Performance of Multi-Level and Multi-Component Compressed Bitmap Indexes

Kesheng Wu, Kurt Stockinger and Arie Shoshani
Lawrence Berkeley National Laboratory

June 11, 2007

Abstract

Bitmap indexes are known as the most effective indexing methods for range queries on append-only data, especially for low cardinality attributes. Recently, bitmap indexes were also shown to be just as effective for high cardinality attributes when certain compression methods are applied. There are many different bitmap indexes in the literature but no definite comparison among them has been made, largely because there is no accurate prediction of their index sizes and search time. This paper presents a systematic evaluation of two large subsets of compressed bitmap indexes that use multi-component and multi-level encodings. We combine extensive analyses with ample experimental results to confirm them, whereas earlier studies of these indexes are either empirical or for uncompressed indexes only. Our analyses provide highly accurate predictions that agree with test measurements. These analyses not only identify the best methods in terms of index size and query processing cost, but also reveal new ways of using multi-level methods that significantly improve their performance. Using the best parameters obtained through analyses, we produce three two-level indexes with the optimal computational complexity. Furthermore, the fastest two-level indexes are predicted and observed to be 5 to 10 times faster than other well-known indexes.

1 Introduction

Various bitmap indexes have been demonstrated to significantly speed up searching operations in data warehousing, On-Line Analytical Processing (OLAP), and many scientific data management tasks [4, 8, 12, 16, 17, 23, 25, 33]. This has led a number of commercial database management systems (DBMS) to support bitmap indexes [2, 17]. However, most of the bitmap index implementations in commercial DBMS are relatively simple, such as the basic bitmap index [20] or the bit-sliced index [21]. There is a significant number of promising techniques proposed in the research literature that have not gained wide acceptance yet. We believe that a poor understanding of their performance characteristics is the most important reason for this lack of implementation. For example, many of these bitmap indexes are analyzed without considering compression, even though compression is critical to ensure good performance in practice. In this paper, we conduct a thorough performance evaluation to account for the effects of compression, with the goal of finding the best bitmap indexing method.

All known strategies to improve the basic bitmap index may be divided into three categories called binning, encoding and compression [28]. Among the three, encoding is by far the largest category, however their impacts on the overall index performance are less studied than those of binning or compression. For this reason, studying the encoding methods is more likely to lead to the best bitmap index method.

All encoding methods proposed in the past ten years can be categorized as either a multi-component encoding or a multi-level encoding. So far, most of the encoding methods are either studied without compression, or with a limited number of empirical observations. In this paper, we plan to remedy this situation by performing an extensive analytical study of all encoding methods. In order for the study to be realistic, we always use the most powerful compression method found in literature. Under this compression, we are

able to obtain precise analytical formulas for both the index sizes and the query processing costs for all multi-component and multi-level encodings. We also provide extensive tests to show that the formulas are indeed accurate for a variety of data and queries.

Through our analyses, we also identify a number of encoding methods that produce compact indexes and require low query processing costs. Among these methods, a new two-level interval-equality encoding appears to be the winner. It requires much less query response time on average than the bit-sliced index and the basic bitmap index (with compression), at the cost of using somewhat more space. However, it still uses less space than a typical B-Tree index from a commercial DBMS system.

In the remaining of this paper, we first present in Section 2 a review of different bitmap indexes and point out how this paper goes beyond what is in literature. We discuss the multi-component and multi-level indexes along with their performance characteristics without compression in Section 3 and Section 4. The analyses of the compressed bitmap indexes span three sections, from Section 5 to 7. In Section 5, we analyze three basic one-component encodings; in Section 6, the multi-component encodings; and in Section 7, the multi-level encodings. As a summary of these analyses, we discuss the optimality of these compressed indexes in Section 8. In Section 8, we also identify five indexing methods for an empirical performance study. These indexes all take less disk space than commonly used B-Tree indexes, and have lower query processing costs. The empirical study is presented in Section 9. A brief summary and discussion on future directions are given in Section 10.

2 Review

Following the publication of O'Neil's seminal paper on bitmap indexes [20], there has been a number of publications on the subject [5, 6, 7, 15, 18, 22, 21, 26, 32, 39, 38, 37]. In addition, there are a number of techniques from closely related indexing techniques which can be easily adapted to bitmap indexes as well, such as inverted files [31] and signature files [11, 40]. In this section, we briefly review known bitmap indexes based on how they are constructed. We use this review to help us identify the largest groups of methods that can be analytically studied. We also discuss the distinctions between this work and related ones.

We regard an indexing technique to be a *bitmap index* if the bulk of the index data is stored as sequences of bits and these bit sequences are primarily used in bitwise logical operations to answer queries. Typically, each sequence of bits has as many bits as the number of rows in the data table. Each such sequence is called a *bitmap*, which gives rise to the name of bitmap index. Following the taxonomy outlined in [28], we divide the bitmap indexing techniques into three categories, namely, *binning* [15, 32, 39], *encoding* [6, 7, 21] and *compression* [1, 5, 34]. A bitmap index typically uses a combination of these three types of techniques, though it is common to omit one or two. For example, the first commercial implementation of a bitmap in Model 204 uses equality encoding without binning or compression [20].

Binning: Given a set of arbitrary values, the objective of the binning step is to produce a set of identifiers (e.g., bin numbers) to be used in later steps of bitmap index construction. The example shown in Figure 1 illustrates how a set of floating-point values are divided into three bins. In a more general case, the values may be divided into an arbitrary number of bins, and a bin may contain disjoint ranges of values. A basic binning strategy is to have each distinct value in a single bin which effectively implies *no binning*. On a digital computer, all values are discrete and therefore it is always possible to build a bitmap index with no binning, regardless whether the values are strings, integers or floating-point numbers.

The *cardinality* C of an attribute in a dataset is defined to be the number of distinct values it has. With no binning, we can think of it as using C bins. A typical binning strategy will use less bins. The key advantage of binning is that it may reduce index sizes; however, the disadvantage is that the index is no longer able to fully resolve all queries. For example, the range condition "A$<$ 0.3" can not be resolved with the bins shown in Figure 1 because the query boundary (0.3) does not fall on a bin boundary. The bin a query boundary falls in is known as an *edge bin*. One has to examine all values in the edge bin in order to accurately answer the query. This step of checking all values in the edge bins called *candidate check* can dominate the total

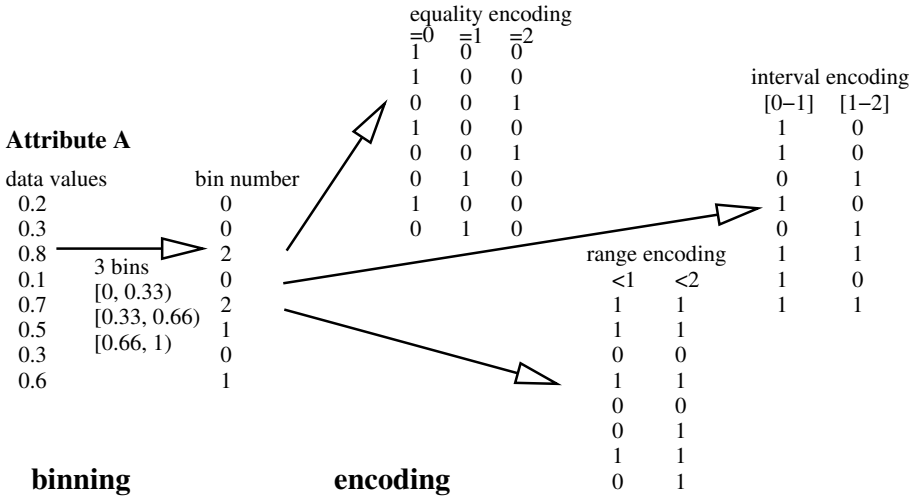


Figure 1: An illustration of the first two steps of the bitmap index construction process.

query processing time [29, 30, 26].

In later analyses, we choose to consider only *no binning* to avoid the candidate check. Additionally, with no binning, the compressed bitmap indexes can have reasonable sizes and good performance.

Encoding: Conceptually, the encoding step takes the bin identifier for each row and translates it to a set of bits. Usually, each bit is appended to a bitmap. This results in a series of bitmaps with as many bits as the number of rows. In the simplest encoding, a bitmap corresponds to exactly one bin. This encoding is known as equality encoding [6] (which was also called 1-of-N and unary encoding in [31]), where a bit is set to 1 if the value of a row falls in the bin corresponding to the bitmap. An illustration of this is shown in Figure 1, where each bitmap (a column in the illustration) represents a single bin. Note that equality encoding is well suited for a type of queries called *equality queries*, such as, "$A = 3$." We use the term *basic bitmap index* to denote an equality encoded index with no binning.

Other common encoding schemes include *range encoding* and *interval encoding*. In these encodings, an ordering among the bin identifiers is needed. Since bins are typically identified with integers, this requirement is easily satisfied. For range encoding, a bit corresponding to a row is set to 1 if the bin identifier is less than or equals to the identifier associated with the bitmap. This encoding is well suited for *one-sided range queries* such as "$A < 5$."

Each bitmap under interval encoding corresponds to a set of the bins. In the example shown in Figure 1, the first bitmap corresponds to bins 0 and 1, while the second bitmap corresponds to bins 1 and 2. In general, each bitmap under interval encoding corresponds to about half the bins and the number of bitmaps is about half the number of bins [7]. In addition to reducing the number of bitmaps compared with equality encoding and range encoding, this encoding was also shown to be well suited for *two-sided range queries* such as "$5 \leq A < 8$."

More sophisticated encoding schemes can be generated from the above three basic encoding schemes, equality encoding, range encoding and interval encoding. One approach is to decompose the bin number into several components (described in Section 3) and encode each component using a basic encoding scheme [6, 7]. We call them *multi-component* encodings. The best known example of such an encoding is the binary encoding [31], which is also known as the bit-sliced index [21]. Another strategy of extending the basic encoding schemes is the *multi-level encoding* (described in Section 4), which can be viewed as using a hierarchy of bins of different resolutions [35, 27].

If we allow the multi-component encodings to include the one-component variants, then the classes of multi-component and multi-level encodings cover almost all known encoding methods. Other encoding methods, such as the K-of-N encoding [31] and the superimposed encoding [14], are not actively used for

bitmap indexes.

Compression: To reduce the storage requirement, each bitmap generated from the encoding step may be compressed. Any lossless compression method can be used for this purpose, but they may have drastically different query processing cost. For example, most generic text compression methods, such as LZ77, are effective in reducing the index size on disk, but they can also significantly increase the time required to answer a query, because the compressed bitmaps have to decompressed before being used in logical operations. To reduce the query response time, specialized compression methods are usually used. One such method is the Byte-Aligned Bitmap Code (BBC) [1, 13]. It can compress bitmaps as well as the best generic text compression methods, and at the same time it also reduces the query response time. The BBC compressed basic bitmap index is implemented in ORACLE DBMS.

Recently, another compression method called the Word-Aligned Hybrid (WAH) code has been shown to outperform BBC in most cases [29, 37, 34]. This method trades some space for more efficient CPU operations. In one set of tests, it was shown to use about 50% more space than BBC, but answered queries 10 times faster on average [36]. WAH compressed indexes are efficient because they particularly simple (see [37] for a contrast with BBC). This simplicity also makes it amenable to analysis as demonstrated in [34]. Existing literature involving WAH compression mostly concentrate on demonstrating the effectiveness of this compression method using one index, the basic bitmap index. In fact, a WAH compressed basic bitmap index can answer a range query in $O(h)$ time, where h is the number of hits. In terms of computational complexity theory, it is an *optimal* indexing method, an exclusive club including some of the best B-Tree indexes such B+Tree and B*-Tree [9]. Because of these reasons, we regard WAH as the best compression for our study and choose to use it to limit the scope of this paper.

In summary, among all possible combinations of bitmap indexes, we plan to analyze those with WAH compression and no binning. This significantly extends earlier work on WAH compression, which only considered the basic bitmap index [29, 34]. It also significantly extends the earlier work on different encoding methods [6, 7], which did not thoroughly consider the effects of compression. Even though some authors did consider compression in their performance studies [35, 27], they presented only a small number of empirical measurements. In this paper, we present extensive analyses of all encoding methods proposed in the last ten years and support the analytical study with careful performance measurements. Furthermore, the analytical results also lead to new and more efficient encoding methods that were not previously considered.

3 Multi-Component Indexes

The multi-component indexes include most of the common encoding methods, such as the basic bitmap index [20] and the bit-sliced index [17, 21]. They are constructed from the three basic encoding schemes by decomposing the bin numbers into several components. For example, the bin numbers $(0,\ldots,999)$ for 1000 bins may be broken into three components of basis size 10 each. Each of these components would be a digit of the three-digit decimal numbers. Let i denote the bin number, i_1, i_2 and i_3 denote the values of the three components, the relation among them can be written as $i = 100i_1 + 10i_2 + i_3$. In general, each component may be a different size. Such a three-component index can be viewed as composed of three separate bitmap indexes on i_1, i_2 and i_3; and each component could be encoded independently with one of the three basic encoding methods.

The multi-component encoding defined above mostly follows the definition given in reference [6]. However, this encoding was also described earlier by Wong at al. [31].

A multi-component encoding is usually constructed with some user input parameters. For example, if the user chooses the number of components, then it is possible to automatically decide the size of each component to minimize the number of bitmaps generated. For instance, if a user specified to use a two-component encoding for 100 bins, then having each component of size 10 is a reasonable choice. One objective of analyzing this type of multi-component encoding is to find the optimal strategies to decide the number of components to use. In an earlier study, Chan and Ioannidis [6] suggested that the optimal number of components to use is 2. Their analysis only considered bitmap indexes without compression. With WAH

compression, the optimal number may be different.

An alternative to fixing the number of components is fixing the base size of each component and use as many components as necessary to represent all the bin numbers. For example, we may fix the component base size to be always 2. In the above example of 100 bins, it requires 7 components ($2^7 > 100$). In this special case, we only need to store one bitmap for each component. This is true no matter which of the three basic encoding schemes is used. We call this special case the *binary encoding* [31]. The binary encoded bitmap index is also known as the *bit-sliced index* [21]. Among all possible choices of fixed base sizes, this base size 2 is very unique because it produces the least number of bitmaps. In our performance study, we only consider this binary encoding as the representative of fixing base sizes for multi-component encodings.

