	- Difficult to find the optimal block size - Query processing time is slower than WAH

Continued on next page

Algorithm	Year	Method	Pros.	Cons.
SECOMPAX [45]	2014	- Use Nearly Identical and codebook concept - Identify dirty byte within a literal word and compress	- Be able to perform compression if more than 1 bit occur within a literal word - Consider both the nearly identical of 0-fill and 1-fill	- Interpret types of codeword - Cannot identify more than 1 byte within a literal word - Query processing time is slower than WAH
SPLWAH [46]	2015	- Use Nearly Identical and codebook concept - Identify dirty bits within a literal word and then compress	- Be able to identify and compress 2-tuple and 3-tuple codebook - Index size is smaller than WAH - Be able to identify more than 1 dirty bit within a literal word	- Interpret types of codeword - Cannot identify more than 4 bits within a literal word - Query processing time is slower than WAH
SBH [32]	2016	- Combine concepts of BBC and WAH - Divide a sequence of bits into super buckets, containing 8 bits each - Apply WAH to compress - Use extended fill concept	- Index size is smaller than BBC and WAH - Slightly faster query processing time than WAH	- Difficult to find the optimal length of bit in super buckets - Don't achieve a good performance when the cardinality is less than 50

Continued on next page

Algorithm	Year	Method	Pros.	Cons.
COMBAT [51]	2016	- Similar to CONCISE and COMPAX - Compress two or three contiguous words into a single word - Introduce NI2-L for a literal word	- Be able to compress 2 dirty bytes in a literal word - Index size is smaller than CONCISE and COMPAX - Faster than CONCISE and COMPAX	- The codeword is more complex, which is an impact on high compression time

2.3 Binning

The Basic bitmap index works well for low cardinality attributes. However, for high cardinality attributes, the Basic bitmap index is impractical due to large storage requirement and computation time, particularly range query processing. Another strategy for reducing space requirements and improving query execution time of the Basic bitmap index is binning [9, 24, 25, 52]. This strategy partitions attribute values into smaller ranges, called bins. The bitmap vectors then represent the bins, rather than the distinct values. It is clear that this strategy enables to predefine the arbitrary number of bins, which impact the index size. Furthermore, some range queries can be accurately answered when they match the binning. For other queries, parts of original data have to be read from a storage and checked against the specified conditions. This additional process is called candidate check [9, 53]. The candidate check dominates the query processing time for equality and some of range queries, when the query does not match the binning. Furthermore, it is difficult to determine the optimal number of bins that balances the space requirements and the query time complexity.

2.4 Encoding

In the third strategy, the attribute values are encoded by a specific pattern of the bit 0s and 1s in bitmap vectors. The common encoding bitmap indexes have been implemented as Range bitmap index [33], Interval bitmap index [34], Encoded bitmap index [28], Scatter bitmap index [35], and Dual bitmap index [36], respectively. These encoding bitmap indexes minimize the number of bitmap vectors as well as improve query processing time by computing a retrieval function or retrieving encoding patterns from the mapping table. Furthermore, the encoding bitmap indexes allow us to use Boolean operations on the specified bitmap vectors without decompression or additional processes. For this reason, the encoding bitmap indexes are likely to improve the performance of the Basic bitmap index, in terms of space and time trade-off for various types of queries. The details of the encoding bitmap indexes will be described in Chapter 3.

2.5 Multi-level & Multi-component

More sophisticated strategies can combine the preceding three strategies. A method called multi-level bitmap index partitions the attribute values into multiple levels and encode each level separately. For example, the multi-level bitmap index can be built by using binning at the first level and using encoding at the next level, which removes the need of candidate check produced by binning strategy. Another method called multi-component bitmap index breaks the attribute values into several components and represent the components with encoding separately, which each component can generally have a different size. The simple example for multi-component bitmap index applies the converting between the decimal number and the number in any base. The attribute values are decomposed into digits according to the chosen base.

Using more levels and more components can significantly reduce the number of bitmap vectors and therefore reduce the total index size. However, the number of bitmap vectors accessed is considerably increasing when using more levels and more components, which is an impact on the increasing of query execution time. Furthermore, it is difficult to choose the optimal numbers of level or numbers of components for improving index size with maintaining query execution time.

In this chapter, several strategies have been discussed with improving space requirements of the Basic bitmap index. The compression strategy focuses on the reduction of space requirements on each bitmap vector. This strategy faces the problems of execution times over the compressed bitmap vectors. The ability of binning strategy offers the arbitrary of bitmap index size. However, this strategy requires the access and validation some parts of original data for some queries if they do not meet the binning. The encoding strategy is designed to tackle space requirements as well as query execution times by generating the small number of bitmap vectors. The good encoding design affects an efficiency in both space requirements and execution times. The multi-level and multi-component utilizes the combination of above three strategies to reduce space requirements. The strategy increases the complexity of designing bitmap index and query processing.

Obviously, the bitmap index with encoding significantly enables the efficient space requirements and query execution time, which is likely an impact on the performance in space requirement and execution time trade-off as well. Therefore, the next chapter gives the characteristics of existing encoding bitmap indexes.

CHAPTER 3

Existing Encoding Bitmap Indexes

The encoding strategy is regarded as being one of the preferable strategies for improving bitmap index if the small numbers of bitmap vectors are generated and these bitmap vectors are primarily used in the bitwise operations to precisely answer queries without any decompression and additional processing. This chapter describes the existing encoding bitmap indexes, including Range bitmap index, Interval bitmap index, Encoded bitmap index, Scatter bitmap index, and Dual bitmap index.

3.1 Range bitmap index

Let C be the attribute cardinality, which is the number of distinct values of that indexed attribute. The Range bitmap index formed on the range encoding scheme [33] produces a set of $C-1$ bitmap vectors, says $R = \{R^0, R^1, \ldots, R^{C-2}\}$. The attribute values represented by bitmap vector R^j are ranging from 0 to j. The encoding function for this bitmap index, for attribute value v, is given in Eq. (3.1)

$$R^j = \begin{cases} 1 & v \leq j \leq C-2 \\ 0 & \text{Otherwise.} \end{cases} \tag{3.1}$$

Assume a domain of attribute A given by table T is $\{0, 1, 2, \ldots, 14\}$, as shown in Figure 3.1a. The Range bitmap index therefore consists of 14 bitmap vectors since the cardinality of attribute A is 15, says $\{R^0, R^1, R^2, \ldots, R^{13}\}$. Using Eq. (3.1) to represent attribute value '3', all bits from R^3 to R^{13} are set to bit value 1; otherwise, the bits remained are set to bit value 0, and these are highlighted in Figure 3.1b.

Querying both equality and range on the Range bitmap index uses the retrieval function in Eq. (3.2). Clearly, the equality queries deploy the first three conditions in Eq. (3.2), and the range queries therefore deploy the remaining conditions.

	A
1	3
2	9
3	14
4	8
5	10
6	3
7	4
8	0
9	12
10	5
.	.
.	.
.	.
100,000	2

(a) Table T

R^0	R^1	R^2	R^3	R^4	R^5	R^6	R^7	R^8	R^9	R^{10}	R^{11}	R^{12}	R^{13}
0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1
.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.
0	0	1	1	1	1	1	1	1	1	1	1	1	1

(b) Range bitmap index

Figure 3.1 An example of the Range bitmap index: encoding of attribute A with $C = 15$.

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

$$v_1 \leq A \leq v_2 = \begin{cases} R^0 & v_1 = v_2 = 0, \\ R^{v_1} \oplus R^{v_1 - 1} & 0 < v_1 = v_2 < C - 1 \\ \overline{R^{C-2}} & v_1 = v_2 = C - 1 \\ \overline{R^{v_1 - 1}} & 0 < v_1 < C - 1, v_2 = C - 1 \\ R^{v_2} & v_1 = 0, 0 \leq v_2 \leq C - 1 \\ R^{v_2} \oplus R^{v_1 - 1} & \text{Otherwise.} \end{cases} \tag{3.2}$$

For example, to evaluate the equality query in the form of $A = 3$, using Eq. (3.2), the bitmap vector R^2 and R^3 were scanned, and then the bitwise-XOR operator is performed on them to answer this equality query, yields $R^2 \oplus R^3$. This query results the 1^{st} and 6^{th} rows.

To evaluate range query $1 \leq A \leq 4$, using Eq. (3.2), the bitmap vector R^0 and R^4 were scanned, and then the bitwise-XOR operator is performed on them to answer this range query, yields $R^4 \oplus R^0$.

Advantage

The Range bitmap index offers a good query performance for equality and range queries with low cardinality. Sometimes, the Range bitmap index scans only one bitmap vector to answer equality queries if the query value is equal to 0 or $C - 1$.

Limitations

The Range bitmap index decreases one bitmap vector from the Basic bitmap index, which still suffered storage problem with high cardinality attributes. The query performance of Range bitmap index is degraded with high cardinality attributes for both equality and range queries in point of views space and time trade-off.

3.2 Interval bitmap index

The Interval bitmap index based on the interval encoding scheme [34] 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. (3.3). Figure 3.2 shows as an example of the Interval bitmap index for the data in Figure 3.2a, with the 8 bitmap vectors, $\{I^0, I^1, \ldots, I^7\}$. On encoding attribute value '3', all bits between I^0 and I^3 are set to 1 and the remaining bits are set to 0.

	A
1	3
2	9
3	14
4	8
5	10
6	3
7	4
8	0
9	12
10	5
.	.
.	.
.	.
100,000	2

(a) Table T

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

(b) Interval bitmap index

Figure 3.2 An example of the Interval bitmap index: encoding of attribute A with $C = 15$.

$$I^j = \begin{cases} 1 & v \leq j \leq j+m \\ 0 & \text{Otherwise.} \end{cases} \tag{3.3}$$

The retrieval functions in Eq. (3.4) – (3.6) checks for equality, one-side range, and two-side range queries, respectively.

For example, to evaluate the equality query in the form of $A = 3$, then the retrieval function of this query is $I^3 \wedge \overline{I^4}$. To evaluate range query in the form of $1 \leq A \leq 4$, the bitmap vector I^1 and I^5 were scanned, and then the bitwise-OR operator is performed on them, yields $I^1 \wedge \overline{I^5}$.

$$A = v = \begin{cases} I^0 & v = 0, m = 0 \\ \overline{I^0} & v = 1, C = 2 \\ I^1 & v = 1, C = 3 \\ I^v \wedge \overline{I^{v+1}} & v < m \\ I^1 \wedge I^0 & v = m, m > 0 \\ I^{v-m} \wedge \overline{I^{v-m-1}} & m < v < C-1, m > 0 \\ \overline{I^{\lceil \frac{C}{2} \rceil - 1} \vee I^0} & v = C-1. \end{cases} \tag{3.4}$$

For $0 < v < C - 1$,

$$A \leq v = \begin{cases} I^0 \wedge \overline{I^{v+1}} & v < m \\ I^0 & v = m \\ I^0 \vee I^{v-m} & m < v < C-1. \end{cases} \tag{3.5}$$

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

$$v_1 \leq A \leq v_2 = \begin{cases} I^{v_1} \wedge \overline{I^{v_2+1}} & v_2 < m \\ I^{v_1} \wedge I^0 & v_2 = m \\ I^{v_1} \wedge I^{v_2-m} & v_2 < v_1 + m, v_1 < n \\ I^{v_1} & v_2 = v_1 + m, v_1 < n \\ I^{v_1} \vee I^{v_2-m} & v_2 > v_1 + m, v_1 < m \\ I^{v_1} \vee I^{v_1+1} & v_2 = v_1 + m + 1, v_1 = m \\ I^{v_2-m} \wedge \overline{I^{v_1-m-1}} & v_1 \geq n. \end{cases} \tag{3.6}$$

Advantage

The index size used by the Interval bitmap index is much smaller than that used by the Basic and Range bitmap index. In addition, the Interval bitmap index offers improved query performance against the Range bitmap index for both equality and range queries with low cardinality.

Limitations

The Interval bitmap index still produces many bitmap vectors with high cardinality, which is an impact on storage requirement. The performance of Interval bitmap index is degraded with high cardinality attributes for both equality and range queries in point of views space and time trade-off.

3.3 Encoded bitmap index

To our knowledge, the Encoded bitmap index [28] 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 which stores the binary patterns of all distinct attribute values. The attribute values are encoded with $\lceil \log_2 C \rceil$ bits in corresponding position of the bitmap vectors. Figure 3.3 shows an example of the Encoded bitmap index for the attribute with cardinality 15, 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 attribute value '3'

	A
1	3
2	9
3	14
4	8
5	10
6	3
7	4
8	0
9	12
10	5
.	.
.	.
.	.
100,000	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

(b) Encoded bitmap index with a mapping table

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

Figure 3.3 An example of the Encoded bitmap index: encoding of attribute A with $C = 15$.

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 evaluate 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. Unfortunately, the traditional equality query processing used by the Encoded bitmap index takes a long time to answer the query because of the comparison of all $\lceil \log_2 C \rceil$ bitmap vectors. Accordingly, the E-EBI [54] is introduced to improve the traditional equality query processing in the Encoded bitmap index without comparison all bitmap vectors. In this algorithm, the bitmap vector can be performed bitwise-AND operation directly if the corresponding bit is set to 1. Otherwise, the negation of the bitmap vector is required before performing bitwise-AND operation. For example, to evaluate the equality query $A = 3$, the binary pattern for attribute value '3' is '0011', given by the mapping table in Figure 3.3b. Therefore, the negation of bitmap vector E^0 and E^1 is required before performing bitwise-AND operation. As a result, the retrieval function of this query is simply created as $\overline{E^0 E^1} E^2 E^3$.

For evaluating range queries, the retrieval function for the query is formed of Boolean expressions that apply bitwise-OR operators on the expressions

to get the final result. For example, to evaluate range query $1 \leq A \leq 4$, the binary patterns for each item are retrieved from the mapping table, and transformed to Boolean expressions, yields $\overline{E^0E^1E^2}E^3$ for value 1, $\overline{E^0E^1}E^2\overline{E^3}$ for value 2, $\overline{E^0E^1}E^2E^3$ for value 3, $\overline{E^0}E^1\overline{E^2E^3}$ for value 4. Then, the bitwise-OR operators are used on them, yields $(\overline{E^0E^1E^2}E^3) \vee (\overline{E^0E^1}E^2\overline{E^3}) \vee (\overline{E^0E^1}E^2E^3) \vee (\overline{E^0}E^1\overline{E^2E^3})$. Furthermore, the generated retrieval function can be further reduced to optimize range query performance by utilizing Boolean minimization method, such as Quine-McCluskey algorithm [55, 56]. The reduced retrieval function by Quine-McCluskey algorithm is generated as $(\overline{E^0E^1}E^3) \vee (\overline{E^0E^1}E^2\overline{E^3}) \vee (\overline{E^0}E^1\overline{E^2E^3})$. Additionally, the improved algorithms for Encoded bitmap index were introduced for querying equality and range queries by using data mining techniques and parallel processing over large dataset [54, 57–59]. However, both equality and range queries on the Encoded bitmap index take long execution times, even though this bitmap index is effective from the space requirement point of view.

Advantage

The Encoded bitmap index requires the smallest number of bitmap vectors for all cardinalities, which is an efficiency in space requirement.