Let B denote the number of bins. The binary encoding uses $\lceil \log_2 B \rceil$ bitmaps. For ease of comparisons, we will neglect the ceiling operator ($\lceil\rceil$) and simply write the number of bitmaps as $\log_2 B$. Because nearly every bitmap is needed to answer a query when using a binary encoded index, we always read in all the bitmaps from disk when a binary encoded index is needed. This decreases the number of I/O operations and reduces the I/O overhead. In terms of operations on the bitmaps to answer a query, a majority of bitmaps needs to participate in more than one bitwise operation. Therefore, answering a query using a binary encoded index may take more CPU time than other indexes mentioned above. Next, we briefly recapitulate the key performance characteristics of other multi-component indexes without compression.

Let $C_1, C_2, \ldots, C_k$ denote the sizes of a k-component encoding, or *basis sizes*. Using equality encoding, the ith component has C_i bitmaps. The total number of bitmaps is $D^E \equiv \sum_{i=1}^{k} C_i$. The values $C_1, C_2, \ldots, C_k$ satisfy the condition $\prod_{i=1}^{k} C_i \geq B$. Chan and Ioannidis [6] proposed the following choice of basis sizes to minimize D^E,

$$C_1 = C_2 = \ldots = \sqrt[k]{B}. \tag{1}$$

Note that $C_1, C_2, \ldots, C_k$ have to be integers and the above expression is close to the true integer solution only if B is relatively large. One practical way of assigning values of C_i is to assign some of them $\lceil \sqrt[k]{B} \rceil$ and some of them $\lfloor \sqrt[k]{B} \rfloor$ so that $\prod_{i=1}^{k} C_i$ is no less than B but close to B. However, to make it easier to compare different indexing methods, we drop the floor and ceiling operators, and assume $\prod_{i=1}^{k} C_i = B$. Under the above choice, the minimal number of bitmaps is given by the following formula,

$$D^E_{min} = k\sqrt[k]{B}. \tag{2}$$

Similarly, for a component of size C_i, there are $C_i - 1$ bitmaps under range encoding and $(C_i - \lceil C_i/2 \rceil + 1 (\sim C_i/2)$ bitmaps under the interval encoding [6, 7]. Let D^R_{min} and D^I_{min} denote the minimal number of bitmaps under these encodings respectively, then they are given by the following formulas,

$$D^R_{min} = k\sqrt[k]{B} - k, \tag{3}$$

$$D^I_{min} = \frac{k}{2}\sqrt[k]{B} \tag{4}$$

Without compression, each bitmap is always the same size. In this case, minimizing the number of bitmaps also minimizes the index size. Next, we consider the query processing cost by counting the maximum number of bitmaps needed to answer a query.

A multi-component index with a base size greater than 2 uses more space than the binary encoded index, however, it only accesses some of the bitmaps in order to answer a query. Therefore, it is possible that a multi-component index may actually require less I/O time than a binary encoded index. To make the analysis of query processing cost concrete, we consider the cost to answer three types of range queries: the equality query (EQ), e.g., "$A = 3$," the one-sided range query ($1RQ$), e.g., "$A < 3$," and the two-sided range query ($2RQ$), e.g., "$1 \leq A < 3$". In later discussion, we call the constants in these range expressions as *query boundaries* because they mark the boundaries of the ranges.

To understand the query processing cost, we give an illustration of how a multi-component index is used to answer a range query. Complete algorithms for query evaluation have been published elsewhere [6, 7]. Let attribute A be an integer ranging from 0 to 99. We build a two-component equality encoded index where

	Encoding		
	Equality	**Range**	**Interval**
EQ	k	$2k$	$2k$
$1RQ$	$\frac{k}{2}\sqrt[k]{B}$	$2k-1$	$3k-1$
$2RQ$	$\frac{k}{2}\sqrt[k]{B}$	$4k-2$	$5k-2$

Table 1: The maximum number of bitmaps used to answer an equality query (EQ), a one-sided range query ($1RQ$), and a two-sided range query ($2RQ$) using a k-component bitmap index with the three basic encoding schemes.

both components have the same size of 10. In this case, each component can be viewed as a decimal digit of the two-digit numbers representing attribute A. Let the first component be the first digit and the second component be the second digit. The first bitmap of the first component marks all the records with the first digit being 0, i.e., all values between 0 and 9. Similarly, the third bitmap of the second component marks the position of all records with the second digit being 2.

To find all values less than or equal to 45, we proceed as follows. The first four bitmaps of the first component corresponds to records where the values of A is less than 40, therefore they are hits. We need to read these bitmaps into memory and perform bitwise OR operations on them. Some records represented by the fifth bitmap (containing records with values between 40 and 49) of the first component may satisfy the query condition as well, but we need to examine the second component to identify which records actually satisfy the query condition. The condition on the second component is $i_2 \leq 5$, which can be processed either as the result of ORing the first six bitmaps from the second component or the complement of ORing the last four bitmaps. A bitwise AND operation between the result produced from the second component and the fifth bitmap from the first component produces all records whose values are between 40 and 45. The result of this bitwise AND operation is then ORed with the result of ORing the first four bitmaps of the first component. This produces a bitmap representing all records satisfying the query condition. To answer a typical query, all components of a multi-component index are accessed. This is true no matter what basic encoding methods are used for the components. Table 1 shows the maximum number of bitmaps needed to answer the three types of queries. Notice that the number of components k is prominent in all cases.

Without compression, each bitmap requires the same amount of storage, the same amount of time to read and the same amount of time to perform a bitwise logical operation. Therefore, the number of bitmaps needed to answer a query is a good proxy of the query response time. However, because Table 1 only shows the maximum number of bitmaps used, not the actual number of bitmaps used, how an encoding scheme actually performs still requires direct timing measurements. As shown in [7], interval encoding actually performs better than range encoding even though it requires more bitmaps in the worst case.

With compression, the number of bitmaps required to answer a query does not change, however, the sizes of the bitmaps do change. This changes the I/O time as well as the CPU time required to perform bitwise logical operations. Chan and Ioannidis [7] have conducted some timing measurements using the Byte-Aligned Bitmap Code (BBC). However, their tests were conducted on low cardinality attributes so that the uncompressed indexes could be manageable. In this paper, we target high-cardinality attributes where uncompressed indexes would be too large to be practically useful. Hence, compression is essential in this case.

4 Multi-Level Indexes

From Table 1 it is clear that an equality encoded index may need to access a large number of bitmaps in order to answer a range query. Clearly, it is worthwhile to reduce the number of bitmaps needed to answer a query. Range and interval encoding access less bitmaps, but they produce bitmaps that are hard to compress. Therefore, it is worthwhile to study other alternatives. One such alternative is the multi-level encoding

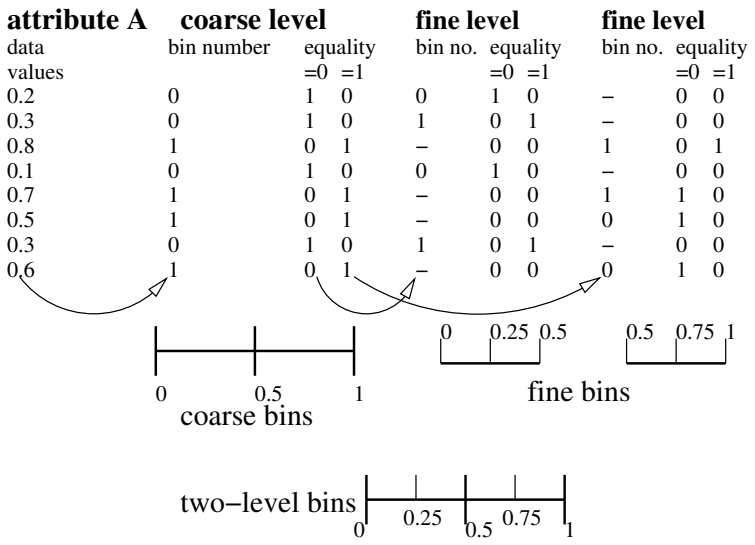


Figure 2: An illustration of a two-level equality-equality encoding of four bins.

[35, 27]. In these studies, the multi-level encoding was used with binning methods that require candidate checks, which resulted in performance measurements that do not truly represent the characteristics of the encoding schemes. In this paper, we study the multi-level encoding with no binning, which removes the need for candidate checks. This allows a better understanding of the performance characteristics of these multi-level encodings.

Conceptually, we can think of a multi-level encoding as binning the data with progressively finer bins as illustrated in Figure 2. Since each level can be encoded separately using any of the three basic encoding schemes, we can have arbitrary combinations of them. The example shown in Figure 2 is a two-level encoding with equality encoding on both levels.

In practice, we bin the data once at the finest level and determine the bitmaps at the coarser levels with bitwise logical operations. In experiments, we have found this option to be much more efficient than the alternative of performing repeated binning.

The two-level equality-equality encoding contains two separate 1-component equality encoded indexes. To answer a query, we could first use the coarse level index to produce an approximate solution and then use the fine level index to produce the precise solution. Let B_c denote the number of coarse bins. Using the coarse index to answer a range query may involve up to $B_c/2$ bitmaps. If the query boundaries fall inside some coarse bins, it would be necessary to use the fine level indexes in those coarse bins to produce the precise answer. We may need to use two such fine level indexes for a two-sided range condition. Since each fine index may have B/B_c bitmaps, if we need to access half of them in each coarse bin on average, we access a total of B/B_c fine level bitmaps. The total number of coarse level and fine level bitmaps accessed is therefore $B_c/2 + B/B_c$. Given the number of bins B, the value of B_c that minimizes this total number of bitmaps is

$$B_c = \sqrt{2B}. \tag{5}$$

5 Analysis of Basic Encoding Methods

Our analyses of encoding methods are split into three sections. In this section, we give the definitions and analyze the basic encoding methods. In the next two sections, we analyze the more complex versions of multi-component and multi-level encodings. We present a summary of analysis results in Section 8 to identify the most effective methods. For the most part, our analyses concentrate on the average performance

of an encoding method on random queries, instead of the cost of answering a single query using a specific bitmap index as in [34].

The bulk of this section is on analyzing the three basic encoding methods, equality encoding (Section 5.3), range encoding (Section 5.4) and interval encoding (Section 5.5). Before these analyses, we first introduce the model data and queries used in the analyses (Section 5.1), and give the formulas for computing the sizes of compressed bitmaps (Section 5.2), which are the building blocks of all further analyses.

Among the three basic encodings, without compression, interval encoding produces the smallest indexes; equality encoding is the most efficient for equality queries in term of query processing cost, range encoding is the most efficient for one-sided range queries and interval encoding is the most efficient for two-sided range queries. However, with compression, equality encoding produces the most compact indexes. Even though range encoding and interval encoding are still more efficient in answering one-sided and two-sided range queries than equality encoding, their index sizes grow linearly with the attribute cardinality for a large variety of data, which makes them unsuitable for high-cardinality attributes.

5.1 Model data and queries

For our analyses, we need to have data and queries that can be easily characterized. In general, each attribute of our synthetic data is generated by a separate random process and the ith value generated by this process will be placed in the ith row. The synthetic data we use are always integer values, though the indexing methods can just as easily index floating-point values and string values. Because sizes of bitmaps can be computed the same way, we use integer values as examples in the analyses.

The simplest synthetic data is the uniform random data, where each value is randomly taken from the range [1, ..., C] with probability 1/C. In the main body of this article, the analysis mainly concentrates on this uniform random data. Many of the standard benchmarks including TPC[1] use this type of synthetic data. It is simple to describe but relatively hard to compress. In particular, it is the hardest type of data to compress. Therefore, it represents the worst-case scenario for WAH compressed bitmap indexes. However, we also use two other types of synthetic data that are considered as more realistic than the uniform random data, the Zipfian data [**?**] and the Markovian data [3]. Due to space limitations, the derivation of index sizes for these two types of data are given in the appendix.

To keep the analysis concise, we only consider bitmap indexes with no binning, therefore the number of bins B is equal to the attribute cardinality C. Throughout this paper, we use N to denote the number of rows in a dataset. The uniform random data can be fully described with two parameters N and C. For Zipfian data, we need another parameter known as the Zipf exponent z. A typical value for z is 1. When z is 0, the Zipfian data degenerates to uniform random data. As the Zipf exponent increases, the Zipfian data is said to be more skewed. The Zipfian data with a $z = 2$ is considered as high-skewed. For Markovian data, the additional parameter we use is called the clustering factor f, which measures the average number of time a value appears consecutively. For example, if the values of an attribute always appear twice together in a dataset, such as, 1, 1, 2, 2, 1, 1, 2, 2, ..., its clustering factor is 2. The uniform random data has a clustering factor of about 1 ($C/(C-1)$ to be exact). It is common to see a set of application data to have f between 2 and 4. We say that data with large f are more clustered. For the purpose of our discussion, a clustering factor f of less than 10 is considered moderate.

Earlier, we defined three types of queries, equality (EQ), one-sided range ($1RQ$), and two-sided range ($2RQ$) queries. To simplify the discussion, we restrict the query boundaries and the operators used to define the range conditions. To distinguish them from arbitrary queries, we call our restricted forms the *canonical forms* of queries, or simply canonical queries. A canonical query only uses query boundaries that have appeared in the actual data records. Let a_i denote the ith distinct values of attribute A. Each canonical query such as "$A \leq a_i$"is the representative of all queries with query boundaries between a_i (inclusive) and a_{i+1} (exclusive). One important characteristics of these canonical queries is that there is only finite number of them. For example, for an attribute with cardinality C, there are exactly C canonical equality queries.

[1] TPC benchmark is defined by the TPC council, more information available at *http://tpc.org*.

For the range queries, the canonical queries only use the less than or equal to operator ($\leq$) to define range conditions. Let a_i and a_j be the ith and jth distinct values of an attribute A. A canonical one-sided range query is "$A \leq a_i$" and a canonical two-sided range query is "$a_i \leq A \leq a_j$." We note that an arbitrary query either has no hit or can be transformed into a canonical query that produces the same answer. For example, if attribute A has integer values, and 2 is an observed value but 3 is not, then "$A > 3$" can be translated to "not $A \leq 2$." The similar transformation can be done for floating-point values as well. Interested readers may consult IEEE Standard 754 on floating-point numbers.

In the following analyses, we count query processing cost as the I/O cost, more specifically the number of words accessed. This gives us a machine-independent measure for comparing different indexing methods. It is a good proxy of the query response time because earlier analyses and performance measurements have established that the total query response time is dominated by the cost of I/O operations [29]. Based on this definition of query processing cost, transforming an arbitrary query into its canonical form does not reduce the query processing cost. In the above example, the answer to a query is the complement of an answer to a canonical query. In this process, we need to compute the complement of a bitmap. Since computing the complement can be carried out in memory, this operation has an I/O cost of 0.

To compute the average query processing cost, we assume that all observed values are equally likely to be used as the query boundaries of an equality query or a one-sided range query. For an attribute with cardinality C, there are C instances of the canonical equality query and the canonical one-sided range query. The *average query processing cost* is the average cost to answer the C queries. For the two-sided range queries, we construct C^2 instances as follows. Take two values with equal probability from C observed values, use the smaller one as the lower bound of the query range and use the larger one as the upper bound. If the two values are the same, the two-sided range query degenerates to an equality query. If one of the values is the minimal or maximal value, the two-sided range is equivalent to a one-sided range. Thus, the two-sided range query is the most general form of the three types of queries. When comparing different encoding methods, we compare the cost of answering an average two-sided range query. However, we also study the performance of equality queries and one-sided range queries because these queries are simpler and studying their performance characteristics helps us better understand the performance characteristics of two-sided range queries.

5.2 Size of compressed bitmap

Since we use WAH compression, the bulk of a bitmap index is a set of WAH compressed bitmaps. In this subsection, we give definitions and summarize the key results on the size of a WAH compressed bitmap [34].

For random data, each bitmap in a bitmap index is a random bitmap. Such a random bitmap can be described by two parameters, the number of bits N and the *bit density* d that is the fraction of bits that are 1. A bitmap in a bitmap index typically corresponds to some values in a dataset, and the density of the bitmap is the same as the total frequency of these values. Given enough records in the dataset, this total frequency is same as the total probability of the values if the data is generated following a probability distribution. In our analyses, we assume N is very large and the observed frequency of a value is the same as the prescribed probability for the value.

Following the notation used in [34], a sequence of consecutive 1s is called a 1-fill. The clustering factor f of a bitmap is the average number of bits in all 1-fills. Since WAH is based on the run-length encoding, a bitmap with larger f will likely require less space to store. A sequence of consecutive 0s is a 0-fill. Let w denote the number of bits in a computer word, WAH divides a bitmap into groups of $(w-1)$ bits each. A literal word in WAH stores one such group and a count word in WAH represents a fill spanning multiple such groups.

It takes $\lfloor \frac{N}{w-1} \rfloor + 2$ words if every word is a literal word, where $\lfloor \frac{N}{w-1} \rfloor$ are regular literal words, one additional word to store the remaining $N\%(w-1)$ leftover bits and one to store the value $N\%(w-1)$, where $\%$ denotes the modulo operator. This is the maximum number of words needed. If there are two consecutive

C:	The attribute cardinality, the number of distinct values of the attribute in a dataset.
d:	The bit density, the fraction of bits that are 1.
f:	Clustering factor of a bitmap, the average number of bits in 1-fills.
m:	The number of regular words in a WAH compressed bitmap, $m_r(d)$ is the expected number words for a random bitmap and $m_M(d,f)$ is the expected number of words for a bitmap generated from a two-state Markov model. We also use m_i to indicate the size of the ith bitmap.
N:	The number of rows in a dataset, also the number of bits in a bitmap of a bitmap index.
s:	Expected size of a compressed bitmap index, for example, s_U^E is the index size for a uniform random attribute using equality encoding. In general, the superscript E is used for equality encoding, R for range encoding, and I for interval encoding. The subscript U is for uniform random data, z for Zipfian data and M for Markovian data.
t_i:	The query processing cost (a proxy of query response time) to answer the ith canonical query. Used only during the derivation of the average cost.
T:	The average query processing cost. We use a superscript to denote the encoding method (E, R, and I) and a subscript to denote the type of queries, EQ for equality queries, $1RQ$ for one-sided range queries and $2RQ$ for two-sided range queries. For example, T_{EQ}^E for the average query processing cost to answer equality queries using equality encoded indexes and T_{1RQ}^R for the average query processing cost to answer one-sided range queries using range encoded indexes.
w:	Word size, the number of bits in a word, typically, 32 or 64.

Table 2: List of frequently used symbols.

literal words that actually store the same 0-fill or 1-fill, they could be combined into one count word and reduce the storage requirement by one word. There are $\lfloor \frac{N}{w-1} \rfloor - 1$ such two-word groups. In a random bitmap, each bit is independent from another and each has probability d to be 1, therefore the probability for all $2w-2$ bits in a two-word group to be 1 is d^{2w-2}. Similarly, the probability for all of them to be 0 is $(1-d)^{2w-2}$. This leads to the following expressions for the number of words required by a WAH compressed random bitmap,

$$\begin{aligned} m_r(d) &= \left\lfloor \frac{N}{w-1} \right\rfloor + 2 - \left(\left\lfloor \frac{N}{w-1} \right\rfloor - 1 \right) \left((1-d)^{2w-2} + d^{2w-2} \right) \\ &\approx \frac{N}{w-1} \left(1 - (1-d)^{2w-2} - d^{2w-2} \right). \end{aligned} \tag{6}$$

5.3 Equality encoding

We now consider the expected performance of a one-component equality encoded bitmap index. We first compute the index sizes and then the average query processing costs.

5.3.1 Index sizes

The uniform random dataset is the simplest to evaluate. In this case, the bitmap index for an attribute with C distinct values contains C bitmaps. Each of these C bitmaps is a random bitmap with bit density $d = 1/C$.

Based on Equation 6, which gives the sizes of each such random bitmap, the total size of bitmaps in an index is

$$s_U^E = C * m_r(1/C) \approx \frac{CN}{w-1}(1-(1-1/C)^{2w-2}-1/C^{2w-2}) \tag{7}$$

For a range of attribute cardinality C, where $1 \ll C \ll N$, $1/C^{2w-2} \approx 0$, and by Taylor expansion we have $(1-1/C)^{2w-2} \approx 1-(2w-2)/C$, the above expression for s_U^E can be approximated as

$$s_U^E \approx 2N. \tag{8}$$

This indicates that for a large range of attribute cardinalities, a WAH compressed, equality encoded bitmap index takes about $2N$ words for uniform random data [37, 34]. If C is very small, the bitmaps cannot be compressed by WAH and each of them takes about $N/(w-1)$ words. If C is close to N, the overhead of WAH compression starts to dominate the total index size. We refer interested readers to [37, 34] for more details about this case.

5.3.2 Query processing cost

Given above index sizes, we can now compute the I/O cost of answering the three types of queries. In our evaluations, we include only the cost of accessing the necessary bitmaps, but not the metadata required to locate these bitmaps. Given a large data set, the metadata is typically much smaller than the bitmaps, as we observe from the actual measurements shown in Section 9. Next, we consider the cost of answering three types of queries in turn.

Equality query To evaluate the average cost to answer an equality query, we assume that each observed value is equally likely to be used as the query boundary. There are C canonical equality queries for an attribute with cardinality C. Since the cost of answering an equality query is to read one bitmap, the average query processing cost is therefore the average size of a bitmap,

$$T_{EQ}^E = s/C. \tag{9}$$

In this case, the total size s can be any of s_U^E, s_z^E or s_M^E.

One-sided range query There are C instances of canonical one-sided range queries. If the ith observed value is used as the query boundary, the first i bitmaps may be used to answer the query as $b_1 \mid \ldots b_i$, where b_i denotes the ith bitmap (corresponding to a_i) and $\mid$ denotes the bitwise OR operation. In later discussion, we also shorten this expression as $\sum_{j=1}^{i} b_j$.

Since we can compute the complement of a bitmap fast, if more than a half of the bitmaps are to be accessed, our query evaluation procedure will evaluate the complement of the query, which accesses less bitmaps, and then compute the complement of the resulting bitmap to produce the final answer for the query. We denote the complement option as computing $\sim \sum_{j=i+1}^{C} b_j$, where $\sim$ denotes the bitwise complement operation.

Let m_i denote the size of the ith bitmap, the straightforward evaluation strategy accesses $\sum_{j=1}^{i} m_j$ words, while the alternative strategy accesses $\sum_{j=i+1}^{C} m_j$ words. We always choose the less expensive option, which leads to the following expression for the query processing cost, $t_i = \min(\sum_{j=1}^{i} m_j, \sum_{j=i+1}^{C} m_j)$. Knowing the size of each bitmap m_j, we can compute the average query processing cost using the following double summation,

$$T_{1RQ}^E = \frac{1}{C}\sum_{i=1}^{C} \min\left(\sum_{j=1}^{i} m_j, \sum_{j=i+1}^{C} m_j\right). \tag{10}$$

For attributes following uniform distribution, where all m_j are the same, we can give a much more concise formula for the average query processing cost. Let m denote the size of a bitmap, the cost of

processing a query involving the ith value as the query boundary is $t_i = \sum_{j=1}^{i} m = mi$ if $i \leq \lfloor C/2 \rfloor$, and $t_i = \sum_{j=i+1}^{C} m = m(C-i)$ otherwise. The total cost of evaluating all C instances of one-sided canonical queries can be split into two parts following the two different formulas for t_i. Furthermore, we know that $\sum_{i=1}^{\lfloor C/2 \rfloor} i = (1+\lfloor C/2 \rfloor)\lfloor C/2 \rfloor/2$, and $\sum_{i=\lfloor C/2 \rfloor+1}^{C}(C-i) = (C-\lfloor C/2 \rfloor)(C-\lfloor C/2 \rfloor-1)/2$. Taken together, the expression for T^E_{1RQ} can be rewritten as follows,

$$\begin{aligned} T^E_{1RQ} &= \frac{1}{C}\sum_{i=1}^{C} t_i \\ &= \frac{m}{C}\left(\frac{1}{2}\left(1+\left\lfloor\frac{C}{2}\right\rfloor\right)\left\lfloor\frac{C}{2}\right\rfloor + \frac{1}{2}\left(C-\left\lfloor\frac{C}{2}\right\rfloor\right)\left(C-\left\lfloor\frac{C}{2}\right\rfloor-1\right)\right) \\ &= \frac{m}{2C}\left(2\left\lfloor\frac{C}{2}\right\rfloor^2 + (C-1)\left(C-2\left\lfloor\frac{C}{2}\right\rfloor\right)\right) \\ &\approx \frac{m}{C}\left\lfloor\frac{C}{2}\right\rfloor^2 \approx \frac{mC^2}{4C} = \frac{mC}{4} = \frac{s}{4}. \end{aligned} \tag{11}$$

We note that the above approximation is based on $\lfloor C/2 \rfloor \approx C/2$, which is accurate when C is large. Because we assumed that the bitmaps are of the same size, the variable s in the above formula can be either s^E_U or s^E_M.

For non-uniform data distributions, we cannot derive such compact formula for the query processing cost, but can directly evaluate the summation defined in Equation 10. Generally, the average average cost is a smaller fraction of the total index sizes than in the case of uniform random data. For example, for Zipfian data with $C = 100$ and $z = 1$, this average is about $0.22 s^E_{z=1}$, which is about 12% less than $\frac{1}{4}s^E_{z=1}$ for uniform data (Equation 11).

Two-sided range query Given that a_i and a_j are two observed values of A (where i may be the same as j), an instance of the canonical two-sided range query is "$\min(a_i, a_j) \leq \mathrm{A} \leq \max(a_i, a_j)$." There are a total of C^2 instances of such queries. To evaluate one instance, we could either work with bitmaps between i and j, or outside the range, similar to the case of one-sided range queries. Let b_i denote the ith bitmap, the direct option computes $\sum_{k=i}^{j} b_k$ and the complement option computes $\sim (\sum_{k=1}^{i-1} b_k + \sum_{k=j+1}^{C} b_k)$. Note that we have replaced operator $|$ with $+$ for consistency. Let m_i denote the size of the ith bitmap, the query processing cost can be expressed as

$$t_{ij} = \min\left(\sum_{l=\min(i,j)}^{\max(i,j)} m_l, \sum_{l=1}^{\min(i,j)-1} m_l + \sum_{l=\max(i,j)+1}^{C} m_l\right).$$

Assuming that all C^2 instances of two-sided range queries have equal probability of being used, the average query processing cost is

$$T^E_{2RQ} = \frac{1}{C^2}\sum_{i=1}^{C}\sum_{j=1}^{C} t_{ij}. \tag{12}$$

This summation can be significantly simplified if all m_i are the same, say m. In this case, t_{ij} is strictly a function of difference between i and j, $t_{i,j} = \min(m(|i-j|+1), m(C-|i-j|-1))$. We can rearrange the double summation over i and j into a summation over $|i-j|$. Define $|i-j|+1$ to be the width of the query, there are C query instances with width 1, $2(C-1)$ instances with width 2, $2(C-2)$ instances with width 3, ..., and 2 instances with width C. Therefore, the average query processing cost can be rewritten as

$$T^E_{2RQ} = \frac{1}{C^2}\left(Cm + \sum_{l=2}^{C} 2(C+1-l)\min(ml, m(C-l))\right).$$

If $l \leq \lfloor C/2 \rfloor$, then $l \leq C-l$. We can remove the function min, by splitting the above summation in two parts, the first part with l less than or equal to $\lfloor C/2 \rfloor$, where the query processing cost is ml, and the second

part with l greater than $\lfloor C/2 \rfloor$, where the query processing cost is $m(C-l)$. Moreover, the first part of the summation is $\sum_{l=2}^{\lfloor C/2 \rfloor} 2(C+1-l)ml = m(2+3C-2\lfloor C/2 \rfloor)(1+\lfloor C/2 \rfloor)\lfloor C/2 \rfloor/3 - 2mC \approx mC^3/6$, and the second part of the summation is $\sum_{l=\lfloor C/2 \rfloor+1}^{C} 2(C+1-l)m(C-l) = 2((C-\lfloor C/2 \rfloor+1)(C-\lfloor C/2 \rfloor^2)(C-\lfloor C/2 \rfloor^2-1))/3 \approx mC^3/12$. The preceding approximations rely on the fact that $\lfloor C/2 \rfloor \approx C/2$, which is accurate for large C. Furthermore, only the highest order terms of C are kept, which is again accurate for large C. Therefore, for uniform data with high attribute cardinalities, say $C \geq 100$, the average cost to process a two-sided range query is

$$T^E_{2RQ} \approx mC/4 = s/4. \tag{13}$$

where s can be either s^E_U or s^E_M. Note that this is the same as the average cost T^E_{1RQ} for one-sided queries.