Limitations

The query execution time taken by Encoded bitmap index is undesirable for both equality and range queries. Even though the Encoded bitmap index uses the Boolean minimization method in range queries to reduce the complexity of the retrieval function, it considerably takes long processing times with range queries.

3.4 Scatter bitmap index

For Scatter bitmap index [35], the bitmap vectors are split into two groups, namely Z-group and L-group. The Scatter bitmap index uses $\lceil 2\sqrt{C} \rceil$ bitmap vectors. The Z-group contains $\lceil \frac{C}{m-1} \rceil + 1$ bitmap vectors, says $\{Z^0, Z^1, \ldots, Z^{\lceil \frac{C}{m-1} \rceil}\}$, while the L-group contains $m - 2$ bitmap vectors, say $\{L^1, L^2, \ldots, L^{m-2}\}$, where $m = \lceil \sqrt{C} \rceil + 1$.

The algorithm for creation Scatter bitmap index is shown in Algorithm 3.1. If the value 'v' at i$^{\text{th}}$ row relates to Z^{j-1} and Z^j (or L^k and Z^j), the bits in Z^{j-1} and Z^j (or L^k and Z^j) at i$^{\text{th}}$ row are set to 1. Otherwise, they are set to 0. Figure 3.4 depicts

Algorithm 3.1 The creation of Scatter bitmap index

INPUT: The cardinality and values of of the indexed attribute
OUTPUT: The scatter bitmap index

1: $m \leftarrow \lceil \sqrt{C} \rceil + 1$
2: **for** a value $'v'$ in each row **do**
3: Initial all bits in the bitmap vectors of Z- and L-group to be 0
4: $j \leftarrow \lfloor \frac{v}{m-1} \rfloor + 1$
5: $k \leftarrow v \mod m - 1$
6: **if** $k = 0$ **then**
7: Set bit of Z^{j-1} and Z^j to be 1
8: **else**
9: Set bit of L^k and Z^j to be 1
10: **end if**
11: **end for**

an example of Scatter bitmap index for an attribute of cardinality 15, which consists of 8 bitmap vectors, say $\{Z^0, Z^1, Z^2, Z^3, Z^4, L^1, L^2, L^3\}$, and $m = 5$. On encoding attribute value '3', the bits in Z^1 and L^3 are set to 1 and the remaining bits are set to 0, due to the values of j and k are 1 and 3, respective, corresponding to the 2nd and 11th steps in Algorithm 3.1. Then, the bits in Z^1 and L^3 are set to 1 and the remaining bits are set to 0.

	A
1	3
2	9
3	14
4	8
5	10
6	3
7	4
8	0
9	12
10	5
.	.
.	.
.	.
100,000	2

(a) Table T

Z^0	Z^1	Z^2	Z^3	Z^4	L^1	L^2	L^3
0	1	0	0	0	0	0	1
0	0	0	1	0	1	0	0
0	0	0	0	1	0	1	0
0	0	1	1	0	0	0	0
0	0	0	1	0	0	1	0
0	1	0	0	0	0	0	1
0	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
0	1	0	0	0	0	1	0

(b) Scatter bitmap index

Figure 3.4 An example of the Scatter bitmap index: encoding of attribute A with $C = 15$.

Equality queries with Scatter bitmap index use the retrieval function in Eq. (3.7). For example, to answer the equality query $A = 3$. Using Eq. (3.7), the retrieval function of this query is $Z^1 \wedge L^3$.

$$A = v = \begin{cases} Z^{j-1} \wedge Z^{j} & k = 0 \\ Z^{j} \wedge L^{k} & \text{Otherwise.} \end{cases} \tag{3.7}$$

where: $m = \left\lceil \sqrt{C} \right\rceil + 1$

$j = \left\lfloor \frac{v}{m-1} \right\rfloor + 1$

$k = v \bmod (m-1)$

For evaluating range queries, two bitmap vectors associated with each query value can be dynamically created by using Eq. (3.7), and the retrieval is performed with bitwise-OR operations. For example, to answer the range query $1 \leq A \leq 4$, by using Eq. (3.7), the retrieval functions for representing each item are dynamically generated as $Z^1 \wedge L^1$ for value 1, $Z^1 \wedge L^2$ for value 2, $Z^1 \wedge L^1$ for value 3, $Z^1 \wedge Z^2$ for value 4. Then, the bitwise-OR operators are used on them, yields $(Z^1 \wedge L^1) \vee (Z^1 \wedge L^2) \vee (Z^1 \wedge L^2) \vee (Z^1 \wedge Z^2)$. Furthermore, the retrieval function can be further minimized, which impacts the numbers of bitmap vectors accessed and the numbers of Boolean operations used. The basis idea of Dual-simRQ [60] is modified and applied to improve query processing, especially range queries. Therefore, the final reduced retrieval function is generated as $\left(Z^1 \wedge (L^1 \vee L^2 \vee L^3)\right) \vee (Z^1 \wedge Z^2)$. Additionally, the data clustering technique was employed to optimize the query processing on the Scatter bitmap index by grouping the attribute values which is frequently queried [61], to improve query execution time used by Scatter bitmap index.

Advantage

The Scatter bitmap index requires the less space than the Basic, Range, and Interval bitmap indexes, except the Encoded bitmap index. The Scatter bitmap index is suitable for equality queries because of scanning two bitmap vectors.

Limitations

The query execution time used by Scatter bitmap index is slower than the Basic bitmap index for equality queries. For range queries, the query execution time used by Scatter bitmap index is undesirable. Therefore, the performance of Scatter bitmap index with range queries is poor in space vs. time trade-off point of view.

3.5 Dual bitmap index

In the Scatter bitmap index, one bitmap vector (i.e., Z^0) is used to represent one value, which wastes space. Improving the Scatter bitmap index, the Dual bitmap index [36] efficiently represents attribute values while using two bitmap vectors. The Dual bitmap index consists of $\lceil\sqrt{2C + 0.25} + 0.5\rceil$ 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. (3.8).

$$D^j = \begin{cases} 1 & j = r \text{ and } j = s \\ 0 & \text{Otherwise.} \end{cases} \tag{3.8}$$

where: $hiC = \frac{n(n-1)}{2}$

$$r = \left\lceil \sqrt{2(hiC - v) + 0.25} + 0.5 \right\rceil$$

$$s = \left\lceil r - 1 - \left[\left(v - \frac{(n-r)(n-r-1)}{2} \right) \bmod r \right] \right\rceil$$

Figure 3.5 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. (3.8), to represent attribute value '3', the bits in D^1 and D^5 are set to 1, while the remaining bits are set to 0.

	A
1	3
2	9
3	14
4	8
5	10
6	3
7	4
8	0
9	12
10	5
.	.
.	.
.	.
100,000	2

(a) Table T

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

(b) Dual bitmap index

Figure 3.5 An example of the Dual bitmap index: encoding of attribute A with $C = 15$.

Evaluation of equality queries with the Dual bitmap index uses the retrieval

function in Eq. (3.9). For example, to answer the equality query $A = 3$. Using Eq. (3.9), the retrieval function of this query is $D^5 \wedge D^1$.

$$A = v = D^r \wedge D^s \tag{3.9}$$

To answer range queries, the retrieval function can be dynamically created and performed bitwise-OR operators, similar to the case with Scatter bitmap index. For example, to answer the range query $1 \leq A \leq 4$, by using Eq. (3.9), the retrieval functions for representing each item are dynamically generated as $D^5 \wedge D^3$ for value 1, $D^5 \wedge D^2$ for value 2, $D^5 \wedge D^1$ for value 3, $D^5 \wedge D^0$ for value 4. Then, the bitwise-OR operators are used on them, yields $(D^5 \wedge D^3) \vee (D^5 \wedge D^2) \vee (D^5 \wedge D^1) \vee (D^5 \wedge D^0)$. In addition, the retrieval function can be minimized to reduce the scanning of bitmap vectors as well as the number of Boolean operations, which impacts the query execution time taken. Therefore, Dual-simRQ [60] was proposed to improve the query execution time with range queries. The reduced retrieval function generated by Dual-simRQ is $D^5 \wedge (D^3 \vee D^2 \vee D^1 \vee D^0)$.

Advantage

The Dual bitmap index requires the less space than the Basic, Range, Interval, and Scatter bitmap indexes, except the Encoded bitmap index. The performance of Dual bitmap index is better than the existing bitmap indexes in terms of space and time trade-off for equality queries.

Limitations

The query execution time used by Dual bitmap index is slower than the Basic bitmap index for equality queries. Furthermore, the query execution time with range queries used by Dual bitmap index is undesirable. Therefore, the performance of Dual bitmap index is degraded in space vs. time trade-off for range queries.

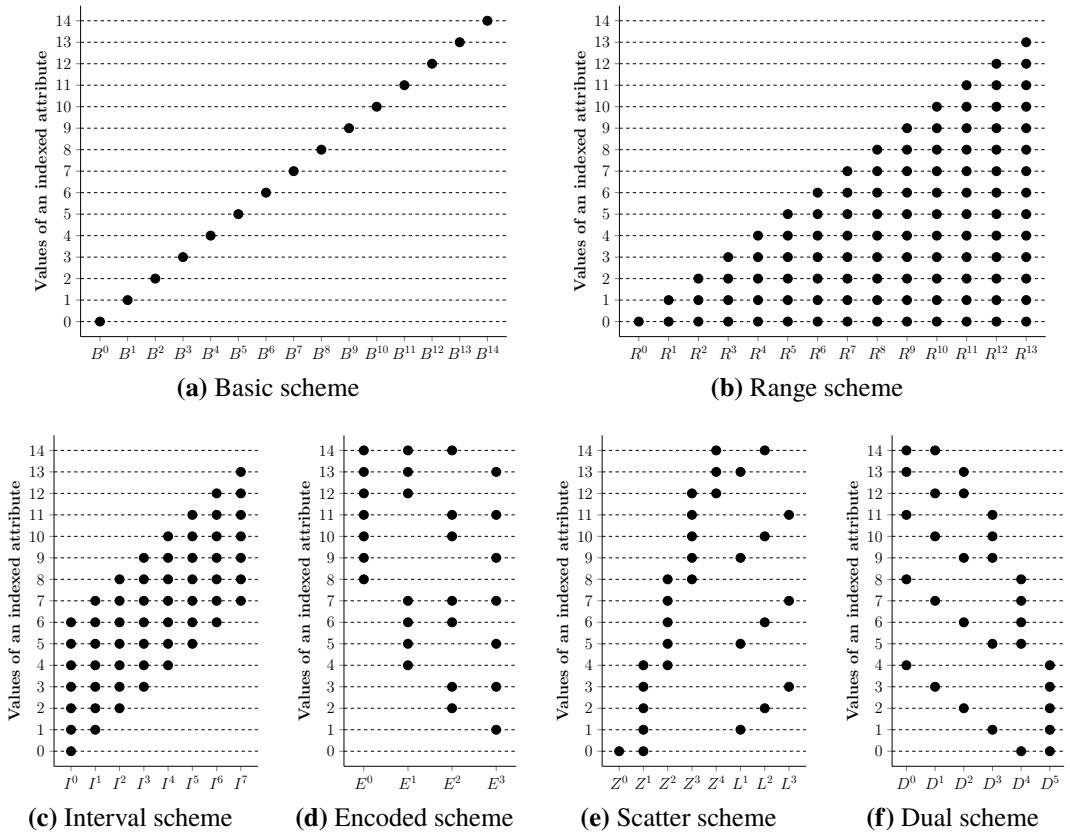
The numbers of bitmap vectors used for encoding bitmap indexes are summarized in Table 3.1. The Basic bitmap index uses C bitmap vectors while the Range and Interval bitmap indexes decrease the number of bitmap vectors by one and half, respectively. The Encoded bitmap index uses $\lceil \log_2 C \rceil$ bitmap vectors. The Scatter and

Dual bitmap indexes utilize $\lceil 2\sqrt{C} \rceil$ and $\lceil \sqrt{2C + 0.25} + 0.5 \rceil$ bitmap vectors, respectively.

Table 3.1 A summarization of the number of bitmap vectors used for six encoding bitmap indexes

Bitmap index	The number of bitmap vectors used
Basic	C
Range	$C - 1$
Interval	$\lceil \frac{C}{2} \rceil$
Encoded	$\lceil \log_2 C \rceil$
Scatter	$\lceil 2\sqrt{C} \rceil$
Dual	$\lceil \sqrt{2C + 0.25} + 0.5 \rceil$

Figure 3.6 illustrates the six encoding schemes with $C = 15$. Note that the black dots denote bit value 1. The Basic bitmap index uses 15 bitmap vectors, shown in Figure 3.6a. The Range bitmap index uses 14 bitmap vectors while the Interval bitmap index uses 8 bitmap vectors to represent the attribute values, shown in Figure 3.6b and



(a) Basic scheme **(b)** Range scheme

(c) Interval scheme **(d)** Encoded scheme **(e)** Scatter scheme **(f)** Dual scheme

Figure 3.6 Six encoding schemes with $C = 15$, (• represents bit 1).

3.6c, respectively. The Encoded bitmap index uses 4 bitmap vectors, shown in Figure 3.6d. Figures 3.6e and 3.6f show the encoding schemes for the Scatter and Dual bitmap indexes, which use 8 and 6 bitmap vectors, respectively.

This chapter described the characteristics of five encoding bitmap indexes, including Range, Interval, Encoded, Scatter, and Dual bitmap indexes. The Range and Interval bitmap indexes are efficient for equality and range queries by setting bit value 1 in the consecutive bitmap vectors to represent attribute values. However, those bitmap indexes suffer from high storage requirements, due to the large numbers of bitmap vectors. While the Encoded bitmap index uses the smallest number of bitmap vectors, it is inefficient with equality and range queries. The Scatter and Dual bitmap indexes can improve the performance for equality query processing, in terms of space and time trade-off, by accessing two bitmap vectors. Unfortunately, the range query processing is unsatisfactory. Table 3.2 summarizes the advantages and limitations of six encoding bitmap index algorithms.

As aforementioned, the existing encoding bitmap indexes are not be able to fully solve the problems of both space requirements and execution times with a variety of submitted queries. Therefore, the proposed encoding bitmap index, called HyBiX for Hybrid Encoding Bitmap Index, will be explained in the next chapter, to deal with those problems.