For non-uniform data, the average query processing cost is expected to be less than that of uniform data. For example, for a random attribute following Zipf distribution with $z = 1$ and $C = 100$, the average cost computed through Equation 12 is $0.23 s^E_{z=1}$, which is about 10% less than $\frac{1}{4} s^E_{z=1}$. Note also that as C increases, the average cost of querying Zipfian data becomes a smaller fraction of the total index size.

5.4 Range encoding

The bitmaps produced using the one-component range encoding are effectively the precomputed answers to one-sided range queries. A key advantage of this encoding is that it can answer range queries quickly, but the bitmaps are not amenable to compression. In this subsection, we quantify the index size and query processing cost to see if it offers a good space-time trade-off.

5.4.1 Index size

As indicated previously, the critical first step in analyzing the performance of a WAH compressed bitmap index is to compute the bit density and the clustering factor of the bitmaps. Based on the definition of range encoding, the bit density of the ith bitmap in an index on a uniform random attribute is $d_i = i/C$, $i = 1, 2, \ldots, C-1$. Note that the last bitmap with all bits set to 1 is dropped by convention. The same formula also applies to the Markovian data.

Given the above formula for the bit densities, we can compute the total number of words of all bitmaps in a range encoded index on uniform random data as follows,

$$\begin{aligned} s^R_U &= \sum_{i=1}^{C-1} m_r(d_i) \\ &\approx \frac{N}{w-1} \sum_{i=1}^{C-1} \left(1 - \left(1 - \frac{i}{C}\right)^{2w-2} - \left(\frac{i}{C}\right)^{2w-2} \right). \end{aligned} \tag{14}$$

This formula defines the total size of bitmaps as a Power Sum[2]. Special functions can be used to express the results of Power Sums. However, for implementation in computer software, directly evaluating the sums may be just as efficient as evaluating those special functions. To get a better sense of how large s^R_U is, we seek to produce an approximation that is more compact.

For a majority of the bitmaps, the bit densities d_i are between 0.05 and 0.95 as shown in Figure 3. For these bitmaps, WAH compressed versions are nearly the same size as uncompressed ones, where each bitmap takes about $N/(w-1)$ words. Therefore, the expression

$$s^R_U \approx N(C-1)/(w-1) \tag{15}$$

should have no more than 10% error for uniform random data (shown as $z = 0$ in Figure 3). In many cases, the observed errors are less than 3%. Previously, we showed that s^E_U approaches $2N$ words (in Equation 8) as C increases, however, s^R_U increases linearly as C increases. This means that the size of a range encoded index could be arbitrarily larger than an equality encoded one.

[2]Eric W. Weisstein. "Power Sum." From *MathWorld*. http://mathworld.wolfram.com/PowerSum.html.

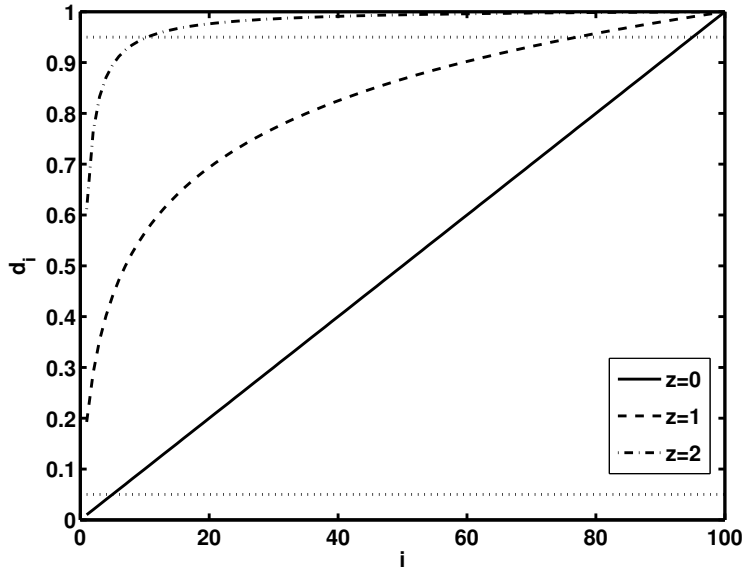


Figure 3: The bit density d_i of bitmaps from a range encoded index on Zipfian data ($C = 100$). WAH is only able to compress random bitmaps with bit density less than 0.05 or more than 0.95.

5.4.2 Query processing cost

Equality query To answer an equality query such as "A = a_i" using a range encoded bitmap index, we access bitmap b_{i-1} and bitmap b_i, and perform $b_i - b_{i-1} \equiv b_i \& \sim b_{i-1}$, where & denotes the bitwise AND operation and $\sim$ denotes the bitwise complement operation. As before, because the cost of performing bitwise logical operation in memory is small compared to the I/O cost, we count the cost of reading the bitmaps from disk as the proxy of the total query processing cost. In this case, to answer a canonical equality query, we need to access two adjacent bitmaps, except the cases involving a_1 and a_C as the query boundaries, where only one bitmap (b_1 and b_{C-1} respectively) needs to be read. Let m_i denote the size of bitmap b_i, the cost of processing a query, t_i, is usually $m_i + m_{i-1}$, except the two special cases mentioned above. The average query processing cost is

$$T^R_{EQ} = \frac{1}{C}\left(m_1 + m_{C-1} + \sum_{i=2}^{C-1}(m_i + m_{i-1}\right) = \frac{2}{C}\sum_{i=1}^{C-1} m_i = \frac{2s}{C}. \tag{16}$$

Note that s in the above equation can be any of s^R_U, s^R_z or s^R_M. For uniform data with moderate clustering factor ($f < 10$) and for Zipf data with $z \leq 1$, each WAH compressed bitmap takes about $N/(w-1)$ words and the query processing cost is approximately $2N/(w-1)$. Because answering an equality query using an equality encoded index requires only one bitmap, while two bitmaps are required using a range encoded index, T^R_{EQ} is at least twice as large as T^E_{EQ}. Similar performance differences between equality encoding and range encoding were also observed without compression [6].

One-sided range query To answer a one-sided range query using a range encoded index, we only need to read one bitmap. Therefore the cost of processing most instances of the one-sided range query is $t_i = m_i$, except the query involving a_C as the query boundary, where every record satisfying the condition, which requires no data from disk to produce the answer. The average query processing cost is the average bitmap size,

$$T^R_{1RQ} = \frac{1}{C}\sum_{i=1}^{C-1} m_i \approx \frac{s}{C}. \tag{17}$$

Note that s in above equation can be the size of any range encoded bitmap index.

Since a one-sided query can be answered with one range encoded bitmap, range encoding is still the most efficient at answering one-sided range queries on average. Take the uniform random data as example, the average cost of processing a one-sided range query using an equality encoded index is $s_U^E/4$ as shown in Equation 11. If the attribute cardinality is large then this average cost is approximate $2N/4 = N/2$ words because $S_U^E \approx 2N$ (see Equation 8). The average cost under range encoding is about $N/(w-1)$ words because $s_U^R \approx N(C-1)/(w-1)$ (see Equation 15). Therefore, using a range encoded index to answer a one-sided range query could be $(w-1)/2$ times faster than using an equality encoded index. This difference is smaller for data with skewed distribution or clustering.

Two-sided range query To answer a two-sided range query, we need to access two bitmaps as in the equality query case. The difference is that the two bitmaps may no longer be adjacent. In practice, because the two bitmaps are not adjacent, they require two separate read operations, while two adjacent bitmaps could usually be read in one operation. Even when the same number of bytes are read, using two read operations is likely to take more time than using one because of the extra function call overhead and possibly additional disk seek time. However, when the number of rows in the data set is large, this time difference could be only a small part of the overall I/O time. For this reason, we continue to count query processing cost as the number of words accessed to answer a query.

To answer a two-sided range query involving a_i and a_j as query boundaries, we access bitmaps $b_{\min(i,j)-1}$ and $b_{\max(i,j)}$, and compute $b_{\max(i,j)} \& \sim b_{\min(i,j)-1}$ as the answer. Range encoding only defines $b_1, \ldots, b_{C-1}$. To make the above expression valid, we assume that b_0 is a bitmap containing only 0s and b_C is a bitmap containing only 1s. Since b_0 and b_C can be generated in memory, the I/O cost for producing them are 0, i.e., $m_0 = m_C = 0$. Using this definition, we express the query processing cost t_{ij} as $t_{ij} = m_{\min(i,j)-1} + m_{\max(i,j)}$. We assume that every i and j between 1 and C are equally likely to be used in a two-sided range query, the average query processing cost is

$$T_{2RQ}^R = \frac{1}{C^2}\sum_{i=1}^{C}\sum_{j=1}^{C} t_{ij} = \frac{1}{C^2}\sum_{i=1}^{C-1} 2Cm_i = \frac{2s}{C}. \tag{18}$$

For uniform random data, the average cost of processing a two-sided range query using an equality encoded index is $s_U^E/4$ as shown in Equation 13. If the attribute cardinality is large, say $C \geq 100$, then this average cost is approximate $2N/4 = N/2$ words based on the index size given in Equation 8. The average cost under range encoding is about $2N/(w-1)$ words based on the formula for s_U^R from Equation 15. Therefore, using a range encoded index to answer a two-sided range query could be $(w-1)/4$ times faster than using an equality encoded index. On a 32-bit system where $w = 32$, we may say that range encoding is about 8 times faster than equality encoding. However, because s_U^R can be arbitrarily larger than s_U^E, we do not believe the range encoded indexes have a good space-time trade-off compared with the equality encoded ones.

5.5 Interval encoding

Interval encoding produces about half as many bitmaps as equality encoding and range encoding. It can answer the queries just as efficient as range encoding. Next, we quantify these performance characteristics for the basic one-component interval encoded index to see whether it presents a better space-time trade-off than the range encoded index.

5.5.1 Index size

Under the (one-component) interval encoding, each bitmap represents about half of the values, $\lceil C/2 \rceil$ to be exact. Therefore, the bit density of a bitmap in an index on a uniform random attribute is $d = \lceil C/2 \rceil / C \approx 1/2$. This type of bitmaps are not compressible with WAH compression. Since there are $C - \lceil C/2 \rceil + 1$ such

bitmaps, the total size of the bitmaps is

$$s_U^I \approx N(C - \lceil C/2 \rceil + 1)/(w-1) \approx \frac{NC}{2(w-1)}. \tag{19}$$

5.5.2 Query processing cost

Equality query To process an equality query, such as "A = a_i," using an interval encoded index, we need to access two bitmaps [7]. For example, if $i < \lceil C/2 \rceil$, the answer can be computed with $b_i - b_{i+1}$. If $i > \lceil C/2 \rceil$, the answer is obtained from $b_{i+1-\lceil C/2 \rceil} - b_{i-\lceil C/2 \rceil}$. These two cases encompass all instances of our C canonical equality queries except one. Though this exceptional case can also be answered by accessing two bitmaps, we can neglect it without adversely affecting the average query processing cost.

In short, the query processing cost to answer an equality query is either $m_i + m_{i+1}$ if $i < \lceil C/2 \rceil$ or $m_{i+1-\lceil C/2 \rceil} + m_{i-\lceil C/2 \rceil}$ if $i > \lceil C/2 \rceil$. We use the average query processing cost of these $C-1$ instances as the average of all C instances, and this average can be expressed as follows,

$$\begin{aligned} T_{EQ}^I &= \frac{1}{C-1}\left(\sum_{i=1}^{\lceil C/2 \rceil - 1}(m_i + m_{i+1}) + \sum_{i=\lceil C/2 \rceil + 1}^{C}(m_{i+1-\lceil C/2 \rceil} + m_{i-\lceil C/2 \rceil})\right) \\ &\approx \frac{1}{C}\sum_{i=1}^{C-\lceil C/2 \rceil + 1} 4m_i = \frac{4s}{C}. \end{aligned} \tag{20}$$

Under interval encoding, each bitmap b_i represents a range of values from a_i to $a_{i+\lceil C/2 \rceil - 1}$. To answer an equality query, each bitmap under interval encoding is used four times on average, in queries with boundaries a_{i-1}, a_i, $a_{i+\lceil C/2 \rceil - 1}$, and $a_{i+\lceil C/2 \rceil}$. This explains the average query processing cost being four times the average bitmap size. In contrast, the average cost of processing an equality query using a range encoded index is twice the average size of the bitmaps. This difference is due to the fact that there are less bitmaps under the interval encoding and therefore each bitmap is used more often than in the range encoded index. If all the bitmaps are not compressible as in the case of uniform data, where $s \approx \frac{CN}{2(w-1)}$, the average cost is approximately $\frac{4CN}{2(w-1)C} = 2N/(w-1)$. This average cost is the same as with range encoding, which is at least twice as expensive as answering the same equality queries with equality encoding.

One-sided range query To answer a one-sided range query using an interval encoded index, we need bitmap b_1 plus one additional bitmap depending on the query boundary a_i. If $i < \lceil C/2 \rceil - 1$, the answer is $b_1 - b_{i+1}$; if $i = \lceil C/2 \rceil - 1$, the answer is simply b_1; otherwise, the answer is $b_1 \mid b_{i-\lceil C/2 \rceil + 1}$, where $\mid$ denotes the bitwise OR operation. This means the cost of query processing $t_i = m_1 + m_{i+1}$ if $i < \lceil C/2 \rceil - 1$, $t_{\lceil C/2 \rceil - 1} = m_1$, if $i = \lceil C/2 \rceil - 1$, and $t_i = m_1 + m_{i-\lceil C/2 \rceil + 1}$ if $i \geq \lceil C/2 \rceil$. We can compute the average cost as follows,

$$\begin{aligned} T_{1RQ}^I &= \frac{1}{C}\left(\sum_{i=1}^{\lceil C/2 \rceil - 2}(m_1 + m_{i+1}) + m_1 + \sum_{i=\lceil C/2 \rceil}^{C}(m_1 + m_{i-\lceil C/2 \rceil + 1})\right) \\ &\approx \frac{1}{C}\left(Cm_1 + 2\sum_{i=2}^{C-\lceil C/2 \rceil + 1} m_i\right) \approx m_1 + \frac{2s}{C}. \end{aligned} \tag{21}$$

To answer the C queries, we always need b_1 and we need one additional bitmap in most cases. Since there are only $C/2$ bitmaps, most bitmaps are used twice. When none of the bitmaps are compressible, such as for uniform random data, Markovian data with moderate clustering factors, or Zipfian data with $z \leq 1$, this average cost is about twice as expensive as using range encoding.

Two-sided range query The procedure to answer a two-sided range query is somewhat similar to that for a one-sided range query. One key difference is that the bitmap b_1 may be replaced with another one according the smaller one of the query boundaries. Instead of going through an exhaustive account of the cost of every query, we observe that each bitmap in the index is used when one of the four query boundaries, a_{i-1}, a_i, $a_{i+\lceil C/2\rceil-1}$, or $a_{i+\lceil C/2\rceil}$, appears in a query. This leads to an average query processing cost of

$$T^I_{2RQ} \approx \frac{4s}{C}. \tag{22}$$

This average cost is the same as that for the equality queries. On uniform data and Zipfian data with $z \leq 1$, this average cost is the same as T^R_{2RQ} for range encoding. This implies that interval encoding is also about 8 times faster in answering two-sided range queries on uniform random data than equality encoding. On nearly uniform data, a interval encoded index is about half the size of a range encoded one, therefore, interval encoding is preferred over range range encoding. However, because an interval encoded index can still be arbitrarily larger than an equality encoded one, we need a way to reduce the index size to make it competitive against the equality encoded index.

6 Analysis of Multi-component Encodings

From the analyses in the previous section we see that range encoding and interval encoding can answer range queries fast, but they require too much space for high-cardinality attributes. Multi-component encodings were proposed to reduce their index sizes at the expense of increased query processing costs. In this section, we compute the index sizes and query processing cost of multi-component encodings to quantify the space-time trade-off. Our basic strategy is to treat each component as a simple encoding analyzed in the previous section.

Let C_1, C_2, ..., C_k denote the basis sizes of a k-component encoding. To simplify the discussions and for the same reasons given in the discussion of Equation 1, we assume $\prod_{i=1}^{k} C_i = C$. Furthermore, we use the same encoding for all components of an multi-component index. This limits our analysis to three multi-component encoding methods based on equality encoding, range encoding, and interval encoding. It will become clear that mixing different encodings for different components does not generate more efficient alternatives than the three discussed.

Without compression, the index sizes monotonically decrease with the number of components k used in a multi-component encoding and the query processing cost monotonically increases as shown in Section 3. Chan and Ioannidis [6] proposed that two-component encodings provide a good balance between the index size and the query processing cost. With compression, the index sizes no longer monotonically decrease with k, though the query processing cost still increases with k. We observe that the best encoding methods have either the minimum number of components ($k = 1$) or the maximum number of components (the binary encoding).

6.1 Multi-component equality encoding

Given a value a_i, the multi-component encoding translates i into a set of integers $(i_1, \ldots, i_k)$, where $i = \sum_{j=1}^{k} i_j \prod_{l=j+1}^{k} C_l$. Note that $\prod_{l=k+1}^{k} C_l$ defaults to 1. Under the k-component equality encoding, for each record, a bitmap from each component is set to 1. In particular, for value a_i, the i_jth bitmap in the jth component is set to 1. To compute the bit density and clustering factor of a bitmap in the jth component, we observe that the i_jth bitmap in the jth component is set to 1 as long as j component is i_j, no matter what the values of the other components are.

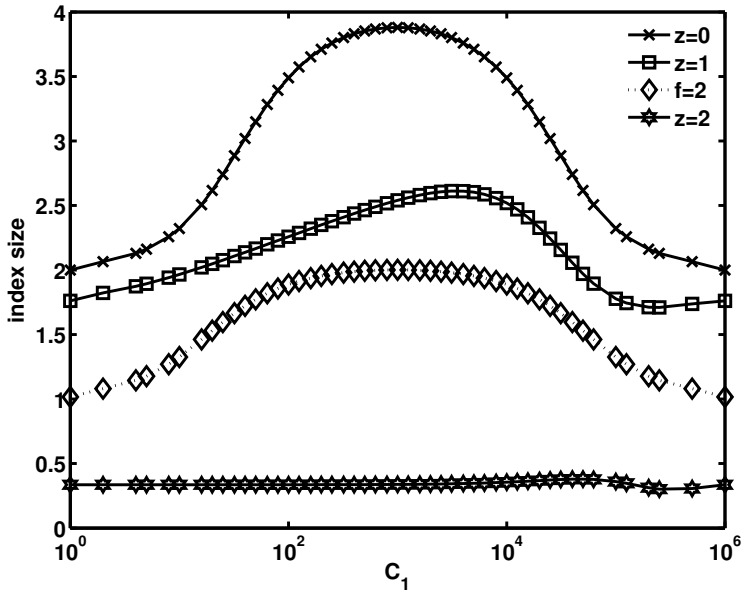


Figure 4: The index sizes (relative to the base data size) of two-component equality encoded indexes for an attribute with $C = 10^6$. The lines marked with 'z=x' are for Zipf attributes and the line marked 'f=2' is for a Markovian data with clustering factor $f = 2$. The horizontal axis C_1 is the basis size of the first component.

6.1.1 Index size

For uniform random data, each value i appears with probability of $1/C$. The probability that the jth component is a particular value i_j is

$$\sum_{i_1=1}^{C_1} \cdots \sum_{i_{j-1}=1}^{C_{j-1}} \sum_{i_{j+1}=1}^{C_{j+1}} \cdots \sum_{i_k=1}^{C_k} \frac{1}{C} = \frac{1}{C} \prod_{l \neq j} C_l = \frac{1}{C_j}.$$

This is the same as saying that each value in a component j is equally likely to appear. Therefore, we treat each component as a uniform random attribute, and express the total index size as follows (see Equation 6 for definition of m_r).

$$\begin{aligned} s_U^{Ek} &= \sum_{j=1}^{k} C_j m_r(1/C_j) \\ &\approx \frac{N}{w-1} \sum_{j=1}^{k} C_j \left(1 - (1-1/C_j)^{2w-2} - 1/C_j^{2w-2}\right) \end{aligned} \tag{23}$$

For very large C, where each C_j is also large ($1 \ll C_j \ll N$), following the same approximation that lead to Equation 8, we can approximate s_U^{Ek} as $2kN$. For relatively small k, the above assumption about C_j is valid, in which case, the total index size increases as the number of components increases. This trend is different from the uncompressed multi-component indexes.

As k becomes larger, the values of C_j become smaller, the total size of bitmaps in each component will be less than $2N$ words, and the total index size will likely decrease as k increases. This suggests that the minimal index sizes are achieved with either the minimal number of components or the maximum number of components.

Figure 4 shows the index sizes for a set of two-component equality encoded indexes. The indexes for four different attributes are plotted, and the lines are labeled with the Zipf exponents for Zipfian data, or the clustering factor for the Markovian data. All of them have an attribute cardinality of 1 million. The horizontal axis in this figure is the basis size of the first component C_1. The values of C_1 varies from 1 to 1

million. In the two extreme cases where $C_1 = 1$ and $C_1 = 10^6$, we effectively have one-component equality encoded indexes. In this figure, the index sizes are measured relative to the base data size assuming that each element of the base data requires one word of storage. For a one-component equality encoded index, the index size is about twice the size of the base data. The above analysis indicates that a two-component equality encoded index could be twice as large as a one-component index for uniform data. From Figure 4, we see that the two-component index sizes are indeed about twice that of the one-component versions for a number of different C_1 values, for both uniform random data ($z = 0$) and Markovian data ($f = 2$).

As the Zipf exponent increases, the index sizes decrease. The index on the Markovian data with $f = 2$ is nearly half the size of that of the uniform random data ($z = 0$). Another trend we observe is that as the Zipf exponent increases, the variation of index sizes becomes smaller as expected. As the Zipf exponent increases, the minimal index size is not achieved with $C_1 = 1$ or $C_1 = 10^6$. Since bitmaps for the first component are generally more compressible than those for the later components, making C_1 larger tends to decrease the total index size. Indeed, we see that the minimal index sizes are achieved with large C_1 values. In the case of $z = 1$, the optimal C_1 is 125,000 ($C/8$), and for $z = 2$, the optimal C_1 is 250,000 ($C/4$). However, in either case, the optimal index sizes are only slightly smaller than the one-component indexes, 3% and 10%, respectively, for $z = 1$ and $z = 2$. Since the indexes on uniform data are significantly smaller with the one-component equality encoding than with the two-component equality encoding, we prefer to use the one-component equality encoding over the two-component one.

6.1.2 Query processing cost

Equality query To answer an equality query, such as "A = a_i," we decompose i into k components $(i_1, \ldots, i_k)$, retrieve b_{i_j} from component j, and then perform bitwise AND on these k bitmaps. We again count the cost of answering this query as the total size of the k bitmaps that have to be read into memory. Since a bitmap from each component is needed, the average query processing cost is the sum of the average size of bitmaps in each component. The following is the expression for uniform random data.

$$T_{EQ}^{Ek} = \sum_{j=1}^{k} m_r(1/C_j) \approx \frac{N}{w-1} \sum_{j=1}^{k} (1 - (1 - 1/C_j)^{2w-2} - 1/C_j^{2w-2}).$$

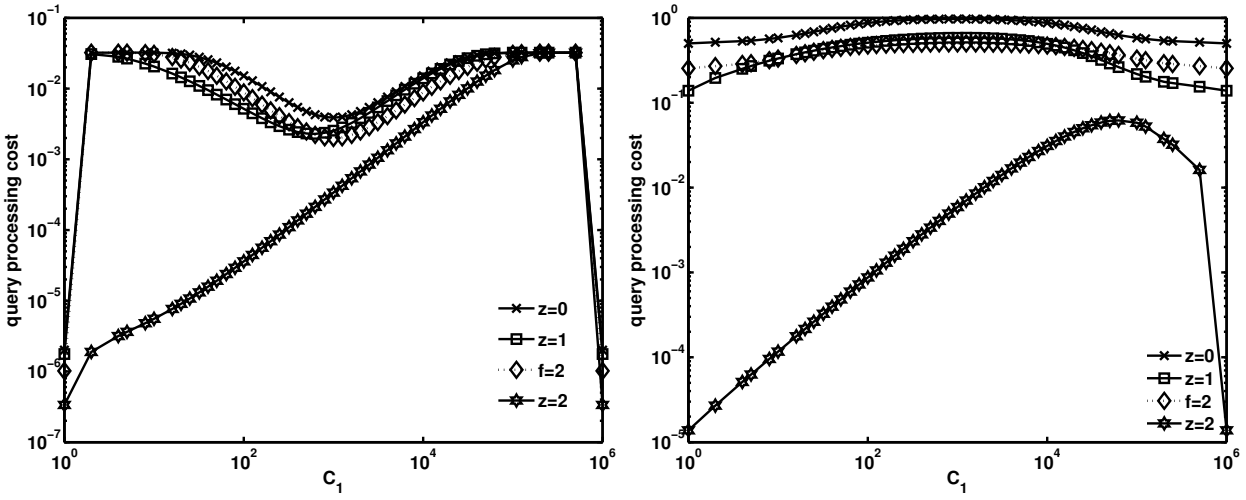
A similar expression can be written for Markovian data and Zipfian data. Because the value of C_j decreases as k increases, the cost of query processing increases faster than a linear function of k. However, as k further increases, most bitmaps become incompressible and the average size of bitmaps approaches $\frac{N}{w-1}$, the average query processing cost ultimately becomes a linear function of k.

One-sided range query In Section 3, we describe the procedure for evaluating a one-sided range query using the multi-component equality encoding. In terms of I/O cost, we see that the procedure is similar to answering a one-sided range query on each component. From the analysis of one-component equality encoding, we know the average I/O cost to answer a one-sided range query is equivalent to reading a quarter of all bitmaps for uniform data. Therefore, the average query processing cost of a one-sided range query is also equivalent to reading one quarter of the bitmaps.

$$T_{1RQ}^{Ek} \approx s/4.$$

In the above expression, s can be either s_U^{Ek} or s_M^{Ek}. Since the index sizes are larger for multi-component index than the one-component version, we expect using a multi-component equality encoded index to take longer than using a one-component equality encoded index.

Two-sided range query A two-sided range query can be answered as the difference between two one-sided range queries. More clever algorithms exist, however, their costs are still close to that of answering two one-sided range queries. For simplicity, we regard the average query processing cost for a two-sided range query to be twice of that for the one-sided range query.



(a) Equality queries (b) One-sided Range queries

Figure 5: The expected query processing cost (relative to the base data size) of two-component equality encoded indexes.

Examples and discussion Figure 5 plots the above formulas of the average query processing costs for a set of two-component equality encoded indexes. We measure the I/O cost relative to the base data size the same way we measure the index sizes in Figure 4. This I/O cost is the vertical axis in Figure 5. The horizontal axis is the basis size of the first component C_1. As in Figure 4, all attributes used have attribute cardinality of 1 million. Comparing the I/O cost with the base data size is equivalent to comparing against the *projection index* that reads a projection of the base data to answer a query. The projection index is known to be efficient for answering ad hoc queries on large datasets [21], so it serves as a good reference method. Alternatively, we may compare with the more commonly used B-Tree indexes. In a number of popular database systems, a B-tree index is observed to take 3 – 4 times more space of the base data. To answer an average one-sided range query, one needs to access about half of the leaves in a B-tree. This translates to a value of 1.5 to 2 in our plot, which is larger than the average query processing cost using a projection index, which is exactly 1. In Figure 4, the average query processing cost of using a two-component equality encoded index is always less than 1.