Table 3.2 A summarization of encoding bitmap index algorithms

Algorithm	Year	Method	Pros.	Cons.
Basic bitmap index [16]	1997	- Formed on equality encoding - Use one bitmap vector for representing one attribute value	- Easy to represent the data - Suited for equality queries - Suited for attribute with low cardinality	- Require a massive storage when build on attribute with high cardinality - Consume long times for answering range queries
Range bitmap index [33]	1998	- Formed on range encoding - Each attribute value is represented by the specific consecutive bitmap vectors - Access 2 bitmap vectors to answer the queries	- Suited for equality and one-side range queries - Suited for attribute with low cardinality	- Index size is dramatically increased when cardinality of indexed attribute is high

Continued on next page

Algorithm	Year	Method	Pros.	Cons.
Interval bitmap index [34]	1999	- Formed on interval encoding - Each attribute value is represented by the specific consecutive bitmap vectors - Access 2 bitmap vectors to answer the queries	- Index size is smaller than Basic and Range bitmap indexes - Suited for equality queries, one-side, and two-side range queries - Suited for attribute with low cardinality	- Index size is dramatically increased when cardinality of indexed attribute is high
Encoded bitmap index [28]	1998	- Formed on binary encoding - Use a mapping table	- Index size is the smallest comparing with existing other encoding bitmap indexes - Suited for attribute with high cardinality	- Take a long query execution time with both equality and range queries - Need to look up at a mapping table and access all bitmap vectors
Scatter bitmap index [35]	2006	- Divide bitmap vectors into 2 groups - Each indexed value is calculated and place into the group - Use 2 bitmap vectors to represent each attribute value	- Index size is the smaller than the Basic, Range, and Interval bitmap indexes - Suited for equality queries - Use 2 bitmap vectors to answer equality queries	- Waste one bitmap vector to represent one value (i.e., Z^0) - Take long times to answer range queries

Continued on next page

Algorithm	Year	Method	Pros.	Cons.
Dual bitmap index [35]	2006	- Improve space requirement of Scatter bitmap index - Use 2 bitmap vectors to represent each attribute value	- Index size is smaller than the Basic, Range, Interval and Scatter bitmap index - Suited for equality queries - Use 2 bitmap vectors to answer equality queries	- Take long times to answer range queries

CHAPTER 4

HyBiX: Hybrid Encoding Bitmap Index

In previous chapter, the characteristics of various encoding bitmap indexes was described. Most of researches on encoding bitmap indexes are interested in reducing storage requirements with maintaining query execution times for various queries. The Basic bitmap index provides an efficient space requirement with low cardinality attributes and it offers the fastest query execution times with equality queries. Unfortunately, the query execution times used by the Basic bitmap index is unsatisfactory. The Range and Interval bitmap indexes offer a better efficiency of query execution times with both equality and range queries. However, they suffer the storage requirement problem against the high cardinality attributes. Although the Encoded bitmap index gives the smallest index size for all cardinalities of attributes, it consumes a long query execution times with both equality and range queries. The Scatter and Dual bitmap indexes are improved space requirement with high cardinality attributes and also improved query execution times with equality queries. Nevertheless, the query execution time with range queries used by the Scatter and Dual bitmap indexes is unsatisfactory. In the real world applications, the submitted queries could be both equality and range queries. Clearly, the existing encoding bitmap index cannot deal with execution times with various submitted queries efficiently, with using a small space requirement. Consequently, this dissertation proposes a new encoding bitmap index, namely HyBiX bitmap index for Hybrid Encoding bitmap index, which uses the small space requirement and provides a good query execution time with equality and range queries.

This chapter describes the design concept of the HyBiX bitmap index, the bitmap index creation, and query processing for HyBiX bitmap index.

4.1 The design concept of HyBiX bitmap index

The HyBiX bitmap index takes the strengths of grouping the indexed attribute values and the design concepts in existing encoding bitmap indexes (i.e., Range

and Dual bitmap indexes) to reduce the storage requirements as well as to improve query processing times, with big data and with high attribute cardinality. Due to grouping the attribute values, the numbers of attribute values contained in each group must be carefully considered. If the numbers of attribute values in each group are equal, the encoding scheme of some attribute values will be duplicated. This is difficult to precisely answer the query values over the bitmap index, which requires the access to the raw data and examines the predicted query values. Therefore, the basic concept of HyBiX bitmap index is to divide the distinct attribute values into several groups under the condition that the amount of values in each group must be unequal. Therefore, the amount of values in each group decreases by 1 in order to efficiently represent attribute values as many as possible. Next, the number of groups and the numbers of values in each group is calculated, as described below.

From the above ideas, let n denotes a possible number of groups and a total number of bitmap vectors. Generally, regarding the amounts of values in each group, the group i has $n - i$ values, where $0 \leq i \leq n - 1$. Therefore, the first group of 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 in every group must be greater than or equal to the cardinality of indexed attribute, as seen in Eq. (4.1).

$$n + (n - 1) + (n - 2) + \cdots + 1 \geq C \tag{4.1}$$

The above equation can be solved as following:

$$\sum_{i=1}^{n} i \geq C$$

$$\frac{n(n+1)}{2} \geq C$$

$$n^2 + n \geq 2C$$

$$n^2 + \frac{2n}{2} + \left(\frac{1}{2}\right)^2 \geq 2C + \left(\frac{1}{2}\right)^2$$

$$\left(n + \frac{1}{2}\right)^2 \geq 2C + \frac{1}{4}$$

$$n + \frac{1}{2} \leq -\sqrt{2C + \frac{1}{4}} \quad \text{or} \quad n + \frac{1}{2} \geq \sqrt{2C + \frac{1}{4}}$$
$$n \leq -\sqrt{2C + \frac{1}{4}} - \frac{1}{2} \quad \text{or} \quad n \geq \sqrt{2C + \frac{1}{4}} - \frac{1}{2}$$

Since the number of groups and the total number of bitmap vectors must be the smallest positive integer, it gives

$$n = \left\lceil \sqrt{2C + 0.25} - 0.5 \right\rceil \tag{4.2}$$

where C is the cardinality of the indexed attribute.

Additionally, the maximum values of C that can be represented by n bitmap vectors is then $n + (n-1) + (n-2) + \cdots + 1$. Therefore,

$$\begin{aligned} C_{max} &= n + (n-1) + (n-2) + \cdots + 1 \\ &= \sum_{i=1}^{n} i \\ &= \frac{n(n+1)}{2} \end{aligned}$$

Assume that the cardinality of attribute A is 15 ($C = 15$), then, the total number of bitmap vectors used for HyBiX bitmap index can be calculated as followed.

$$\begin{aligned} n &= \left\lceil \sqrt{2(15) + 0.25} - 0.5 \right\rceil \\ n &= \left\lceil \sqrt{30.25} - 0.5 \right\rceil \\ n &= \lceil 5.5 - 0.5 \rceil = 5 \end{aligned}$$

Therefore, the HyBiX bitmap index built on the attribute with $C = 15$ uses 5 bitmap vectors, which can indicate 15 values as the maximum. Furthermore, it is implied that the total number of groups is also 5 as well.

Let H denotes a set of bitmap vectors for HyBiX bitmap index and $H = \{H^0, H^1, \ldots, H^{n-1}\}$. To represent attribute values, the bitmap vector H^i represents for all values in group i and for some values in groups $i-1, i-2, \ldots, 0$. For example,

as seen in Figure 4.1, the bitmap vector H^2 represents all values of group 2 (i.e., values 9, 10, and 11) and also represent some values of group 0 and group 1 (i.e., values 2, 3, 4 for group 0 and values 6, 7, 8 for group 1). Figure 4.2 illustrates the encoding scheme used by HyBiX bitmap index for the attribute with $C = 15$, where • represents bit value 1.

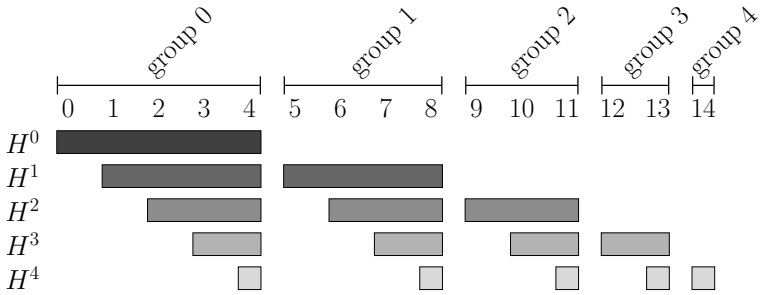


Figure 4.1 A basic concept of HyBiX bitmap index with cardinality 15.

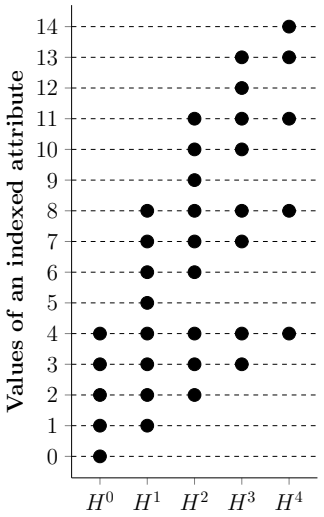


Figure 4.2 HyBiX scheme for attribute A with cardinality 15.

As shown in Figures 4.1 and 4.2, there are four elements characterizing the attribute values in order to facilitate the encoding of attribute values for the HyBiX bitmap index, consisting of group containing the value (g_v), starting value in the group (s_v), ending value in the group (e_v), and level of the value in the group (l_v). Each element is individually clarified as follows.

Definitely, the attribute values are continuously increasing by starting from 0. The first group (i.e., group 0) contain n values. The amount of attribute values in the next group is decreased by 1 from the previous group. Therefore, the group 0 has

the values between 0 and $n-1$, the group 1 has the values between n and $n+(n-1)-1$, and so on, as described below.

Group	**Values**
0	$0 \leq v \leq n-1$
1	$n \leq v \leq n+(n-1)-1$
2	$n+(n-1) \leq v \leq n+(n-1)+(n-2)-1$
3	$n+(n-1)+(n-2) \leq v \leq n+(n-1)+(n-2)+(n-3)-1$

where $v \in [0, C-1]$.

From the above, we consider the range of values contained in the specific groups, as shown in Eq. (4.3).

$$\text{for any group } (g_v): \quad \sum_{i=1}^{n} i - \sum_{i=1}^{n-g_v} i \leq v \leq \left(\sum_{i=1}^{n} i - \sum_{i=1}^{n-g_v-1} i \right) - 1 \tag{4.3}$$

Consider the left side: $\sum_{i=1}^{n} i - \sum_{i=1}^{n-g_v} i \leq v$, where $C_{max} = \sum_{i=1}^{n} i$.

$$\begin{aligned}
C_{max} - \frac{(n-g_v)(n-g_v+1)}{2} &\leq v \\
(n-g_v)(n-g_v+1) &\geq 2(C_{max}-v) \\
(n-g_v)^2 + (n-g_v) &\geq 2(C_{max}-v) \\
(n-g_v)^2 + \frac{2(n-g_v)}{2} + \left(\frac{1}{2}\right)^2 &\geq 2(C_{max}-v) + \left(\frac{1}{2}\right)^2 \\
\left(n-g_v+\frac{1}{2}\right)^2 &\geq 2(C_{max}-v) + \left(\frac{1}{4}\right) \\
n-g_v+\frac{1}{2} \geq +\sqrt{2(C_{max}-v)+\frac{1}{4}} \text{ or } n-g_v+\frac{1}{2} &\leq -\sqrt{2(C_{max}-v)+\frac{1}{4}}
\end{aligned}$$

The value of group must be a positive number. So, we get

$$n - g_v \geq \sqrt{2(C_{max}-v)+\frac{1}{4}} - \frac{1}{2}$$

Consider the right side: $v \leq \left(\sum_{i=1}^{n} i - \sum_{i=1}^{n-g_v-1} i \right) - 1$.

$$v \leq \left(C_{max} - \frac{(n - g_v - 1)(n - g_v)}{2}\right) - 1$$

$$(n - g_v - 1)(n - g_v) \leq 2(C_{max} - v - 1)$$

$$(n - g_v)^2 - (n - g_v) \leq 2(C_{max} - v - 1)$$

$$(n - g_v)^2 - \frac{2(n - g_v)}{2} + \left(\frac{1}{2}\right)^2 \leq 2(C_{max} - v - 1) + \left(\frac{1}{2}\right)^2$$

$$\left(n - g_v - \frac{1}{2}\right)^2 \leq 2(C_{max} - v - 1) + \left(\frac{1}{4}\right)$$

$$n - g_v - \frac{1}{2} \leq +\sqrt{2(C_{max} - v - 1) + \frac{1}{4}} \text{ or } n - g_v - \frac{1}{2} \geq -\sqrt{2(C_{max} - v - 1) + \frac{1}{4}}$$

The value of group must be a positive number. So, we get

$$n - g_v \leq \sqrt{2(C_{max} - v - 1) + \frac{1}{4}} + \frac{1}{2}$$

To satisfy Eq. (4.3), we check $\sqrt{2X + \frac{1}{4}} - \frac{1}{2} \leq \sqrt{2(X-1) + \frac{1}{4}} + \frac{1}{2}$ for the value of $n - g_v$, where $X = C_{max} - v$, as follows.

$$0 \leq X - 1$$

$$0 \leq 2(X - 1)$$

$$\frac{1}{4} \leq 2(X - 1) + \frac{1}{4}$$

$$\frac{1}{2} \leq \sqrt{2(X - 1) + \frac{1}{4}}$$

$$1 \leq 2\sqrt{2(X - 1) + \frac{1}{4}}$$

$$\left(\sqrt{2(X - 1) + \frac{1}{4}}\right)^2 + 1 + 1 \leq \left(\sqrt{2(X - 1) + \frac{1}{4}}\right)^2 + 2\sqrt{2(X - 1) + \frac{1}{4}} + 1$$

$$2(X - 1) + \frac{1}{4} + 2 \leq \left(\sqrt{2(X - 1) + \frac{1}{4}} + 1\right)^2$$

$$2X + \frac{1}{4} \leq \left(\sqrt{2(X - 1) + \frac{1}{4}} + 1\right)^2$$

$$\sqrt{2X + \frac{1}{4}} \leq \sqrt{2(X - 1) + \frac{1}{4}} + 1$$

$$\sqrt{2X+\frac{1}{4}}-\frac{1}{2} \leq \sqrt{2(X-1)+\frac{1}{4}}+\frac{1}{2}$$

Evidently, the value $n-g_v$ is between $\sqrt{2(C_{max}-v)+\frac{1}{4}}-\frac{1}{2}$ and $\sqrt{2(C_{max}-v-1)+\frac{1}{4}}+\frac{1}{2}$. Indeed, an integer number of value g_v is expected, so we get

$$g_v = n - \left\lceil \sqrt{2(C_{max}-v)+\frac{1}{4}}-\frac{1}{2} \right\rceil = n - \left\lfloor \sqrt{2(C_{max}-v-1)+\frac{1}{4}}+\frac{1}{2} \right\rfloor$$

Consequently, the equation for calculating the group containing the predicted attribute values is $g_v = n - \left\lceil \sqrt{2(C_{max}-v)+0.25}-0.5 \right\rceil$. □

Secondly, the starting value (s_v) of the group containing the attribute value is derived, related to the given g_v of the attribute value.

$$\begin{aligned}
s_v &= \sum_{i=0}^{g_v-1}(n-i) \\
s_v &= \sum_{i=0}^{g_v-1} n - \sum_{i=0}^{g_v-1} i \\
s_v &= n\sum_{i=0}^{g_v-1} 1 - \sum_{i=0}^{g_v-1} i \\
s_v &= ng_v - \frac{g_v(g_v-1)}{2} \\
s_v &= \frac{2ng_v - g_v(g_v-1)}{2} \\
s_v &= \frac{g_v}{2}(2n-g_v+1)
\end{aligned}$$

The starting value in the group containing the attribute value can then be calculated by $s_v = \frac{g_v}{2}(2n-g_v+1)$. □

In the third element, the ending value (e_v) of the group containing the attribute value is accounted, related the the given g_v of the attribute value, as shown below.

$$\begin{aligned}
e_v &= \left(\sum_{i=0}^{g_v}(n-i)\right) - 1 \\
e_v &= \left(\sum_{i=0}^{g_v} n - \sum_{i=0}^{g_v} i\right) - 1
\end{aligned}$$

$$
\begin{aligned}
e_v &= \left(n \sum_{i=0}^{g_v} 1 - \sum_{i=0}^{g_v} i\right) - 1 \\
e_v &= \left(n(g_v + 1) - \frac{g_v(g_v + 1)}{2}\right) - 1 \\
e_v &= \left(\frac{2n(g_v + 1) - g_v(g_v + 1)}{2}\right) - 1 \\
e_v &= \left[\left(\frac{g_v + 1}{2}\right)(2n - g_v)\right] - 1
\end{aligned}
$$

Obviously, the ending value in the group containing the attribute value can be calculated by $e_v = \left[\left(\frac{g_v+1}{2}\right)(2n - g_v)\right] - 1$. □

Lastly, the sequence of the attribute value (i.e., the level of the attribute value) indicates the distance between the attribute value (v) and the starting value s_v in the group containing the attribute value (g_v). Therefore, the sequence of the attribute value can be calculated by $l_v = g_v + (v - s_v)$ □

4.2 Bitmap index creation for HyBiX bitmap index

Table 4.1 describes the notation used in bitmap index creation algorithm for the HyBiX bitmap index.