In Figure 5(a), we show the average cost to answer an equality query. In this case, all curves show a dramatic change in the cost near the two ends of the horizontal axis. Recall that at the two ends, the indexes have effectively only one-component. This indicates that the one-component equality encoded indexes are much more efficient in answering equality queries, which is known to be true without compression. It is not surprising that this remains true with WAH compression, however, what is a surprise is that the differences are almost four orders of magnitude.

In Figure 5(b), we show the average cost of answering a one-sided range query. For uniform data, such as the uniform random data and Markovian data, on average, a quarter of all the bitmaps are read into memory. For uniform random data, which is the most expensive case, using a one-component equality encoded index is equivalent to reading about half of the base data; the line for $z = 0$ in Figure 5(b) ends at 0.5 when $C_1 = 1$ and $C_1 = 10^6$.

In Figure 5(b), we again observe that the minimal values on each line are at the end of the horizontal axis where $C_1 = 1$ or $C_1 = 10^6$. This indicates the one-component equality encoded index is also more efficient in answering one-sided range queries than two-component equality encoded indexes. This is very different from the uncompressed case studied in earlier literature [6, 7] that predicted the minimum at $C_1 = \sqrt{C} = 1000$.

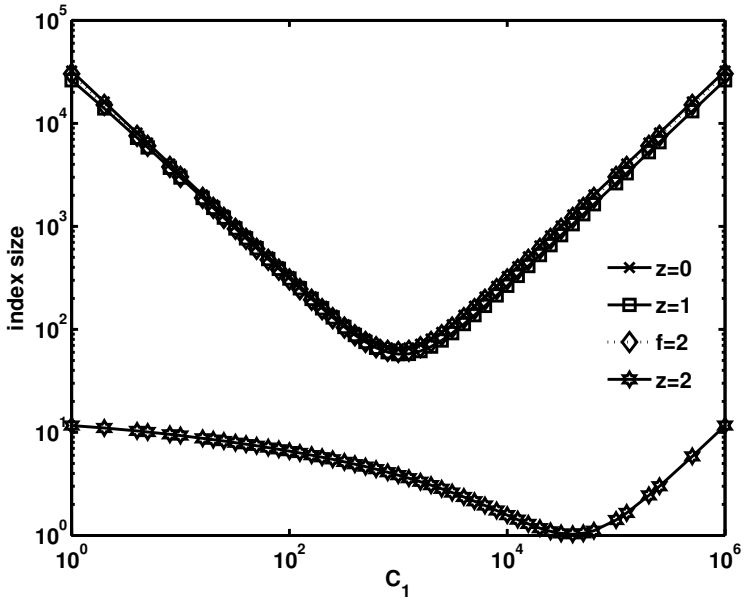


Figure 6: The index sizes (relative to the base data size) of two-component range encoded indexes for an attribute with $C = 10^6$. A value of 10 or larger should be considered impractical to use.

6.2 Multi-component range encoding

After decomposing a value into k components, the bitmaps produced for the multi-component equality encoding could be cumulated one component at a time to produce a multi-component range encoding. This process is similar to how a one-component equality encoding could be turned into a one-component range encoding. Since the last bitmap after the cumulation contains only 1s, we typically discard it to save space. Similar to the multi-component equality encoding, each component of the multi-component range encoding could be treated like a separate index.

Previously, we described how the bit densities and the clustering factors can be computed for the multi-component equality encoding. Since the i_jth bitmap in the jth component under range encoding is a cumulation (i.e., bitwise OR) of the first i_j bitmaps under the equality encoding. It is straightforward to follow the process used in Section 5.4 to compute bit densities and clustering factors of bitmaps under range encoding.

As in the one-component range encoding case, most of the bitmaps in a multi-component range encoded index are incompressible, if the data is uniform or nearly uniform. However, because the multi-component versions use significantly less bitmaps, the index sizes should be considerably less than in the one-component case. Figure 6 shows the sizes of a series of two-component range encoded indexes. As in Figure 4, we measure the index sizes relative to the base data size, and use the basis size of the first component (C_1) as the horizontal axis. We observe that the two uniform data ($z = 0$ and $f = 2$) and the Zipfian data with $z = 1$ have nearly identical index sizes. As in Figure 4, the sample attributes have attribute cardinality of 1 million. When $C_1 = 1$ or $C_1 = 10^6$, i.e., at the left and right ends of the horizontal axis, the two-component indexes degenerate to one-component ones. Clearly, using two components reduces the index sizes. For the three nearly uniform attributes, the minimal is achieved when $C_1 = C_2$ as predicted by Equation 1. However, even at their minimal sizes, the indexes contain 2000 incompressible bitmaps, and are more than 60 times the size of the base data. For the highly non-uniform data, $z = 2$, WAH compression is effective in reducing the index sizes. However, the minimal size shown in Figure 6 (about 1) is still much larger than the minimal size (about 0.3) for the two-component equality encoded indexes shown in Figure 4.

Based on the analysis of index sizes, WAH compressed multi-component range encoding behaves very much like its uncompressed counterpart, therefore we omit the analysis of the query processing cost. Recall that the query processing cost increases as the number of components increases as shown in Table 1.

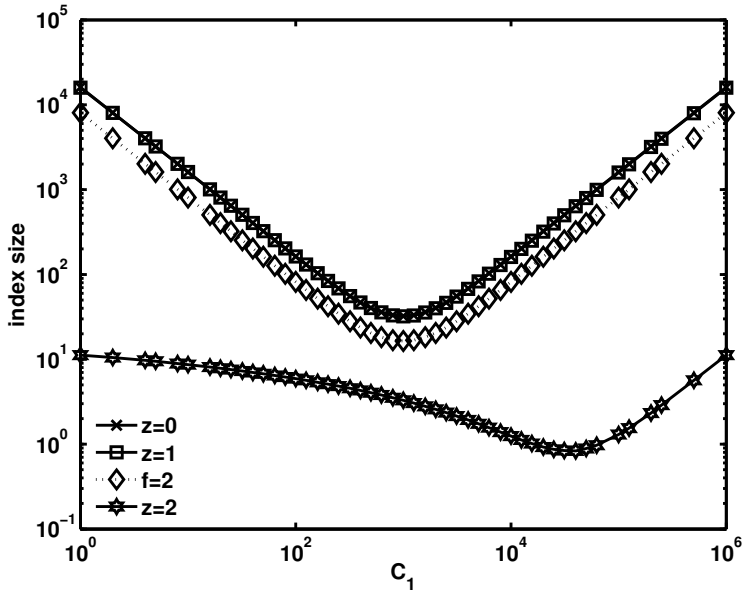


Figure 7: The index sizes (relative to the base data size) of two-component interval encoded indexes for an attribute with $C = 10^6$. A value of 10 or larger should be considered impractical to use.

6.3 Multi-component interval encoding

We can construct a multi-component interval encoded index from a multi-component equality encoded index. Following the same process that was used in Section 5.5 to compute the bit densities and clustering factors for the interval encoded bitmaps from their equality encoded counterparts, we can compute the index sizes. However, since the formulas are long set of summations that cannot be simplified, we choose to plot the expected sizes instead of giving these formulas. In most cases, WAH compression is unable to reduce the size of bitmaps, therefore WAH compressed indexes behave very close to their uncompressed versions. For this reason, we only briefly examine the sizes of the WAH compressed indexes. Since the bitmaps are usually incompressible, the cost of query processing are same as uncompressed indexes studied in Section 3.

Figure 7 shows the index sizes of a series of two-component interval encoded indexes. As in Figures 4 and 6, the vertical axis is the index size relative to the base data size and horizontal axis is the basis size of the first component C_1. Because the attribute cardinality is 1 million, at the left-most end ($C_1 = 1$) and the right-most end ($C_1 = 10^6$) of the figure, a two-component index effectively becomes a one-component interval encoded index. Since each bitmap under interval encoding is a bitwise OR of about half of the bitmaps from the same component in an equality encoded index, for nearly uniform data, $z = 0$, $z = 1$ and $f = 2$, these bitmaps cannot be compressed by WAH compression. Their minimal index sizes are more than 30 times the base data size. Only the index on the highly non-uniform attribute ($z = 2$) can be compressed. Even in this case, the minimal index size (about 0.8) is still considerably larger than the minimal index size (about 0.3) under equality encoding as shown in Figure 4.

6.4 Binary encoding

From the discussions on multi-component indexes, we observe that as the number of components increases, the size of a multi-component index will eventually decrease. The one with the largest number of component is the binary encoding. Because of its unique construction, it is the most compact multi-component index. Assuming that we can map the values of an attribute to an integer starting from 0 to $C-1$, for example, through binning and then number the bins from 0 to $C-1$, the bit density of jth bitmap is the probability that the jth binary digits of the bin number is 1. The clustering factor can be evaluated by estimating the probability of a binary digit changing from 1 to 0. After obtaining these two parameters, we can evaluate

the index sizes and the query processing cost as before.

6.4.1 Index size

The simplest case is the uniform random data, in which case, the probability of each binary digits have equal probability of being 0 or 1 (assuming that $log_2 C$ is an integer). Since bitmaps with density 0.5 are not compressible with WAH in general, we expect the index size to be

$$s_U^{BN} = \frac{N log_2 C}{w-1}. \tag{24}$$

Based on the experience with interval encoding (where most bitmaps has a density of 1/2), we expect that the indexes on mildly non-uniform Zipf data ($z \leq 1$) and Markovian data with relative small clustering factors ($f \leq 10$) to have the same sizes as the uniform random data.

6.4.2 Query processing cost

To answer an equality query, every bitmap of a binary encoding must be accessed, therefore,

$$T_{EQ}^{BN} = s. \tag{25}$$

To answer a one-sided range query, nearly every bitmap is needed. For example, the last bitmap, corresponding to the least significant binary digit, is not needed if the query boundary a_i has an odd number as its index i. The second to the last bitmap, corresponding to the second least significant binary digit, is not needed if the query boundary a_i has an index i that can be expressed as $4j-1$, where j is an integer. This can be generalized to the following expression for the average cost of one-sided range queries,

$$T_{1RQ}^{BN} = \sum_{i=1}^{log_2 C} m_i (1 - 2^{i-log_2 C-1}) \approx \frac{N}{w-1}(log_2 C - 1 + \frac{1}{C}) \approx \frac{N(log_2 C - 1)}{w-1} \tag{26}$$

In practice, we always read all bitmaps to simplify the software implementation. The cost of answering a two-sided range query requires more computations on the bitmaps than answering a one-sided range query, and it is more likely that every bitmap is needed. Therefore we simply state that cost of processing a two-side range query as reading the whole index

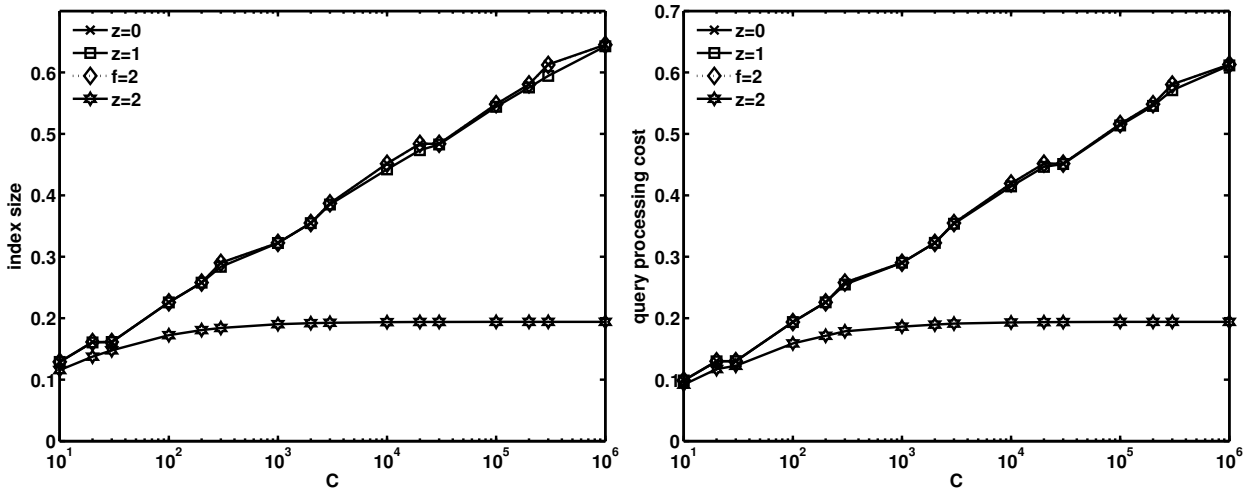
$$T_{2RQ}^{BN} = s. \tag{27}$$

6.4.3 Examples and discussion

Figure 8 shows the sizes of some binary encoded indexes and the cost of processing queries using these indexes. In both graphs, the horizontal axes are the attribute cardinality. The index sizes for the three nearly uniform attributes, $z = 0$, $z = 1$ and $f = 2$, are about the same. The bitmaps in these indexes are not compressible. As attribute cardinality approaches 1 million, it takes about 20 bitmaps under binary encoding. Assuming each word contains 32 bits, the binary encoded index size is about 0.65 ($= 20/31$) of the base data size. The indexes for the highly non-uniform attribute, $z = 2$, can be compressed. As attribute cardinality increases, it approaches about 0.2 times the base data size.

In Figure 8(b), we plotted the average query processing cost to answer a one-sided range query. Because nearly all of the bitmaps are needed, the query processing cost is about the same as the index size. The average I/O cost of answering an equality query or a two-sided query is even closer to the index size, we did not plot them.