Table 4.1 Notation used in creation algorithm for 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, where $i = 0, 1, \ldots, n-1$
v	A distinct value of indexed attribute
g_v	A group containing the value v, $g_v = n - \left\lceil \sqrt{2(C_{max} - v) + 0.25} - 0.5 \right\rceil$
s_v	Starting value in the group that contains the value v,$s_v = \left[(\frac{g_v}{2})(2n - g_v + 1)\right]$
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 4.1 shows the creation algorithm of HyBiX bitmap index in 6 steps. Each value of the indexed attribute A is assigned by a number in an in-

Algorithm 4.1 The creation of 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 $n = \left\lceil \sqrt{2C + 0.25} - 0.5 \right\rceil$
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 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**

creasing sequence (i.e., 0, 1, ..., $C-1$). In the 2nd step, 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 redundant computation, to facilitate creating index for HyBiX bitmap index, and to assist in the queries. The HyBiX assistant table contains five elements: v, g_v, s_v, e_v, and l_v, characterizing the attribute values. However, the HyBiX assistant table is option to create because each element can be calculated by its individual equation. Consequently, from the utilities of the HyBiX assistant table, this algorithm creates the HyBiX assistant table to provide an efficiency of computation times in both processes of index creation and query processing. Next, 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 HyBiX bitmap index can be written in Eq. (4.4).

$$H^i = \begin{cases} 1 & g_v \leq i \leq l_v \\ 0 & \text{Otherwise.} \end{cases} \tag{4.4}$$

Suppose we want to create the HyBiX bitmap index for attribute A with C = 15. The HyBiX bitmap index has 5 bitmap vectors (n), $\{H^0, H^1, H^2, H^3, H^4\}$, which

is enough to represent all 15 distinct values. Figure 4.3b illustrates the accomplishment of the HyBiX bitmap index for attribute A with the given HyBiX assistant table in Figure 4.3a. For example, to encode attribute value '3', the values of g and l for value '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 4.3b.

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

(a) HyBiX assistant table

	A	H^0	H^1	H^2	H^3	H^4
1	3	1	1	1	1	0
2	9	0	0	1	0	0
3	14	0	0	0	0	1
4	8	0	1	1	1	1
5	10	0	0	1	1	0
6	3	1	1	1	1	0
7	4	1	1	1	1	1
8	0	1	0	0	0	0
9	12	0	0	0	1	0
10	5	0	1	0	0	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
100,000	2	1	1	1	0	0

(b) Table T and HyBiX bitmap index

Figure 4.3 HyBiX assistant table, and an example of the HyBiX bitmap index for attribute A with $C = 15$.

4.3 Query processing for HyBiX bitmap index

Normally, a query of the form $v_1 \leq A \leq v_2$ 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 $0 \leq v_1 < v_2 \leq C - 1$. For HyBiX bitmap index, the query is an equality query when $g_{v_1} = g_{v_2}$ and the query is a range query when either $g_{v_1} = g_{v_2}$ or $g_{v_1} < g_{v_2}$.

Equality query processing: Normally, the form of an equality query is '$A = v$' where $v = v_1 = v_2$. The concept for answering the equality query is that the bitmap vector related to the group containing v_1 (or v_2) and the bitmap vector related to the level of v_1 (or v_2) are identified. To evaluate an equality query, the retrieval function for an equality query is created by Eq. (4.5).

$$A = v = P \wedge Q \tag{4.5}$$

where

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

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

For an equality query, the function P is responsible for identifying the relevant bitmap vector belonging the group containing value v. The bitmap vector H^{g_v} is accessed if $g_v = 0$; otherwise the negation of bitmap vector H^{g_v-1} is needed to perform the bitwise-AND operation with the bitmap vector H^{g_v} if $g_v \neq 0$. Moreover, the function Q is responsible for identifying the relevant bitmap vector belonging the level of value v.

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. (4.5), the retrieval function of this query is $H^0 \wedge (H^3 \oplus H^4)$. Figure 4.4 shows the result of this equality query for the attribute A in Table T.

	A	H^0		H^3		H^4		$H^0 \wedge (H^3 \oplus H^4)$
1	3	1		1		0		1
2	9	0		0		0		0
3	14	0		0		1		0
4	8	0		1		1		0
5	10	0		1		0		0
6	3	1		1		0		1
7	4	1		1		1		0
8	0	1	$\wedge$	0	$\oplus$	0	$=$	0
9	12	0		1		0		0
10	5	0		0		0		0
.	.	.		.		.		.
.	.	.		.		.		.
.	.	.		.		.		.
100,000	2	1		0		0		0

Figure 4.4 The result of the equality query $A = 3$.

Example 2 To evaluate the equality query $A = 8$, the information of '8' consists of $g_8 = 1$, $s_8 = 5$, $e_8 = 8$, and $l_8 = 4$ given by the HyBiX assistant table, which the value of g_8 is not equal to 0. Using Eq. (4.5), the retrieval function of this query is

$\overline{H^0} \wedge H^1 \wedge H^4$. Figure 4.5 shows the result of this equality query for the attribute A in Table T.

	A	$\overline{H^0}$		H^1		H^4		$\overline{H^0} \wedge H^1 \wedge H^4$
1	3	0		1		0		0
2	9	1		0		0		0
3	14	1		0		1		0
4	8	1		1		1		1
5	10	1		0		0		0
6	3	0		1		0		0
7	4	0	$\wedge$	1	$\wedge$	1	$=$	0
8	0	0		0		0		0
9	12	1		0		0		0
10	5	1		1		0		0
.	.	.		.		.		.
.	.	.		.		.		.
.	.	.		.		.		.
100,000	2	0		0		0		0

Figure 4.5 The result of the equality query $A = 8$.

Range query processing: A range query can be divided into two cases: either $g_{v_1} = g_{v_2}$ or $g_{v_1} < g_{v_2}$, as aforementioned. Clearly, the value of g_{v_1} is equal to the value of g_{v_2} if v_1 and v_2 are placed in the same group. Otherwise, the value of g_{v_1} is less than the value of g_{v_2} if v_1 and v_2 are place in the different group. In the first case, the bitmap vectors related to group containing value v_1 are identified, and then the bitmap vectors related to level of value v_1 (and v_2, if necessary) are identified. In order to evaluate a range query when $g_{v_1} = g_{v_2}$, the retrieval function can be created by Eq. (4.6). For $g_{v_1} = g_{v_2}$,

$$v_1 \leq A \leq v_2 = \begin{cases} P & v_1 = s_{v_1}, v_2 = e_{v_2} \\ P \wedge Q & \text{Otherwise} \end{cases} \tag{4.6}$$

Similarly, the function P is used for identifying the bitmap vectors belonging the group containing value v_1 while the function Q is used for identifying the bitmap vectors belonging the level of v_1 and v_2. When all values in the group are queried, only function P is executed. Otherwise, the bitwise-AND operator is used over the result of P with the result of Q when some values are queried.

Example 3 To evaluate the range query in the form of $1 \leq A \leq 4$, the information of '1' consists of $g_1 = 0$, $s_1 = 0$, $e_1 = 4$, and $l_1 = 1$ while the information of

'4' consists of $g_4 = 0$, $s_4 = 0$, $e_4 = 4$, and $l_4 = 4$. As the information obtained, the value of v_1 is not equal to s_{v_1} and the value of v_2 is equal to e_{v_2}. Using Eq. (4.6), the retrieval function of this query is $H^0 \wedge H^1$, which only one bitmap vector accessed is required. Figure 4.6 shows the result of this range query for the attribute A in Table T.

	A	H^0		H^0		$H^0 \wedge H^1$
1	3	1		1		1
2	9	0		1		0
3	14	0		0		0
4	8	0		1		0
5	10	0		1		0
6	3	1		1		1
7	4	1	$\wedge$	1	$=$	1
8	0	1		0		0
9	12	0		0		0
10	5	0		0		0
.	.	.		.		.
.	.	.		.		.
.	.	.		.		.
100,000	2	1		1		1

Figure 4.6 The result of the range query $1 \leq A \leq 4$.

Example 4 To evaluate the range query in the form of $6 \leq A \leq 8$, the information of '6' consists of $g_6 = 1$, $s_6 = 5$, $e_6 = 8$, and $l_6 = 2$ while the information of '8' consists of $g_8 = 1$, $s_8 = 5$, $e_8 = 8$, and $l_8 = 4$ given by the HyBiX assistant table. Both values are placed in the same group and the value of their group is not equal to 0 (i.e., $g_{v_1} \neq 0$). Moreover, the value of v_1 is not equal to s_{v_1} but the value of v_2 is equal to e_{v_2} (i.e., $v_1 \neq s_{v_1}, v_2 = e_{v_2}$). Using Eq. (4.6), the retrieval function of this query is $\overline{H^0} \wedge H^1 \wedge H^2$. Figure 4.7 shows the result of this range query for the attribute A in Table T.

Next, the query values cover in many groups of HyBiX bitmap index if $g_{v_1} < g_{v_2}$. It is guaranteed that $g_{v_2} > 0$. There are three parts separately considered. The bitmap vectors related to the group containing values v_1 and v_2 are identified, respectively. Next, the bitmap vectors related to the groups containing relevant query values, except for groups containing v_1 and v_2, are identified. Then, the bitwise-OR operations are used among those three parts to find the final result. In order to answer a range query when $g_{v_1} < g_{v_2}$, the retrieval function is given in Eq. (4.7).

	A		$\overline{H^0}$		H^1		H^2		$\overline{H^0} \wedge H^1 \wedge H^2$
1	3		0		1		0		0
2	9		1		0		1		0
3	14		1		0		0		0
4	8		1		1		1		1
5	10		1		0		1		0
6	3		0		1		1		0
7	4		0	$\wedge$	1	$\wedge$	1	$=$	0
8	0		0		0		0		0
9	12		1		0		0		0
10	5		1		1		0		0
.	.		.		.		.		.
.	.		.		.		.		.
.	.		.		.		.		.
100,000	2		0		0		1		0

Figure 4.7 The result of the range query $6 \leq A \leq 8$.

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

$$v_1 \leq A \leq v_2 = T \vee \begin{cases} RV1 & g_{v_1} = 0 \\ \overline{H^{g_{v_1}-1}} \wedge RV1 & \text{Otherwise} \end{cases} \tag{4.7}$$

where

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

$$RV1 = \begin{cases} H^{g_{v_1}} & v_1 = s_{v_1} \\ H^{g_{v_1}} \wedge H^{l_{v_1}} & v_1 \neq s_{v_1} \end{cases}$$

$$RV2 = \begin{cases} H^{g_{v_2}} & v_2 = e_{v_2} \\ H^{g_{v_2}} \wedge \overline{H^{l_{v_2}+1}} & v_2 \neq e_{v_2} \end{cases}$$

The query values in the group between g_{v_1} and g_{v_2} are retrieved by the function T. The function $RV1$ is used for specifying the relevant query values which are the same group as v_1 while $RV2$ is also used for specifying the relevant values which are in the same group as v_2.

Example 5 To evaluate the range query in the form of $3 \leq A \leq 13$, the information of '3' consists of $g_3 = 0$, $s_3 = 0$, $e_3 = 4$, and $l_3 = 3$ while the information of '13' consists of $g_{13} = 3$, $s_{13} = 12$, $e_{13} = 13$, and $l_{13} = 4$ given by the HyBiX

assistant table. The value of v_1 is not equal to s_{v_1} but the value of v_2 is equal to e_{v_2} (i.e., $v_1 \neq s_{v_1}, v_2 = e_{v_2}$). Moreover, the different of g_3 and g_{13} is 3 (i.e., $g_{v_2} - g_{v_1} \geq 2$). Using Eq. (4.7), the retrieval function of this query is $\left(\overline{H^0} \wedge (H^1 \vee H^2 \vee H^3)\right) \vee (H^0 \wedge H^3)$. Figure 4.8 shows the result of this range query for the attribute A in Table T.

	A	$\overline{H^0}\wedge(H^1\vee H^2\vee H^3)$		$H^0\wedge H^3$		$\left(\overline{H^0} \wedge (H^1 \vee H^2 \vee H^3)\right)\vee(H^0\wedge H^3)$
1	3	0		1		1
2	9	1		0		1
3	14	0		0		0
4	8	1		0		1
5	10	1		0		1
6	3	0		1		1
7	4	0	$\vee$	1	$=$	1
8	0	0		0		0
9	12	1		0		1
10	5	1		0		1
.	.	.		.		.
.	.	.		.		.
.	.	.		.		.
100,000	2	0		0		0

Figure 4.8 The result of the range query $3 \leq A \leq 13$.

Example 6 To evaluate the range query in the form of $6 \leq A \leq 10$, the information of '6' consists of $g_6 = 1$, $s_6 = 5$, $e_6 = 8$, and $l_6 = 2$ while the information of '10' consists of $g_{10} = 2$, $s_{10} = 0$, $e_{10} = 11$, and $l_{10} = 3$ given by the HyBiX assistant table. In this example, the value of v_1 and v_2 is not equal to s_{v_1} and e_{v_2}, respectively, (i.e., $v_1 \neq s_{v_1}, v_2 \neq e_{v_2}$). Using Eq. (4.7), the retrieval function of this query is $\left(\overline{H^1} \wedge H^2 \wedge \overline{H^4}\right) \vee \left(\overline{H^0} \wedge H^1 \wedge H^2\right)$. Figure 4.9 shows the result of this range query for the attribute A in Table T.

The query processing algorithm for HyBiX bitmap index is described in Algorithm 4.2. The algorithm consists of 19 steps which facilitate answering both equality and range queries. It is better if the types of the query submitted can be identified whether it is an equality query or a range query by comparing the value of v_1 and v_2. An equality query is executed at the 1st - 3rd step. In the 2nd step, all information of v_1 is retrieved from the HyBiX assistant table. Next, the complete retrieval function for the query is created by using the bitwise-AND operation between the results of P and Q in the 3rd step. A range query is executed in the 4th - 19th step. In the 5th step, all information of v_1 and v_2 is retrieved from the HyBiX assistant table. The query values

	A	$\overline{H^1} \wedge H^2 \wedge \overline{H^4}$		$\overline{H^0} \wedge H^1 \wedge H^2$		$(\overline{H^1} \wedge H^2 \wedge \overline{H^4}) \vee (\overline{H^0} \wedge H^1 \wedge H^2)$
1	3	0		0		0
2	9	1		0		1
3	14	0		0		0
4	8	0		1		1
5	10	1		0		1
6	3	0		0		0
7	4	0	$\vee$	0	$=$	0
8	0	0		0		0
9	12	0		0		0
10	5	0		0		0
.	.	.		.		.
.	.	.		.		.
.	.	.		.		.
100,000	2	0		0		0

Figure 4.9 The result of the range query $6 \leq A \leq 10$.

which are in the same group (i.e., $g_{v_1} = g_{v_2}$) are executed in the 6^{th} - 11^{th} step. Otherwise, the 12^{th} - 18^{th} step is executed if the query values fall in many groups.