For a high cardinality attribute with $C = 10^6$, the average cost of answering a one-sided range query is about 0.65 using a binary encoded index. The same cost is 0.5 for the basic bitmap index (i.e., a one-component equality encoded index). For such high-cardinality attributes, the basic bitmap index is larger



(a) Index size (b) One-sided range query cost

Figure 8: Performance information about binary encoded indexes. Both index size and the query processing cost are measured as the fraction of base data. Processing an equality query requires all bitmaps in an index.

in size but it can answer a range query with lower I/O cost. If the attribute cardinality is less than lower ($C < 32768, w = 32$), the binary encoded index has both lower space requirement and lower I/O cost to answer a range query. Often, the binary encoding is recommended for high-cardinality attributes; however, our analysis here shows that it is actually more efficient for lower cardinality attributes.

7 Analysis of Multi-level Encodings

We now explore some variations of the multi-level encodings. One feature we notice about these multi-level encodings is that each level is its own one-component index, therefore the analyses of the basic encodings can be easily applied for the analysis of the multi-level methods. Based on this observation, we identify three of the two-level methods that are likely to perform well, namely the equality-equality (EE) encoding, the range-equality (RE) encoding, and the interval-equality (IE) encoding. We only use equality encoding at the finest level because this level typically contains many bitmaps and only equality encoding could keep their total size modest. The reason for not considering more than two levels will become clear after we study the two-level methods.

Because each level is an complete bitmap index, it can be used to answer a query. In particular, the finest level can accurately answer any query that the multi-level index could. We can take advantage of this to minimize the query processing cost. In particular, since the fine level is an equality encoded index, we always use it to answer equality queries. Therefore the cost of answering an equality query is the same as shown in Equation 9 in Section 5.3. The cost of answering range queries may be reduced by using the coarser level index. To simplify the analysis of two-level encodings, we only consider the case of using both levels. In the implementation used in timing measurements to be discussed in Section 9, we examine all options and choose the most cost effective one.

In the remaining of this section, we first discuss how we propose to generate the coarse level bitmaps, then discuss the three two-level encodings in turn. We conclude this section with a discussion on the optimal number of coarse bins.

7.1 Generating coarse bins

To generate a two-level encoding, we build the fine level first and then build the coarse level from the fine level. This is a practical approach because the total size of fine level bitmaps (under equality encoding) is relatively small. In addition, having the fine level bitmaps also enables us to make more intelligent decisions on how to generate the coarse level bitmaps. Effectively, each coarse level bitmap is the result of bitwise logical OR between a number of fine level bitmaps. To decide which fine level bitmaps to group together is equivalent to placing the values corresponding to the fine level bitmaps into bins, hence the term coarse level bins. Recall that B_c denotes the number of coarse bins.

Knowing that the fine level has C distinct values, a simple choice is to divide them into B_c bins so that each bin has the same number of values. This choice was used in an earlier study [35]. It is easy to implement, but may take much longer to answer some queries than others, if the base data is non-uniform. For example, on Zipfian data with $z = 2$, the bitmaps in the first coarse bin would be much larger than those in later bins. A query touching the first coarse bin is likely to take much longer to answer than the average.

To reduce the variations in query processing costs, we make the total size of fine level bitmaps in each coarse bin to be the same. Because the decision on coarse bins is made after the fine level bitmaps are available, we know the sizes of fine level bitmaps and can determine the bin boundaries in $O(C)$ time.

Given C fine level bitmaps b_i, $i = 1,\ldots,C$, let m_i denote the size of each compressed bitmap. Compute the average size to be allocated to B_c bins, $M = \sum m_i/B_c$, and the cumulative sum of m_i, $l_i = \sum_{j=1}^{i} m_j$. The cumulative sums that are closest to a multiple of M could be determined and their positions used as boundaries for the coarse bins.

The above heuristics may place multiple bin boundaries at the same location, so we refine it by progressively locating the bin boundaries d_i. To start with, we compute the cumulative sum of the sizes of fine level bitmaps and then look for the value d_1 such that $\sum_{j=1}^{d_1} m_i \approx \sum_{i=1}^{C} m_i/B_c$. From the remaining bitmaps, we again look for a value d_2 such that $\sum_{j=d_1+1}^{d_2} m_j \approx \sum_{j=d_1+1}^{C} m_j/(B_c-1)$. This process can be carried out for all $B_c - 1$ bin boundaries; the last bin boundary must be C to cover all C fine level bitmaps. This algorithm can better accommodate the situation where some bitmaps are much larger than others. Note that the summations can be computed from the cumulative sums and the total cost of this algorithm is $O(C)$.

For uniform random data and Zipfian data, each coarse bin generated above would correspond to a set of values whose total probability is close to $1/B_c$. For an index on a high cardinality attribute, where C is large and B_c is much smaller than C, this approximation is accurate.

7.2 Equality-equality encoding

7.2.1 Index size

The fine level of this encoding is a basic bitmap index, therefore the total size of bitmaps at this level is given by Equations 7, 44 and 45 for uniform random data, Zipfian data and Markovian data respectively. By construction, the coarse level for our synthetic data is effectively an basic index for uniform random data because the values in each coarse bin have the same probability. Therefore, we can use Equation 7 to compute the total size of bitmaps at the coarse level. Next we give two specific examples, one for the uniform random data and one for the Markovian data.

For uniform random data, the total size of an equality-equality encoded index can be expressed as follows,

$$s_U^{EE} = Cm_r(1/C) + B_c m_r(1/B_c) \approx 2N + B_c N/(w-1). \tag{28}$$

Note the above approximation is for large C, $C \geq 100$, but small B_c, $B_c < 50$.

For the Markovian data, each coarse level bin contains about C/B_c values. The clustering factor of a coarse level bitmap is $f(1-1/C)/(1-1/B_c))$. The total size of coarse level bitmaps is given by $B_c m_M(1/B_c, f(C-1)B_c/(C(B_c-1)))$, which can be further expanded with Equation 40.

7.2.2 Query processing cost

One-sided range query To answer a one-sided range query "A $\leq a_i$" using the equality-equality encoded index, we first use the coarse level index. For $C - B_c$ out of C canonical one-sided range queries, the coarse level index is insufficient to resolve the query, and some fine level bitmaps are needed. Let b_i denote a fine level bitmap, the answer to our query can be expressed as $\sum_{j=1}^{i} b_j \equiv b_1 \mid \ldots \mid b_i$. Let k be the coarse bin that a_i falls in, i.e., $d_{k-1} < i < d_k$. For completeness, we assume $d_0 = 0$. We call the kth coarse bin the edge bin of the query. Let c_i denote the ith coarse level bitmap. The result of our canonical query can be expressed as $\sum_{j=1}^{k-1} c_j + \sum_{j=d_{k-1}+1}^{i} b_j \equiv c_1 \mid \ldots \mid c_{k-1} \mid b_{d_{k-1}+1} \ldots \mid b_i$.

There are a number of different ways to actually evaluate the above expression. For example, the bitwise OR operations involving the coarse level bitmaps can be either directly evaluated as $\sum_{j=1}^{k-1} c_j$, or computed as the complement of bitwise OR of remaining coarse level bitmaps, $\mathbb{1} - \sum_{j=k}^{B_c} c_j \equiv \sim \sum_{j=k}^{B_c} c_j$. Note that the symbol $\mathbb{1}$ denote the bitmap with all bits set to 1. As before, we call these two options as the direct option and the complement option for using the coarse level bitmaps. Similarly, there are also a direct option and a complement option involving the fine level bitmaps. Let m_i denote the size of b_i in number of words and n_i denote the size of c_i. The following table lists all four options for computing the answer to a one-sided range query using both levels of an equality-equality encoded index.

option	computation	I/O cost
1.	$\sum_{j=1}^{k-1} c_j + \sum_{j=d_{k-1}+1}^{i} b_j$	$\sum_{j=1}^{k-1} n_j + \sum_{j=d_{k-1}+1}^{i} m_j$
2.	$\sum_{j=1}^{k} c_j - \sum_{j=i+1}^{d_k} b_j$	$\sum_{j=1}^{k} n_j + \sum_{j=i+1}^{d_k} m_j$
3.	$\mathbb{1} - \sum_{j=k}^{B_c} c_j + \sum_{j=d_{k-1}+1}^{i} b_j$	$\sum_{j=k}^{B_c} n_j + \sum_{j=d_{k-1}+1}^{i} m_j$
4.	$\mathbb{1} - \sum_{j=k+1}^{B_c} c_j - \sum_{j=i+1}^{d_k} b_j$	$\sum_{j=k+1}^{B_c} n_j + \sum_{j=i+1}^{d_k} m_j$

When answering a query, sizes of bitmaps are known, therefore, we can determine which option requires the least amount of I/O time and then use it for answering the query. This can be easily accomplished in software by keeping the cumulative sizes, i.e., $\sum m_j$ and $\sum n_j$ separately. For our analytical evaluation, we observe that if $k < \lfloor B_c/2 \rfloor$, options 1 and 2 are likely to be more competitive than options 3 or 4. Next, we concentrate on computing the average cost for small k. The average value for $k > \lfloor B_c/2 \rfloor$ can be computed the same way and the average is in fact the same for uniform data.

Assume that C is an integer multiple of B_c, for uniform data, the size of a fine level bitmap is $m_i = m_r(1/C)$ and the size of a coarse level bitmap is $n_i = m_r(1/B_c)$. We prefer option 1 if $sum_{j=d_{k-1}+1}^{i} m_j \leq n_k + sum_{j=i+1}^{d_k} m_j$. That is, if $i \leq (\frac{m_r(1/B_c)}{m_r(1/C)} + d_{k-1} + d_k)/2$ option 1 is preferred. Let $\beta \equiv m_r(1/B_c)/\sum_{j=d_{k-1}+1}^{d_k} m_j$, we have

$$\beta = \frac{m_r(1/B_c)}{(d_k - d_{k-1}) m_r(1/C)} = \frac{B_c m_r(1/B_c)}{C m_r(1/C)}.$$

We can rewrite the condition on i as $i \leq (m_r(1/B_c)/m_r(1/C) + d_{k-1} + d_k)/2 = d_{k-1} + (1+\beta)(d_k - d_{k-1})/2 = (1+\beta)C/(2B_c)$. Define $\mu \equiv \lfloor (1+\beta)C/(2B_c) \rfloor$. The average cost involving the fine level bitmaps is

$$\begin{aligned}
u_{1RQ}^{EE} &\equiv \frac{1}{d_k - d_{k-1}} \left(\sum_{i=d_{k-1}+1}^{d_{k-1}+\mu} \sum_{j=d_{k-1}+1}^{i} m_j + \sum_{i=d_{k-1}+\mu+1}^{d_k} \left(n_k + \sum_{j=i+1}^{d_k} m_j \right) \right) \\
&= \frac{B_c}{C} \left(m_r(\frac{1}{C}) \sum_{i=1}^{\mu} i + \sum_{i=\mu+1}^{C/B_c} \left(m_r(\frac{1}{B_c}) + \left(\frac{C}{B_c} - i \right) m_r(\frac{1}{C}) \right) \right) \\
&= \frac{B_c}{2C} \left(\mu(\mu+1) m_r(\frac{1}{C}) + \left(\frac{C}{B_c} - \mu \right) \left(2 m_r(\frac{1}{B_c}) + \left(\frac{C}{B_c} - \mu - 1 \right) m_r(\frac{1}{C}) \right) \right).
\end{aligned}$$

The corresponding average cost involving the coarse level bitmaps is

$$v_{1RQ}^{EE} \equiv \frac{1}{\lfloor B_c/2 \rfloor} \sum_{k=1}^{\lfloor B_c/2 \rfloor} \sum_{j=1}^{k-1} c_j = \frac{\lfloor B_c/2 \rfloor - 1}{2} m_r(\frac{1}{B_c}).$$

The overall cost of answering queries with a_i, $i < C/2$, is given by $u^{EE}_{1RQ} + v^{EE}_{1RQ}$. For uniform data, the average cost involving a_i, $i > C/2$ (using option 3 an 4) should be the same as above. There are a small number of queries involving query boundaries a_i where i is close to $C/2$ and the costs of all four options might be close to each other. Because more options are involved, they may contribute to reduce the overall average cost. However, since there are only a few such cases, we simply take the overall average cost to be

$$t^{EE}_{1RQ} = u^{EE}_{1RQ} + v^{EE}_{1RQ}. \tag{29}$$

If B_c is large, such as one predicted by Equation 5, than $m_r(1/B_c) \approx 2N/B_c$. In this case, we have $\beta \approx 1$, $\mu \approx C/B_c$, and $u^{EE}_{1RQ} \approx \frac{1}{2}(\mu+1)m_r(1/C) \approx N(C/B_c+1)/C$. In other words, about a half of the fine level bitmaps in a coarse bin are accessed to answer an average query. Therefore, the coarse level bitmaps are not helpful in reducing the average query processing cost involving the fine level bitmaps.

The coarse level bitmaps can help reduce the cost involving the fine level bitmaps if B_c is small, say $B_c \leq 50$. In this case, we may assume the coarse level bitmaps are incompressible, e.g., $m_r(1/B_c) \approx N/(w-1)$. This leads to $\beta = \frac{B_c m_r(1/B_c)}{C m_r(1/C)} \approx \frac{B_c}{2(w-1)}$ and $\mu = \lfloor \frac{C}{2B_c} + \frac{C}{4(w-1)} \rfloor$. To further simplify the evaluation, we assume that $B_c > 5$ and drop the floor operators in the expressions. With all these simplifications, we obtain the following expression for the query processing cost.

$$u^{EE}_{1RQ} \approx \frac{N}{2B_c} + \frac{N}{2(w-1)} - \frac{B_c N(C-4(w-1))}{8C(w-1)^2}; \tag{30}$$

$$v^{EE}_{1RQ} \approx \frac{B_c/2-1}{2}\frac{N}{w-1} = \frac{(B_c-2)N}{4(w-1)};$$

$$t^{EE}_{1RQ} \approx \frac{N}{2B_c} + \frac{BN(C(2w-3)+4(w-1))}{8C(w-1)^2}. \tag{31}$$

The optimal number of coarse bins that minimizes the average query processing cost is

$$B_c = 2(w-1)\sqrt{\frac{C}{C(2w-3)+4(w-1)}}.$$

For Large C, the above expression can be approximated as $B_c = 2(w-1)/\sqrt{2w-3}$. For the typical word sizes $w = 32$ and $w = 64$, the corresponding B_c is 8 and 11. On a 32-bit system, using 8 coarse bins, the average cost is $1937N/15376 + 4N/(31C)$. Compared with the projection index which always accesses N words, the average cost of this encoding is about 8 times smaller for high cardinality attributes.