Algorithm 4.2 The query processing of HyBiX bitmap index

INPUT: Query values: v_1 and v_2
OUTPUT: A retrieval function (RF)

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**

This chapter described the basic concept of HyBiX bitmap index, which utilizes the idea of grouping attribute values and the encoding designs of existing bitmap

indexes. Moreover, the bitmap creation algorithm for HyBiX bitmap index and its query processing for equality and range queries are also given.

The next chapter shows the experimental analysis of seven encoding bitmap indexes in point of views space requirement, execution time with equality and range queries, and space and time trade-off.

CHAPTER 5

Performance Study

This chapter presents a theoretical analysis and experimental results on which space-efficiency, time-efficiency, and the trade-off between space and execution time for equality and range queries, are compared for seven encoding bitmap indexes.

5.1 Theoretical analysis

Table 5.1 shows a theoretical analysis of seven bitmap indexes comparing space requirements, numbers of bitmap vectors to be scanned and numbers of Boolean operations for equality and range queries. The Basic bitmap index is the worst in space usage (uses C bitmap vectors) while the Encoded bitmap index is the best space usage (uses $\lceil \log_2 C \rceil$ bitmap vectors). The Range bitmap index uses $C - 1$ bitmap vectors, while the Interval bitmap index decreases this by about half to $\lceil \frac{C}{2} \rceil$ bitmap vectors. The Scatter and Dual bitmap indexes use $\lceil 2\sqrt{C} \rceil$, and $\lceil \sqrt{2C + 0.25} + 0.5 \rceil$ bitmap vectors, respectively. The HyBiX bitmap index uses $\lceil \sqrt{2C + 0.25} - 0.5 \rceil$ bitmap vectors.

In order to answer equality queries, the Basic bitmap index scans 1 bitmap vector without any Boolean operation. The scans of the Range bitmap index range from 1 to 2 bitmap vectors with 0 to 1 Boolean operations. The Interval bitmap index scans 2 bitmap vectors with 1 to 2 Boolean operations. The Encoded bitmap index scans $\lceil \log_2 C \rceil$ bitmap vectors with $2\lceil \log_2 C \rceil$ Boolean operations. Both the Scatter, and Dual bitmap indexes scan 2 bitmap vectors with 1 Boolean operation. The scans of the HyBiX bitmap index range from 2 to 4 bitmap vectors with 1 to 4 Boolean operations.

In order to answer range queries, the Basic bitmap index scans the most related bitmap vectors and performs the most Boolean operations. The scans of the Range bitmap index range from 1 to 2 bitmap vectors with 0 or 1 Boolean operations. The Interval bitmap index scans 2 bitmap vectors and performs 2 Boolean operations. The Encoded, Scatter, and Dual bitmap indexes utilize the Boolean simplification to reduce the complexity of retrieval functions. Therefore, the number of bitmap vectors scanned

for Encoded bitmap index ranges from 1 to $(v_2 - v_1 + 1) \times (\lceil \log_2 C \rceil)$ bitmap vectors and the number of Boolean operations ranges from 0 to $(v_2 - v_1 + 1) \times (\lceil \log_2 C \rceil + 1)$. Both the Scatter and Dual bitmap indexes scan from 3 to $2(v_2 - v_1 + 1)$ bitmap vectors and perform from 2 to $2(v_2 - v_1 + 1)$ Boolean operations. The number of scanned bitmap vectors with HyBiX bitmap index ranges from 1 to $g_{v_2} - g_{v_1} + 4$ and the number of Boolean operations used ranges from 0 to $g_{v_2} - g_{v_1} + 5$, where $0 \leq g_{v_1} \leq g_{v_2} \leq \left\lceil \sqrt{2C + 0.25} - 0.5 \right\rceil - 1$.

5.2 Experimental results

5.2.1 Data set used and experimental setting

The experiment was 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 data set, which is retrieved from [62]. This benchmark is composed of eight separate tables. The benchmark data are generated along with a scale factor (SF), which specifies the size of data. We experimented on table LINEITEM with four different scale factors: 25, 50, 75, and 100, which contains over 150, 300, 450, and 600 million rows, respectively. We selected two representative attributes having different cardinalities: a small data set (Quantity attribute with cardinality 50) and a large data set (Shipdate attribute with cardinality 2,526).

The space-efficiency of an encoding bitmap index is measured in terms of space requirements for storing all its bitmap vectors. The time-efficiency of an encoding bitmap index is measured in terms of average query execution time over all 10 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 (including Boolean simplifications with range queries used by Encoded, Scatter, and Dual bitmap indexes). The Boolean simplification requires some execution times, but does not significantly impact the overall query execution time. The space and time trade-off 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.

Table 5.1 A comparative study of seven encoding bitmap index algorithms

Bitmap index	Space requirement	Execution time for equality queries		Execution time for range queries	
	Number of bitmap vectors created	Number of bitmap vectors scanned	Number of Boolean operations used	Number of bitmap vectors scanned	Number of Boolean operations used
Basic	C	1	0	$v_2 - v_1 + 1$	$v_2 - v_1$
Range	$C - 1$	1 to 2	0 to 1	1 to 2	0 to 1
Interval	$\lceil \frac{C}{2} \rceil$	2	1 to 2	2	2
Encoded	$\lceil \log_2 C \rceil$	$\lceil \log_2 C \rceil$	$2\lceil \log_2 C \rceil$	1 to $(v_2 - v_1 + 1)(\lceil \log_2 C \rceil)$	0 to $(v_2 - v_1 + 1)(\lceil \log_2 C \rceil + 1)$
Scatter	$\lceil 2\sqrt{C} \rceil$	2	1	3 to $2(v_2 - v_1 + 1)$	2 to $2(v_2 - v_1 + 1)$
Dual	$\lceil \sqrt{2C + 0.25} + 0.5 \rceil$	2	1	3 to $2(v_2 - v_1 + 1)$	2 to $2(v_2 - v_1 + 1)$
HyBiX	$\lceil \sqrt{2C + 0.25} - 0.5 \rceil$	2 to 4	1 to 4	1 to $(g_{v_2} - g_{v_1} + 4)$	0 to $(g_{v_2} - g_{v_1} + 5)$

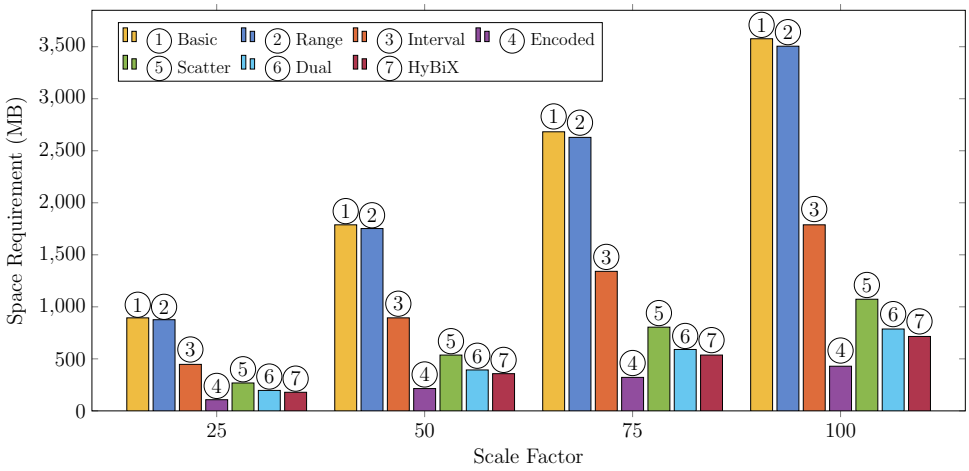
C is cardinality of the indexed attribute.

g_{v_1} is the group in HyBiX bitmap index containing value v_1.

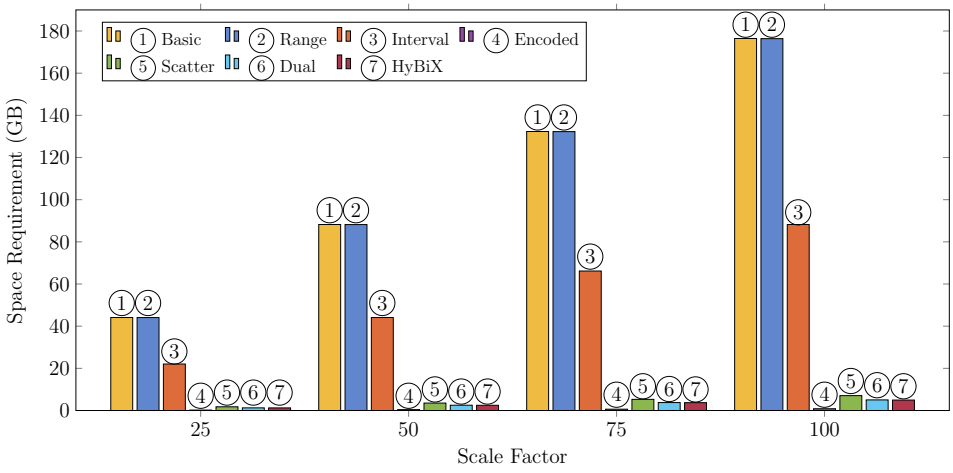
g_{v_2} is the group in HyBiX bitmap index containing value v_2.

5.2.2 The space-efficiency of seven encoding bitmap indexes

Figure 5.1 affirms the above theoretical analysis of space requirements for the seven encoding bitmap indexes, for the two selected attributes. The Basic bitmap index requires the most space while the Encoded bitmap index requires the least, for both Quantity and Shipdate attributes. The HyBiX requires less space than the other remaining bitmap indexes for both attributes. As the cardinality increases, the size of Basic, Range, and Interval bitmap indexes exceed the size of 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.



(a) Quantity attribute



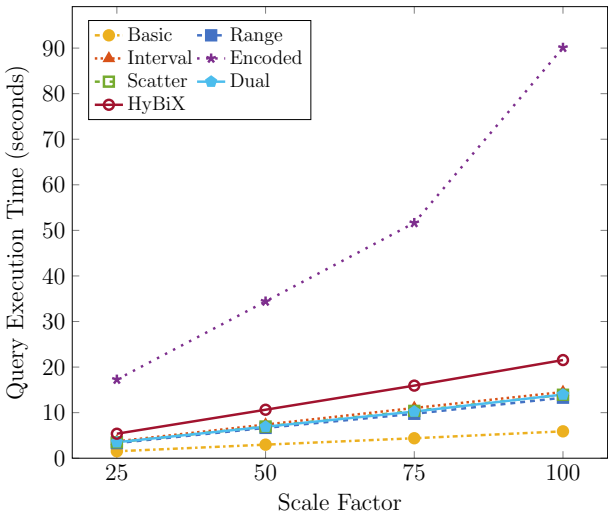
(b) Shipdate attribute

Figure 5.1 The space requirement of seven alternative encoding bitmap indexes.

5.2.3 The time-efficiency of seven encoding bitmap indexes

Equality queries

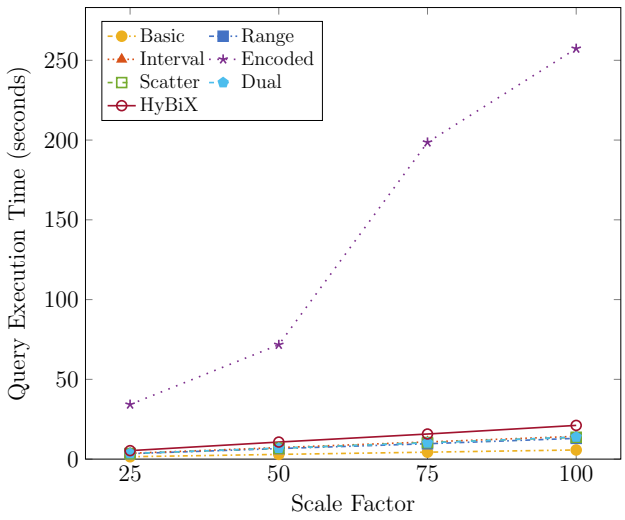
Figure 5.2 illustrates the query execution times of the seven encoding bitmap indexes with equality queries on Quantity and Shipdate attributes. To answer equality queries, the Basic bitmap index uses 1 bitmap vectors with no Boolean operations. The Range, Interval, Scatter, and Dual bitmap indexes use 2 bitmap vectors and 1 Boolean operation for the Range, Scatter, and Dual bitmap indexes, and 2 Boolean operations for the Interval bitmap index. The Encoded bitmap index uses $\lceil \log_2 C \rceil$ bitmap vectors and



Query execution time

Bitmap index	Scale factor			
	25	50	75	100
Basic	1.52350	2.97570	4.38406	5.89897
Range	3.31602	6.65942	9.75298	13.28298
Interval	3.70201	7.39024	11.02924	14.53051
Encoded	17.26202	34.40758	51.63899	90.08717
Scatter	3.51548	6.96173	10.38928	13.93256
Dual	3.51088	6.91374	10.23746	13.97350
HyBiX	5.37558	10.63795	15.93171	21.54041

(a) Quantity attribute



Query execution time

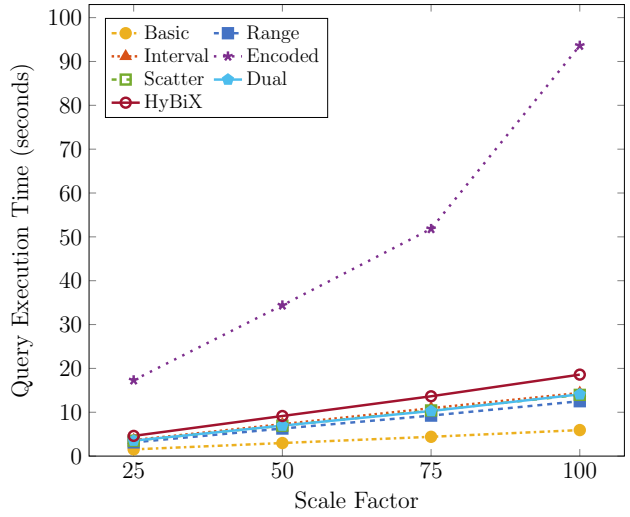
Bitmap index	Scale factor			
	25	50	75	100
Basic	1.49902	2.95526	4.34581	5.73083
Range	3.40169	6.59404	9.63845	13.02037
Interval	3.65312	7.21872	10.77571	14.27777
Encoded	34.21895	71.69528	135.59237	257.33834
Scatter	3.49090	6.83654	10.40266	13.68498
Dual	3.53580	6.85803	10.18445	13.88376
HyBiX	5.30256	10.69024	15.73132	21.13676

(b) Shipdate attribute

Figure 5.2 The query execution time of seven encoding bitmap indexes for equality queries: A query value falls in any group of HyBiX bitmap index.

$2\lceil \log_2 C \rceil$ Boolean operations. The HyBiX bitmap index uses 4 bitmap vectors and at most 4 Boolean operations. Furthermore, the query execution time with Encoded bitmap index significantly increases with cardinality of the indexed attribute. On the other hand, the cardinality does not affect the overall query execution time with the other encoding bitmap indexes, for equality queries.

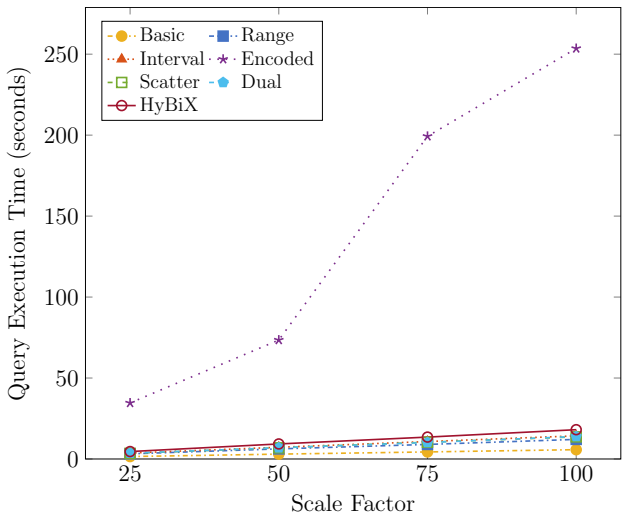
Furthermore, the HyBiX bitmap index scans at most 3 bitmap vectors and uses at most 2 Boolean operations when the query value falls in group 0 of the HyBiX bitmap index. Consequently, the query execution time with equality queries used by the



Query execution time

Bitmap index	Scale factor			
	25	50	75	100
Basic	1.53019	2.98000	4.39870	5.93306
Range	3.13163	6.28674	9.23499	12.51907
Interval	3.64989	7.28978	10.92295	14.40356
Encoded	17.31165	34.39093	51.82539	93.62092
Scatter	3.51284	6.94743	10.41126	13.96687
Dual	3.52583	6.94786	10.28392	14.08316
HyBiX	4.60724	9.14358	13.64289	18.58051

(a) Quantity attribute



Query execution time

Bitmap index	Scale factor			
	25	50	75	100
Basic	1.51019	2.99994	4.35410	5.75393
Range	3.17464	6.23674	8.98826	12.10525
Interval	3.67009	7.23935	10.77528	14.29833
Encoded	34.59703	73.45627	136.66326	253.51565
Scatter	3.49733	6.88701	10.35326	13.67484
Dual	3.58912	6.93893	10.17872	13.88270
HyBiX	4.58966	9.28651	13.51708	18.09189

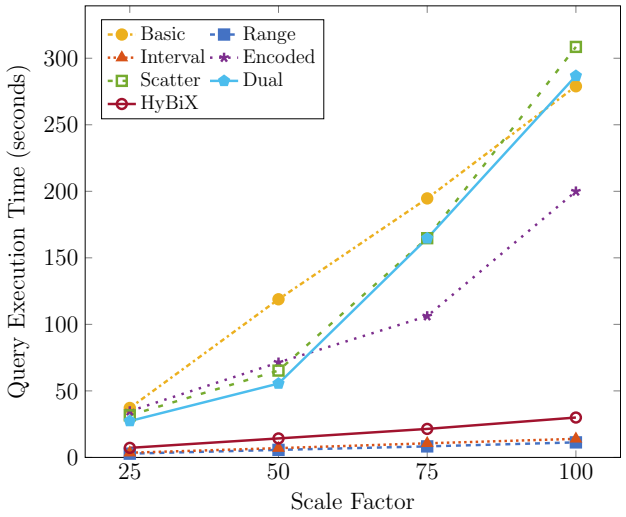
(b) Shipdate attribute

Figure 5.3 The query execution time of seven encoding bitmap indexes for equality queries: Query values fall in group 0 of HyBiX bitmap index.

HyBiX bitmap index is faster for both Quantity and Shipdate attributes, as shown in Figure 5.3.

Range queries

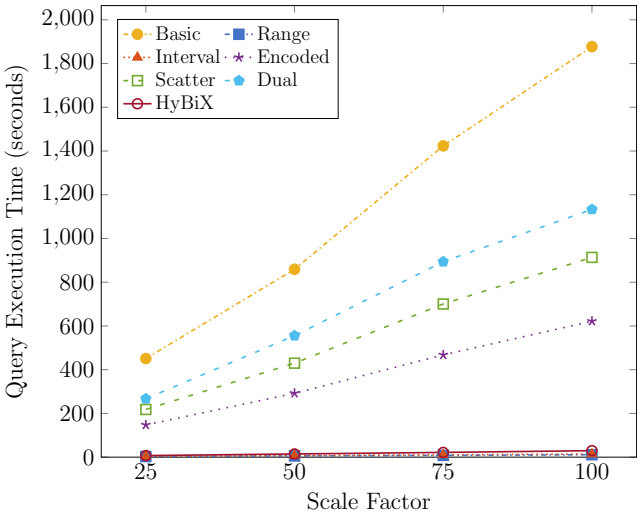
Figure 5.4 shows a comparison of the query execution times with seven encoding bitmap indexes for range queries on Quantity and Shipdate attributes. The Basic bitmap index had the slowest query execution time. The Range and Interval bitmap indexes use 2 bitmap vectors and 2 Boolean operations to answer range queries. The execution time with the Encoded, Scatter and Dual bitmap indexes is undesirable



Query execution time

Bitmap index	Scale factor			
	25	50	75	100
Basic	37.15093	118.85027	194.66048	278.94760
Range	2.83513	5.67528	8.29288	11.30590
Interval	3.53298	6.97236	10.60351	13.95199
Encoded	34.75150	71.36784	106.18017	199.94313
Scatter	31.64861	65.25394	164.77096	308.52223
Dual	27.34000	55.51722	164.65944	286.76945
HyBiX	7.18307	14.27602	21.42800	29.97768

(a) Quantity attribute



Query execution time

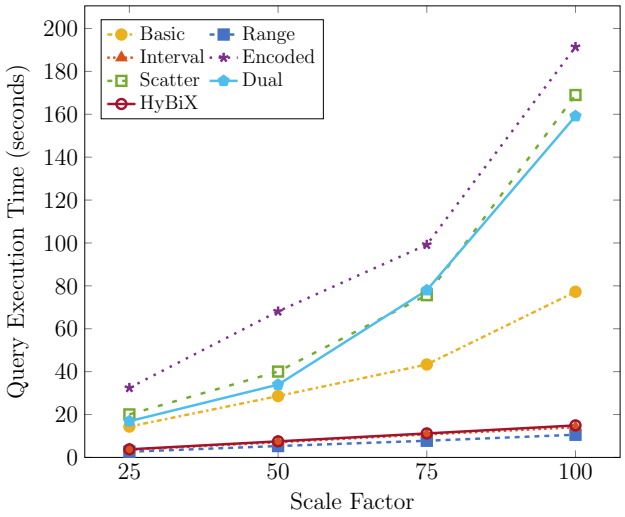
Bitmap index	Scale factor			
	25	50	75	100
Basic	450.99062	859.16574	1423.62817	1877.31356
Range	2.85536	5.55180	8.22076	11.13717
Interval	3.61514	7.09763	10.70304	14.21356
Encoded	147.40354	292.13895	467.57429	621.80406
Scatter	218.15120	429.82490	700.52176	913.66721
Dual	267.00535	555.46259	893.73769	1133.03459
HyBiX	7.21929	14.62329	21.71678	29.69490

(b) Shipdate attribute

Figure 5.4 The query execution time of seven encoding bitmap indexes for range queries: A query value falls in any groups of HyBiX bitmap index.

because of the high complexity of their retrieval functions. With increased cardinality, the range of query values is broadened. Then, the query execution times of the Basic, Encoded, Scatter, and Dual bitmap indexes increased exponentially. In addition, the query execution time of HyBiX bitmap index is extremely fast but slightly slower than with Range and Interval bitmap indexes.

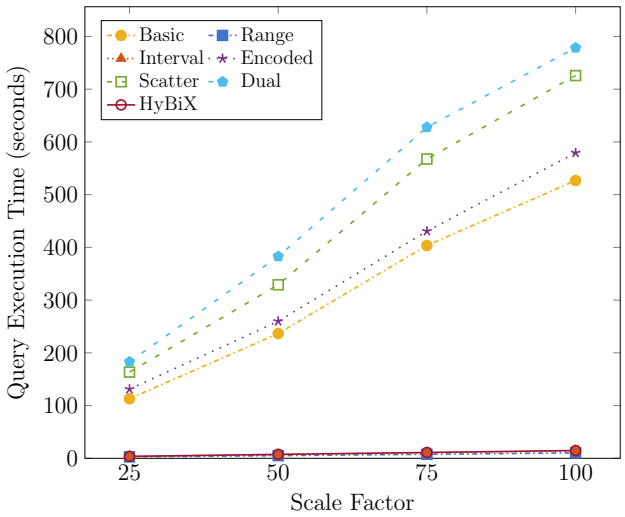
Furthermore, the HyBiX bitmap index uses at most 3 bitmap vectors scanned and at most 2 Boolean operations, when the query values fall in group 0 of the HyBiX bitmap index. Therefore, the query execution time used by the HyBiX index is



Query execution time

Bitmap index	Scale factor			
	25	50	75	100
Basic	14.39272	28.60017	43.33009	77.23544
Range	2.65342	5.30979	7.76708	10.56740
Interval	3.57124	7.06484	10.73904	14.04948
Encoded	32.44833	68.09462	99.17240	191.41579
Scatter	20.06376	40.02940	75.65696	168.98251
Dual	16.89297	33.92998	77.95817	159.14699
HyBiX	3.77014	7.49748	11.18407	14.92868

(a) Quantity attribute



Query execution time

Bitmap index	Scale factor			
	25	50	75	100
Basic	113.06728	236.60501	403.63629	526.77949
Range	2.64677	5.13098	7.60756	10.30360
Interval	3.57492	7.01377	10.57472	14.03874
Encoded	131.11841	259.80727	430.72772	579.23984
Scatter	163.51328	329.09298	567.56730	725.79919
Dual	183.00531	382.76829	627.80206	778.47931
HyBiX	3.68813	7.46313	11.04335	14.80563

(b) Shipdate attribute

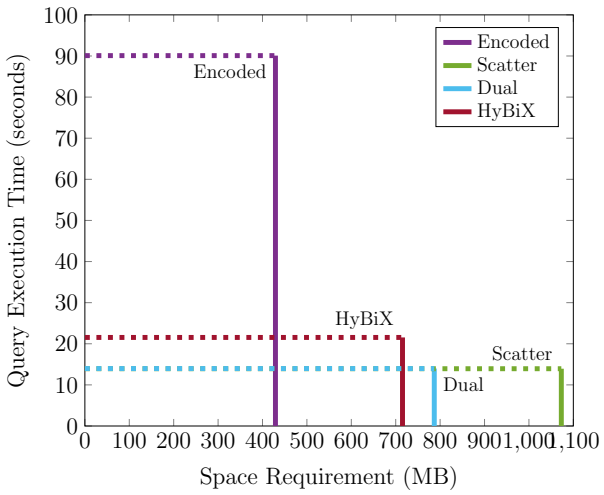
Figure 5.5 The query execution time of seven encoding bitmap indexes for range queries: Query values fall in group 0 of HyBiX bitmap index.

improved for both Quantity and Shipdate attributes, as shown in Figure 5.5.

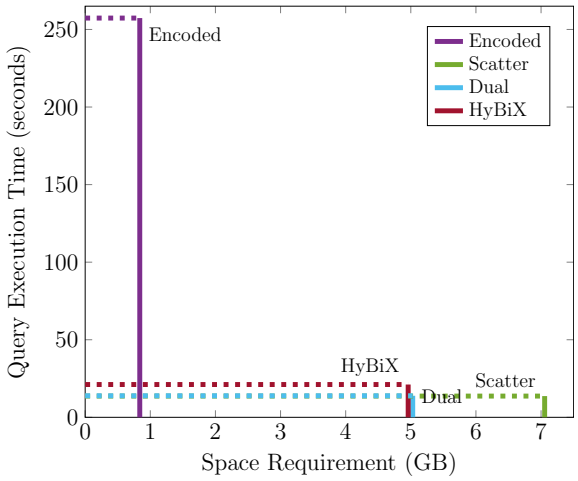
5.2.4 The trade-off between space and time of four encoding bitmap indexes

Equality query

Since the space requirements used by the Basic, Range and Interval bitmap indexes are undesirable, which are exactly an impact on the overall performance in terms of space and time trade-off. Therefore, we present the result only scale factor 100 as the result for any scale factors. Figure 5.6 depicts the trade-off between space



(a) Quantity attribute

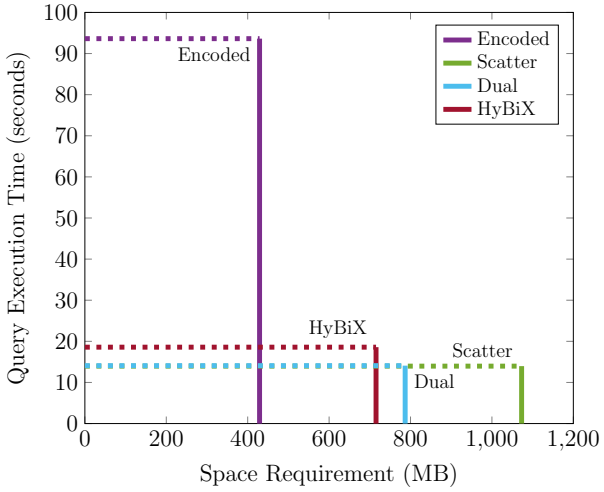


(b) Shipdate attribute

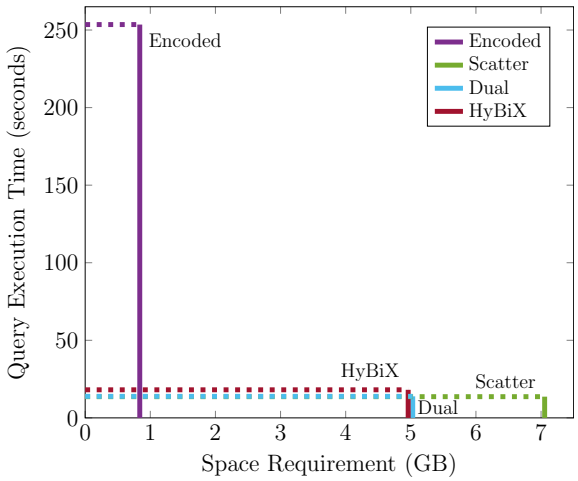
Figure 5.6 The space vs. time trade-off of four encoding bitmap indexes for equality queries in scale factor 100: A query value falls in any group of HyBiX bitmap index.

and time for the four alternative encoding bitmap indexes (Encoded, Scatter, Dual, and HyBiX) for equality queries on Quantity and Shipdate attributes, respectively, with the query value in any group of HyBiX bitmap index. The Dual bitmap index achieves the best performance in terms of the trade-off for equality queries. The performance of HyBiX bitmap index is better than those of four alternative encoding bitmap indexes (Basic, Range, Interval, and Encoded bitmap indexes), but slightly poorer than with Scatter bitmap index.

However, when a query value falls in group 0 of HyBiX bitmap index, the scan of 3 bitmap vectors and 2 Boolean operations are needed to answer the query.



(a) Quantity attribute



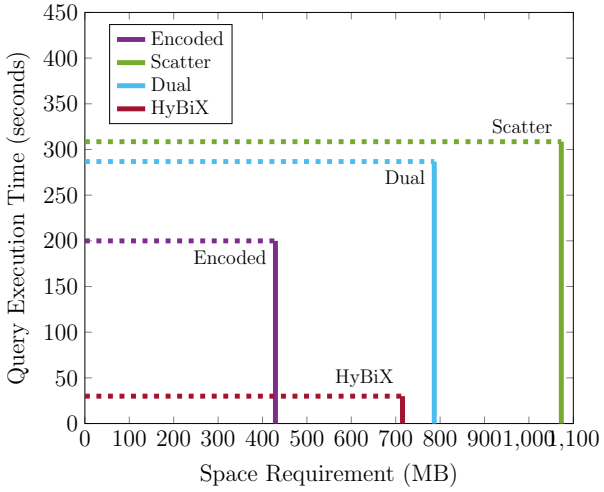
(b) Shipdate attribute

Figure 5.7 The space vs. time trade-off of four encoding bitmap indexes for equality queries in scale factor 100: A query value falls in group 0 of HyBiX bitmap index.

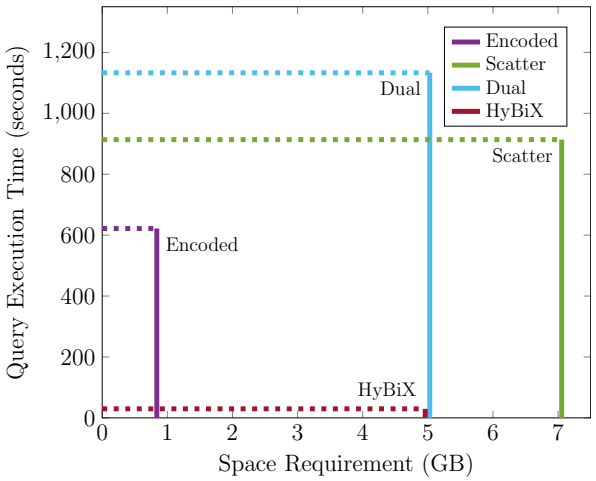
Then, the performance of HyBiX bitmap index is better than that of the Scatter bitmap index for both attributes, shown in Figure 5.7.

Range query

Figure 5.8 shows space and time trade-off with the four encoding bitmap indexes for range queries on Quantity and Shipdate attributes, when the query values fall in any groups of HyBiX bitmap index. The number of bitmap vectors scanned used by the HyBiX bitmap index is less than that used by other bitmap indexes, which directly impacts the execution time consumed. This results the query performance with the HyBiX bitmap index outperforms the other bitmap indexes, for both attributes.



(a) Quantity attribute

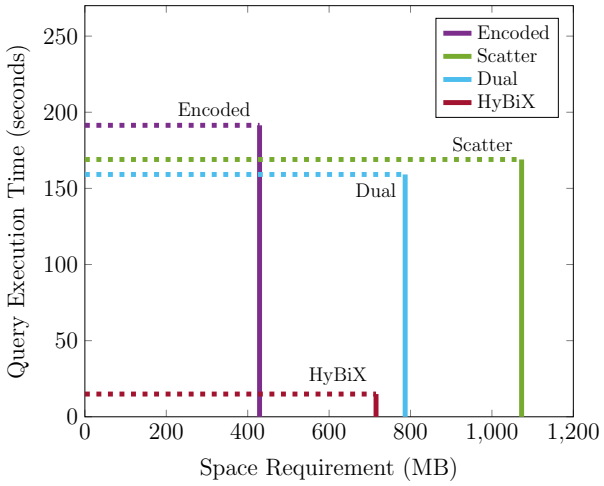


(b) Shipdate attribute

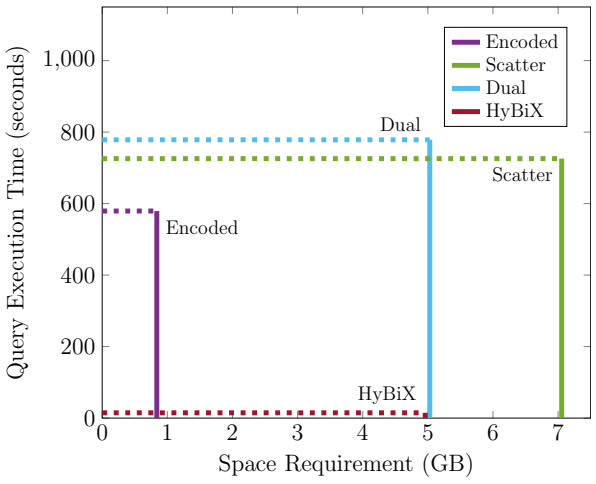
Figure 5.8 The space vs. time trade-off of four encoding bitmap indexes for range queries in scale factor 100: Query values fall in any groups of HyBiX bitmap index.

Due to the execution time used by the HyBiX bitmap index is improved when the query values fall in group 0 of HyBiX biitmap index, the performance with HyBiX bitmap index is then improved in terms of space and time trade-off, as shown in Figure 5.9. Consequently, the HyBiX bitmap index outperforms the other encoding bitmap indexes for range queries regarding the space and time trade-off, with both attributes.

Clearly, the execution time used by the HyBiX bitmap index is improved when the query values fall in group 0, for both equality and range queries, which is an impact on the improved performance with the HyBiX bitmap index in terms of space



(a) Quantity attribute



(b) Shipdate attribute

Figure 5.9 The space vs. time trade-off of four encoding bitmap indexes for range queries in scale factor 100: Query values fall in group 0 of HyBiX bitmap index.

and time trade-off for both equality and range queries.

Advantage

The HyBiX bitmap index requires the less space than the Basic, Range, Interval, Scatter and Dual bitmap indexes, except the Encoded bitmap index. The query execution time with equality queries is considerably faster then the Encoded bitmap index. When the query value falls at group 0 of the HyBiX bitmap index, the query execution time with equality and range queries used by the HyBiX bitmap index is significantly improved, especially range queries. Therefore, the HyBiX bitmap index provides a good time-efficiency with both equality and range queries when the majority of submitted queries cannot be specified.

Limitations

The query execution time with equality queries used by the HyBiX bitmap index is slightly slower than that used by the Basic, Range, Interval, Scatter, and Dual bitmap index. Moreover, the HyBiX bitmap index requires many bitmap vectors accessed to answer range queries, when the query values cover many groups of the HyBiX bitmap index.

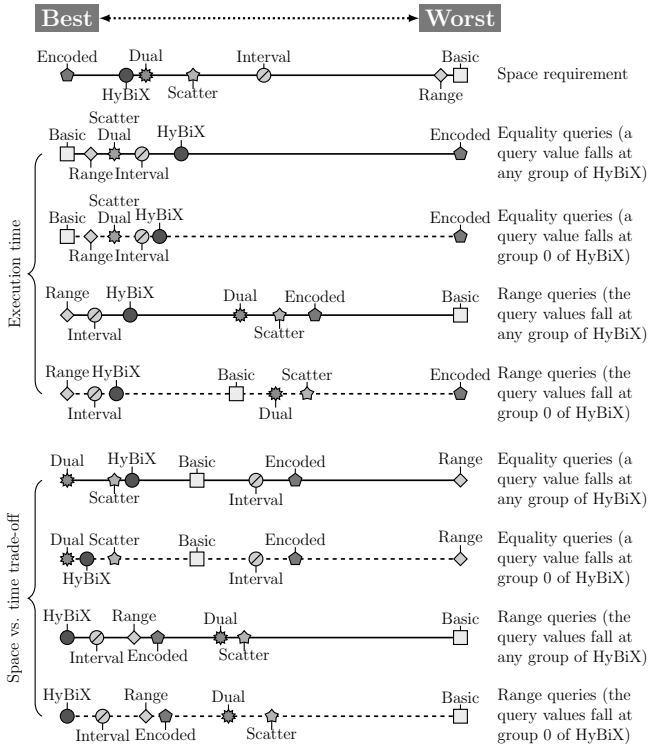
CHAPTER 6

Conclusion and Future Work

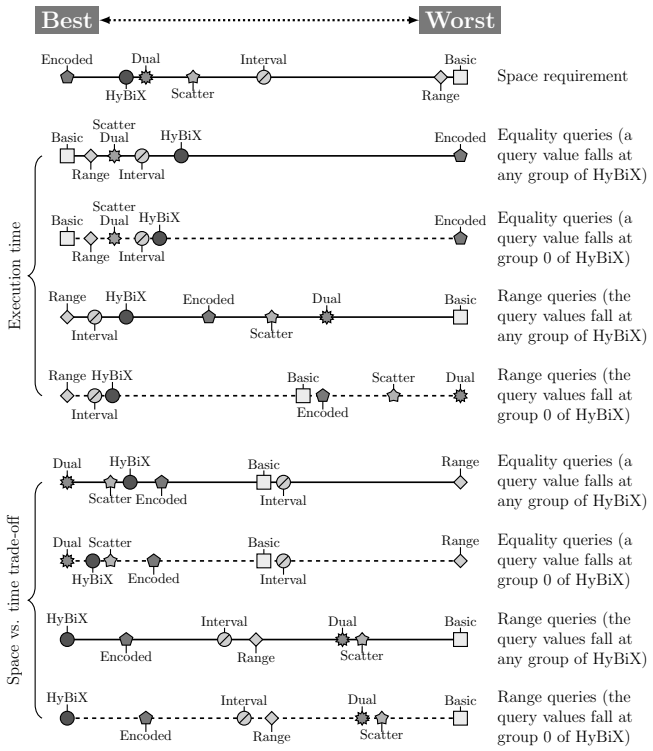
This dissertation introduced a new encoding bitmap index for providing an efficient space requirement and query execution time with equality and range queries over the attribute with high cardinalities. The encoding design of bitmap indexes plays an important role in effectively and efficiently both space requirements and query execution times. The HyBiX bitmap index utilize the idea of grouping attribute values and the encoding design of existing bitmap indexes.

6.1 Summary

Figures 6.1a and 6.1b show a comparison of performances for seven encoding bitmap indexes in terms of space-efficiency, time-efficiency, and space-time trade-off, for small and large datasets, respectively. Among the seven bitmap indexes, the Encoded bitmap index requires the least space, regardless of attribute cardinality. In addition, the performance of Encoded bitmap index in terms of space-time trade-off decreases for low cardinality attributes. Clearly, the Encoded bitmap index emphasizes space-efficiency rather than execution time with both equality and range queries. The Dual bitmap index requires small space and gives good execution times with equality queries, outperforming other bitmap indexes in space and time trade-off. However, the Dual bitmap index spends long query execution times with range queries. Therefore, the Dual bitmap index is most suitable with only equality queries. However, both equality and range queries are possibly submitted. The HyBiX bitmap index is able to reduce the space usage as well as answer both equality and range queries with good execution times for unpredictable queries. Furthermore, the HyBiX bitmap index is improved execution time with both equality and range queries when the query value falls in group 0 of HyBiX bitmap index for both a small and large data set. Therefore, the performance of HyBiX bitmap index is satisfactory in terms of space and time trade-off, especially with range queries.



(a) A small dataset



(b) A large dataset

Figure 6.1 Performances of seven encoding bitmap indexes for small and large datasets.

The explosion of data has emerged in recent years. The current era of big data poses challenges in managing those data and obtaining the benefits from querying it. Various encoding bitmap indexes have been introduced in order to reduce index size and efficiently speed up query processing. When various types of query submitted, 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 dissertation, a new encoding bitmap index was presented, called the HyBiX bitmap index. The comparative study shows that HyBiX bitmap index can reduce the space requirements with high cardinality of indexed attributes. Although the performance of HyBiX bitmap index is worse than that of the Dual bitmap index for equality queries, in terms of space and time trade-off, the query execution time with equality queries is still satisfactory. The HyBiX bitmap index outperforms the existing encoding bitmap indexes in terms of space and time trade-off with range queries. Furthermore, when the query values fall in group 0, the HyBiX bitmap index has improved query execution time and space and time trade-off, both with equality and range queries. To deal with big data, the Apache Hadoop is a well-known open source software framework based on Java language. The Apache Hadoop provides a distributed storage and a parallel processing on multiple computers with commodity hardware. Therefore, it is an opportunity for creating the HyBiX bitmap index in a distributed environment underlying the Apache Hadoop. Additionally, the advantage of parallel processing on the Apache Hadoop definitely allows the fast query execution time used by the HyBiX bitmap index over a big data.

6.2 Future work

In future work, the HyBiX bitmap index will be implemented in a distributed and parallel environment (e.g., Apache Hadoop). The performance of HyBiX bitmap index can be improved by applying data mining techniques to select frequent query values submitted, and placing the respective values in group 0 of HyBiX bitmap index. In addition, the combination of HyBiX and other existing encoding bitmap indexes is necessary to be comprehensively studied.

BIBLIOGRAPHY

[1] S. Chaudhuri, "What Next? A Half-Dozen Data Management Research Goals for Big Data and the Cloud," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems (PODS '12)*, Arizona, USA, 2012, pp. 1–4.

[2] S. Sagiroglu and D. Sinanc, "Big Data: A Review," in *2013 International Conference on Collaboration Technologies and Systems (CTS)*, California, USA, 2013, pp. 42–47.

[3] A. Katal, M. Wazid, and R. H. Goudar, "Big Data: Issues, Challenges, Tools and Good Practices," in *Proceedings of 2013 6th International Conference on Contemporary Computing (IC3)*, Noida, India, 2013, pp. 404–409.

[4] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in Big Data Analytics," *Journal of Parallel and Distributed Computing*, vol. 74, no. 7, pp. 2561–2573, Feb. 2014.

[5] J. Archenaa and E. A. Anita, "A Survey of Big Data Analytics in Healthcare and Government," *Procedia Computer Science*, vol. 50, pp. 408–413, May 2015.

[6] A. Ali, J. Qadir, R. ur Rasool, A. Sathiaseelan, and A. Zwitter, "Big Data For Development: Applications and Techniques," *Big Data Analytics*, vol. 1, pp. 1–24, Jul. 2016.

[7] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," *ACM SIGMOD Record*, vol. 26, no. 1, pp. 65–74, Mar. 1997.

[8] X. Wu, X. Zhu, G. Q. Wu, and W. Ding, "Data Mining with Big Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[9] K. Stockinger and K. Wu, "Bitmap Indices for Data Warehouses," in *Data Warehouses and OLAP Concepts Architectures and Solutions*. IRM Press, 2007, ch. 7, pp. 157–178.

[10] C. Y. Chan and Y. E. Ioannidis, "Bitmap Index Design and Evaluation," *ACM SIGMOD Record*, vol. 27, no. 2, pp. 355–366, Jun. 1998.

[11] W. Stallings, "Parallel Processing," in *Computer Organization and Architecture Designing for Performance*, 9th ed. Pearson Education, Inc., 2013, ch. 17, pp. 611–663.

[12] S. H. Chung, S. C. Oh, K. R. Ryu, and S. H. Park, "Parallel Information Retrieval on A Distributed Memory Multiprocessor System," in *Proceedings of 3rd International Conference on Algorithms and Architectures for Parallel Processing*, Victoria, Australia, 1997, pp. 163–176.

[13] J. Xiong, J. Wang, and J. Xu, "Research of Distributed Parallel Information Retrieval based on JPPF," in *Proceedings of 2010 International Conference of Information Science and Management Engineering (ISME 2010)*, Xi'an, China, 2010, pp. 109–111.

[14] L. Jiamin and F. Jun, "A Survey of MapReduce based Parallel Processing Technologies," *China Communications*, vol. 11, no. 14, pp. 146–155, 2014.

[15] W. Andrzejewski and R. Wrembel, "GPU-PLWAH: GPU-based Implementation of the PLWAH Algorithm for Compressing Bitmaps," *Control and Cybernetics*, vol. 40, no. 3, pp. 627–650, Jan. 2011.

[16] P. O'Neil and D. Quass, "Improved Query Performance with Variant Indexes," *ACM SIGMOD Record*, vol. 26, no. 2, pp. 38–49, Jun. 1997.

[17] K. Wu, E. J. Otoo, and A. Shoshani, "Optimizing Bitmap Indices with Efficient Compression," *ACM Transactions on Database Systems (TODS)*, vol. 31, no. 1, pp. 1–38, Mar. 2006.

[18] T. J. Lehman and M. J. Carey, "A Study of Index Structures for Main Memory Database Management Systems," in *Proceedings of the 12th International Conference on Very Large Data Bases (VLDB '86)*, California, USA, 1986, pp. 294–303.

[19] A. Silberschatz, H. F. Korth, and S. Sudarshan, "Indexing and Hashing," in *Database System Concepts*, 6th ed. New York: MaGraw-Hill, 2011, ch. 11, pp. 475–536.

[20] L. J. Gosink, K. Wu, E. W. Bethel, J. D. Owens, and K. I. Joy, "Bin-Hash Indexing: A Parallel Method for Fast Query Processing," Laurence Berkeley National Laboratories, California, Tech. Rep., 2008.

[21] Z. Q. Abdulhadi, Z. Zuping, and H. I. Housien, "Bitmap Index as Effective Indexing for Low Cardinality Column in Data Warehouse," *International Journal of Computer Applications*, vol. 68, no. 24, pp. 38–42, Apr. 2013.

[22] Y. Mei, K. Ji, and F. Wang, "A Survey on Bitmap Index Technologies for Large-scale Data Retrieval," in *Proceedings of 2013 6th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, Shenyang, Chaina, 2013, pp. 316–319.

[23] Z. Chen, Y. Wen, J. Cao, W. Zheng, J. Chang, Y. Wu, G. Ma, M. Hakmaoui, and G. Peng, "A Survey of Bitmap Index Compression Algorithms for Big Data," *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 100–115, Feb. 2015.

[24] J. Wook and K. Assistant, "Binning Strategy for Hierarchical Bitmap Indices with Large Scale Domain Hierarchy," *International Journal of Applied Engineering Research*, vol. 11, no. 18, pp. 9289–9296, 2016.

[25] X. Qin, "Performance Comparison of Index Schemes for Range Query of Big Data," in *Proceedings of 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, Changsha, China, 2016, pp. 1469–1473.

[26] K. Stockinger, K. Wu, and A. Shoshani, "Strategies for Processing ad hoc Queries on Large Data Warehouses," in *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP (DOLAP '02)*, Virginia, USA, 2002, pp. 72–79.

[27] R. R. Sinha and M. Winslett, "Multi-resolution Bitmap Indexes for Scientific Data," *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 3, pp. 1–38, Aug. 2007.

[28] M.-C. Wu and A. P. Buchmann, “Encoded Bitmap Indexing for Data Warehouses,” in *Proceedings of 14th International Conference on Data Engineering*, Florida, USA, 1998, pp. 220–230.

[29] K. Wu, E. Otoo, and A. Shoshani, “On The Performance of Bitmap Indices for High Cardinality Attributes,” in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30 (VLDB ’04)*, Toronto, Canada, 2004, pp. 24–35.

[30] K. Wu, A. Shoshani, and K. Stockinger, “Analyses of Multi-level and Multi-component Compressed Bitmap Indexes,” *ACM Transactions on Database Systems (TODS)*, vol. 35, no. 1, pp. 1–52, Feb. 2010.

[31] M. Stabno and R. Wrembel, “RLH: Bitmap Compression Technique based on Run-length and Huffman Encoding,” *Information Systems*, vol. 34, no. 4-5, pp. 400–414, Jul. 2009.

[32] S. Kim, J. Lee, S. R. Satti, and B. Moon, “SBH: Super Byte-aligned Hybrid Bitmap Compression,” *Information Systems*, vol. 62, pp. 155–168, Dec. 2016.

[33] K.-L. Wu and P. S. Yu, “Range-based Bitmap Indexing for High Cardinality Attributes with Skew,” in *Proceedings of The Twenty-Second Annual International Computer Software and Applications Conference (COMPSAC ’98)*, Vienna, Austria, 1998, pp. 61–66.

[34] C.-Y. Chan and Y. E. Ioannidis, “An Efficient Bitmap Encoding Scheme for Selection Queries,” *ACM SIGMOD Record*, vol. 28, no. 2, pp. 215–226, Jun. 1999.

[35] S. Vanichayobon, J. Manfuekphan, and L. Gruenwald, “Scatter Bitmap: Space-Time Efficient Bitmap Indexing for Equality and Membership Queries,” in *Proceedings of 2006 IEEE Conference on Cybernetics and Intelligent Systems*, Bangkok, Thailand, 2006, pp. 6–11.

[36] N. Wattanakitrungroj and S. Vanichayobon, “Dual Bitmap Index: Space-time Efficient Bitmap Index for Equality and Membership Queries,” in *Proceedings of*

2006 International Symposium on Communications and Information Technologies (ISCIT), Bangkok, Thailand, 2006, pp. 568–573.

[37] A. Hamadou and K. Yang, "An Efficient Bitmap Indexing Strategy based on Word-aligned Hybrid for Data Warehouses," in *Proceedings of International Conference on Computer Science and Software Engineering (CSSE 2008)*, Hubei, China, 2008, pp. 486–491.

[38] N. Goyal, S. K. Zaveri, and Y. Sharma, "Improved Bitmap Indexing Strategy for Data Warehouses," in *Proceedings of the 9th International Conference on Information Technology (ICIT '06)*, Bhubaneswar, India, 2006, pp. 213–216.

[39] G. Antoshenkov, "Byte-aligned Bitmap Compression," in *Proceedings of Data Compression Conference, 1995 (DCC '95)*, Utah, USA, 1995, p. 476.

[40] K. Wu, E. J. Otoo, and A. Shoshani, "Compressing Bitmap Indexes for Faster Search Operations," in *Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM '02)*, Scotland, UK, 2002, pp. 99–108.

[41] A. Colantonio and R. Di Pietro, "Concise: Compressed 'n' Composable Integer Set," *Information Processing Letters*, vol. 110, no. 16, pp. 644–650, Jul. 2010.

[42] F. Deliège and T. B. Pedersen, "Position List Word Aligned Hybrid: Optimizing Space and Performance for Compressed Bitmaps," in *Proceedings of the 13th International Conference on Extending Database Technology (EDBT '10)*, Lausanne, Switzerland, 2010, pp. 228–239.

[43] D. Lemire, O. Kaser, and K. Aouiche, "Sorting Improves Word-aligned Bitmap Indexes," *Data and Knowledge Engineering*, vol. 69, no. 1, pp. 3–28, Jan. 2010.

[44] F. Fusco, M. Stoecklin, and M. Vlachos, "NET-FLi: On-the-fly Compression, Archiving and Indexing of Streaming Network Traffic," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1382–1393, Sep. 2010.

[45] Y. Wen, Z. Chen, G. Ma, J. Cao, W. Zheng, G. Peng, S. Li, and W. L. Huang, "SECOMPAX: A Bitmap Index Compression Algorithm," in *Proceedings of*

2014 23rd International Conference on Computer Communications and Networks (ICCCN), Shanghai, China, 2014, pp. 1–7.

[46] J. Chang, Z. Chen, W. Zheng, J. Cao, Y. Wen, G. Peng, and W. L. Huang, "SPLWAH: A Bitmap Index Compression Scheme for Searching in Archival Internet Traffic," in *Proceedings of IEEE International Conference on Communications*, London, UK, 2015, pp. 7089–7094.

[47] W. Andrzejewski and R. Wrembel, "GPU-WAH: Applying GPUs to Compressing Bitmap Indexes with Word Aligned Hybrid," in *Proceedings of the 21st International Conference on Database and Expert Systems Applications (DEXA 2010)*, Bilbao, Spain, 2010, pp. 315–329.

[48] S. J. V. Schaik and O. D. Moor, "A Memory Efficient Reachability Data Structure Through Bit Vector Compression," in *Proceedings of 2011 ACM SIGMOD International Conference on Management of data (SIGMOD '11)*, Athens, Greece, 2011, pp. 913–924.

[49] J. Chang, Z. Chen, W. Zheng, Y. Wen, J. Cao, and W.-L. Huang, "PLWAH+: A Bitmap Index Compressing Scheme based on PLWAH," in *Proceedings of 2014 ACM/IEEE Symposium on Architectures for Networking and Communication Systems, (ANCS)*, California, USA, 2014, pp. 257–258.

[50] G. Guzun, G. Canahuate, D. Chiu, and J. Sawin, "A Tunable Compression Framework for Bitmap Indices," in *Proceedings of 2014 IEEE 30th International Conference on Data Engineering (ICDE)*, Illinois, USA, 2014, pp. 484–495.

[51] Y. Wu, Z. Chen, Y. Wen, W. Zheng, and J. Cao, "COMBAT : A New Bitmap Index Coding Algorithm for Big Data," *TSINGHUA SCIENCE AND TECHNOLOGY*, vol. 21, no. 2, pp. 136–145, Apr. 2016.

[52] N. Goyal and Y. Sharma, "New Binning Strategy for Bitmap Indices on High Cardinality Attributes," in *Proceedings of the 2nd Bangalore Annual Compute Conference (COMPUTE '09)*, Bangalore, India, 2009, pp. 1–5.

[53] D. Rotem, K. Stockinger, and K. Wu, “Optimizing Candidate Check Costs for Bitmap Indices,” in *Proceedings of the 14th ACM international conference on Information and knowledge management, (CIKM ’05)*, Bremen, Germany, 2005, p. 648.

[54] A. Keawpibal, N. Wattanakitrungroj, and S. Vanichayobon, “Enhanced Encoded Bitmap Index for Equality Query,” in *Proceedings of 2012 8th International Conference on Computing Technology and Information Management (ICCM)*, Seoul, South Korea, 2012, pp. 293–298.

[55] W. V. O. Quine, “The Problem of Simplifying Truth Functions,” *The American Mathematical Monthly*, vol. 59, no. 8, pp. 521–531, 1952.

[56] E. J. McCluskey, “Minimization of Boolean Functions,” *Bell Labs Technical Journal*, vol. 35, no. 6, pp. 1417–1444, 1956.

[57] G. R. Alam, M. Y. Arafat, M. Kamal, and U. Iftekhar, “A New Approach of Dynamic Encoded Bitmap Indexing Technique based on Query History,” in *Proceedings of the 5th International Conference on Electrical and Computer Engineering (ICECE ’08)*, Dhaka, Bangladesh, 2008, pp. 20–22.

[58] J. Sainui, S. Vanichayobon, and N. Wattanakitrungroj, “Optimizing Encoded Bitmap Index Using Frequent Itemsets Mining,” in *Proceeding of 2008 International Conference on Computer and Electrical Engineering (ICCEE ’08)*, Phuket, Thailand, 2008, pp. 511–515.

[59] N. Keawpibal, J. Duangsuwan, W. Wettayaprasit, L. Preechaveerakul, and S. Vanichayobon, “DistEQ: Distributed Equality Query Processing on Encoded Bitmap Index,” in *Proceedings of the 2015 12th International Joint Conference on Computer Science and Software Engineering, (JCSSE)*, Songkhla, Thailand, 2015, pp. 309–314.

[60] N. Keawpibal, L. Preechaveerakul, and S. Vanichayobon, “Optimizing Range Query Processing for Dual Bitmap Index,” *Walailak Journal of Science and Technology (WJST)*, vol. 16, no. 2, pp. 133–142, Feb. 2019.

[61] W. Weahama, S. Vanichayobon, and J. Manfuekphan, “Using Data Clustering to Optimize Scatter Bitmap Index for Membership Queries,” in *Proceedings of 2009 International Conference on Computer and Automation Engineering (ICCAE 2009)*, Bangkok, Thailand, 2009, pp. 174–178.

[62] “TPC-H: a decision support benchmark,” (Accessed on 2017, Dec. 15). [Online]. Available: http://www.tpc.org/tpch/

VITAE

Name Mr. Naphat Keawpibal

Student ID 5710230025

Educational Attainment

Degree	Name of Institution	Year of Graduation
Master of Science (Computer Science)	Prince of Songkla University	2012
Bachelor of Science (Computer Science) with first class honor	Prince of Songkla University	2010

Scholarship Awards

PSU.GS financial support for thesis, Fiscal year 2017 from Graduate School, Prince of Songkla University, 2017 – 2018.

PSU-PhD scholarship from Prince of Songkla University, 2014 – 2019.

Research Assistant Scholarship supported by Faculty of Science, Prince of Songkla University, 2010 – 2012.

Work – Position and Address

Lecturer at Department of Information Technology Business, Prince of Songkla University, Surat Thani campus, Surat Thani, Thailand, May 2013 – October 2013.

Lecturer at Department of Computer Science, Suratthani Rajabhat University, Surat Thani, Thailand, May 2012 – April 2013.

List of Publications

- Naphat Keawpibal, Ladda Preechaveerakul, Sirirut Vanichayobon, "HyBiX: A Novel Encoding Bitmap Index for Space- and Time-Efficient Query Processing", *Turkish Journal of Electrical Engineering & Computer Sciences*, Vol. 27, No. 2, 2019, pp. 1504–1522. doi: 10.3906/elk-1807-277.

• Naphat Keawpibal, Ladda Preechaveerakul, Sirirut Vanichayobon, “Optimizing Range Query Processing for Dual Bitmap Index”, *Walailak Journal of Science and Technology (WJST)*, Vol. 16, No. 2, 2019, pp. 133–142.

List of Proceedings

• Naphat Keawpibal, Jarunee Duangsuwan, Wiphada Wettayaprasit, Ladda Preechaveerakul, Sirirut Vanichayonon, “DistEQ: Distributed Equality Query Processing on Encoded Bitmap Index”, in *2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE 2015)*, Hat Yai, Thailand, 2015, pp. 309–314. doi: 10.1109/JCSSE.2015.7219815.

• Amorntep Keawpibal, Niwan Wattanakitrungroj, Sirirut Vanichayonon, “Enhanced Encoded Bitmap Index for Equality Query”, in *2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT)*, Seoul, South Korea, 2012, pp. 293–298.