Two-sided range query In general, a two-sided range query will touch two edge bins instead of one as for the one-sided range query. The introduces more options for evaluating the query. There are two ways to evaluate each of the three sets of bitmaps, two set of fine level bitmap in two edge bins and one set of coarse level bitmaps; altogether there are eight options. We observed that within each edge bin, each fine bitmap is equally likely to be the end of our canonical two-sided range query. This means the previous analysis of the cost involving fine level bitmaps in each edge bin is valid for the two-sided range queries. If a two-sided range query involves two edge bins, the average cost involve the fine level bitmaps is twice as in the one-sided case, $u^{EE}_{2RQ} = 2u^{EE}_{1RQ}$. One out of every B_c two-sided range queries has two query boundaries in the same coarse bin. In these cases, the total query processing cost is u^{EE}_{1RQ}. The overall average cost involving the fine level bitmaps is

$$u^{EE}_{2RQ} = \frac{1}{B_c}u^{EE}_{1RQ} + \left(1-\frac{1}{B_c}\right)2u^{EE}_{1RQ} = \left(2-\frac{1}{B_c}\right)u^{EE}_{1RQ}.$$

To compute the cost associated with the coarse level bitmaps, we again divide the different case into two major categories that are relatively straightforward to analyze and neglect a smaller number of cases that are more complex. Given B_c coarse level bitmaps with equality encoding, we use these bitmaps to

compute either $\sum_{j=k_1+1}^{k_2-1} c_j$ or $\mathbb{1} - \sum_{j=1}^{k_1-1} c_j - \sum_{j=k_2+1}^{B_c} c_j$. The corresponding costs are either $\sum_{j=k_1+1}^{k_2-1} n_j$ or $\sum_{j=1}^{k_1-1} n_j + \sum_{j=k_2+1}^{B_c} n_j$. By construction, we have $n_j = m_r(1/B_c)$. This simplifies the two cost functions to be $(k_2-k_1-1)m_r(1/B_c)$ and $(B_c-k_2+k_1-1)m_r(1/B_c)$. It is clear that if $k_2-k_1 \leq \lfloor B_c/2 \rfloor$, the first cost function is smaller, otherwise the second one is smaller.

By our definition of the canonical two-sided range query, we can describe the values of k_1 and k_2 as follows. Take two integers from between 1 and B_c, call the smaller one k_1 and the larger one k_2. Among B_c^2 combinations of k_1 and k_2, there are B_c different possible values for k_2-k_1, $k_2-k_1=0$ appears B_c times, $k_2-k_1=1$ appears $2(B_c-1)$ times, $k_2-k_1=2$ appears $2(B_c-2)$ times, ..., and $k_2-k_1=B_c-1$ appears 2 times. The average cost involving the coarse level bitmaps is

$$\begin{aligned} v_{2RQ}^{EE} &\equiv \frac{1}{B_c^2}\left(\sum_{k=1}^{\lfloor B_c/2 \rfloor} 2(B_c-k)(k-1)m_r(\frac{1}{B_c}) \right. \\ &\qquad \left. + \sum_{k=\lfloor B_c/2 \rfloor+1}^{B_c-1} 2(B_c-k)(B_c-k-1)m_r(\frac{1}{B_c})\right) \\ &= \frac{m_r(1/B_c)}{3B_c^2}(\lfloor B_c/2 \rfloor(\lfloor B_c/2 \rfloor-1)(3B_c-2\lfloor B_c/2 \rfloor-2) \\ &\qquad +2(B_c-\lfloor B_c/2 \rfloor)(B_c-\lfloor B_c/2 \rfloor-1)(B-\lfloor B_c/2 \rfloor-2)) \end{aligned}$$

The overall cost for an average two-sided query is

$$t_{2RQ}^{EE} = u_{2RQ}^{EE} + v_{2RQ}^{EE}. \tag{32}$$

Assuming that B_c is between 5 and 50 and $C \geq 100$, we can use the same approximations that lead to Equation 30 to obtain a simplified expression for the average query processing cost for two-sided range queries as follows.

$$\begin{aligned} u_{2RQ}^{EE} &\approx \left(2-\frac{1}{B_c}\right)\left(\frac{N}{2B_c}+\frac{N}{2(w-1)}-\frac{B_cN(C-4(w-1))}{8C(w-1)^2}\right); \\ v_{2RQ}^{EE} &\approx \frac{(B_c-2)^2N}{4B_c(w-1)}; \\ t_{2RQ}^{EE} &\approx (2B_c-1)N\left(\frac{1}{2B_c^2}+\frac{1}{8(w-1)^2}\right)+\frac{4C+2B_c(B_c(4+C)-2)}{8B_cC(w-1)}N. \end{aligned} \tag{33}$$

Since B_c can only be integers between 5 and 50, it is easy to find the best values that minimize the query processing cost either by computing the cost for all possible B_c or using symbolic computing tools such as Mathematica or matlab. For $w=32$, the optimal B_c is 11. For $w=64$, the optimal $B_c=16$. When $C=10^6$ and $B_c=11$, $t_{2RQ}^{EE} \approx 0.174N$, which is about 1/6th of the cost of using the projection index.

7.3 Range-equality encoding

The key difference between this encoding and the equality-equality encoding is that the coarse level bitmaps are constructed using range encoding. As in the previous case, we can compute the index size by considering the fine level and the coarse level separately.

7.3.1 Index size

This encoding can be thought of as constructed from equality-equality encoding. Let b_i denote the bitmaps from equality encoding. Range encoding computes a set of bitmaps by ORing b_i progressively, where the ith new bitmap is computed as $\sum_{j=1}^{i} b_i$. The total size of coarse level bitmaps can be computed in the same way as the total sizes of bitmaps are computed in Section 5.4.

For the synthetic data we are considering, each coarse bin is assumed to have $1/B_c$ of the possible values. Under this assumption, the total size of range encoded bitmaps is given by $\sum_{i=1}^{B_c-1} m_r(i/B_c)$. As explained in Section 5.4, because more than 90% of the bitmaps are incompressible, their total size can be approximated with $(B_c-1)N/(w-1)$. In fact, the same approximation is valid for the Markovian data if the clustering factor f is modest, say $f \leq 10$.

For uniform random data, the total size of a range-equality encoded index is

$$s_U^{RE} = Cm_r(1/C) + \sum_{i=1}^{B_c-1} m_r(i/B_c) \approx 2N + (B_c-1)N/(w-1). \tag{34}$$

Note that the above approximation is accurate for large C, say $C \geq 100$.

7.3.2 Query processing cost

One-sided range query As with the equality-equality encoding, we can evaluate the cost involving the coarse level and the fine level separately. Following the analysis given in Section 5.4, we only need one coarse level bitmaps to answer a one-sided range query. Let c_i denote the ith coarse level bitmap and b_i denote the ith fine level bitmap, the solution to query "A $\leq a_i$" can be to computed as $c_{k-1} + \sum_{j=d_{k-1}+1}^{i} b_j$, where the values d_i are coarse bin boundaries and $d_{k-1} < i < d_k$. As before, there is a complement option which can be expressed as $c_k - \sum_{j=i+1}^{d_k} b_j$. The operation $a - b$ is define to be $a\& \sim b$.

Since the c_{k-1} and c_k are about the same size, we can choose to use either the direct option or the complement option according to how many bytes are to read read from disk. This minimizes the overall query processing cost. On average, the cost of answering a one-sided range query is equivalent to reading one coarse level bitmap and one quarter of the fine level bitmaps in a coarse bin. For uniform random data, we can express the average cost of processing a one-sided range query as follows,

$$t_{1RQ}^{RE} = \frac{Cm_r(1/C)}{4B_c}(1 - B_c/C) + \frac{1}{B_c}\sum_{i=1}^{B_c-1} m_r(i/B_c) \approx \frac{N}{2B_c} + \frac{N}{w-1}. \tag{35}$$

Similar to the approximate in Equation 29, the above approximate is accurate for large C and small B_c.

Two-sided range query The answer to a two-sided range query can be effectively constructed from answers to two one-sided range queries. In other words, "$a_1 \leq$ A $\leq a_2$" is equivalent to "A $\leq a2$ AND NOT A $< a_1$." This leads to the following expression for the average query processing cost.

$$t_{2RQ}^{RE} = 2t_{1RQ}^{RE} \approx \frac{N}{B_c} + \frac{2N}{w-1}. \tag{36}$$

Since there are two options to compute the answer to a one-sided range query, there are a total of four options for a two-sided range query. Considering that there are two additional options of answering the same query using the fine level bitmaps only, altogether there are six different ways to answer a two-sided range query using the range-equality encoding. Our test software implements all these six options.

7.4 Interval-equality encoding

An interval-equality encoded index can be constructed from an equality-equality encoded index by recomputing the coarse level bitmaps using interval encoding. Because interval encoding produces about half as many bitmaps as range encoding, interval-equality encoding may produce a more compact index than the range-equality encoding. In fact, an interval-equality encoded index may even be smaller than an equality-equality encoded index.

7.4.1 Index size

For B_c coarse level bins, interval encoding produces $B_c + 1 - \lceil B_c/2 \rceil$ bitmaps, where each of them has the bit density of $\lceil B_c/2 \rceil / B_c$. This is the case for all three types of synthetic data considered so far. Following the same analysis that leads to Equation 51, we can compute the clustering factor of the bitmaps to be $f(1 - 1/C)/(1 - \lceil B_c/2 \rceil / B_c) \approx 2f$. From the earlier analysis, we know that such bitmaps can not be compressed with WAH compression.

For uniform random data, the total size of an interval-equality encoded index is

$$s_U^{IE} = Cm_r(1/C) + (B_c + 1 - \lceil B_c/2 \rceil) m_r(\lceil B_c/2 \rceil / B_c) \approx 2N + \frac{B_c N}{2(w-1)}. \tag{37}$$

Note that the above approximation is accurate for large C and modest B_c, say $C \geq 100$ and $B_c \leq 50$.

7.4.2 Query processing cost

As illustrated in the equality-equality encoding and the range-equality encoding cases, the process of answering a range query (either one-sided or two-sided) involves using the coarse level bitmaps to answer the query approximately and then use the fine level bitmaps to make the answer accurate. Since the procedure involving the fine level bitmaps are the same as before, we only need to consider how the interval encoded coarse level bitmaps can be used to answer a range query.

In [7], the authors have thoroughly explained how to answer a range query using the interval encoded index. Some of the dominant cases have also been reviewed in Section 5.5. The key observation is that it requires two bitmaps to answer any range query, be it an equality query, one-sided range query or two-sided range query. Because all coarse level bitmaps are likely to be the same size, the average cost of processing either a one-sided range query or a two-sided range query is

$$t_{2RQ}^{IE} \approx \frac{N}{B_c} + \frac{2N}{(w-1)}. \tag{38}$$

7.5 Optimal number of coarse bins

The optimal number of coarse bins for the equality-equality encoding can be computed by minimizing the query processing cost as we have done. Recall that for two-sided range queries, the optimal number of coarse bins is 11 on a 32-bit system. However, without considering compression, the optimal number of coarse bins would have been 1,400 according to Equation 5. With $B_c = 1400$, the equality-equality encoding is no more efficient than the basic bitmap index, however, with $B_c = 11$, the equality-equality encoding answers a query with 1/3th the cost of the one-component equality encoding ($\frac{0.5N}{0.174N}$). This advantage also motivates us to find the optimal number of coarse bins for other two-level encodings.

The optimal choice for the number of coarse bins for the range-equality encoding and the interval-equality encoding cannot be produced by minimizing the index size or the query processing cost, because those functions are monotonic functions of the number of coarse bins. In the test software to be used later, we use the following choices. For the range-equality encoding and the interval-equality encoding, we choose the number of coarse bins to make the query processing cost involving the coarse level to be the same as that involving the fine level. We choose B_c based on the cost for processing two-sided range queries as shown in Equations 36 and 38. In the case of range-equality encoding, see Equation 36, the cost involving the coarse level is $2N/(w-1)$ and the cost involving the fine level is N/B_c. The choice of

$$B_c = (w-1)/2 \tag{39}$$

makes the costs due to two levels the same.

It is possible to minimize any convex function of the size and time to obtain a choice for the number of coarse level bins. For example, minimizing $s_U^{RE} t_{2RQ}^{RE}$ leads to $B_c = \sqrt{(w-1)(2w-3)/2}$. For $w = 32$,

	binary	equality	equality-equality	range-equality	interval-equality
size	$\frac{N\log_2(C)}{w-1}$ [24]	$2N$ [7]	$2.35N$ [28]	$2.48N$ [34]	$2.29N$ [37]
EQ	$\frac{N\log_2(C)}{w-1}$ [25]	$\frac{2N}{C}$ [9]	$\frac{2N}{C}$ [9]	$\frac{2N}{C}$ [9]	$\frac{2N}{C}$ [9]
$1RQ$	$\frac{N\log_2(C)}{w-1}$ [26]	$\frac{N}{2}$ [11]	$0.129N$ [29]	$0.064N$ [35]	$0.095N$ [38]
$2RQ$	$\frac{N\log_2(C)}{w-1}$ [27]	$\frac{N}{2}$ [13]	$0.174N$ [32]	$0.095N$ [36]	$0.095N$ [38]

Table 3: Index sizes and query processing costs of five encodings on uniform random attributes with large attribute cardinalities. The equations defining the more general cases are given in the square brackets.

$B_c = 31$. Alternatively, we may minimize $s_U^{RE} + t_{2RQ}^{RE}$, which leads to a choice of $B_c = \sqrt{w-1}$. For $w = 32$, this suggest $B_c = 6$. Obviously, other functions of s_U^{RE} and t_{2RQ}^{RE} will lead to different choices for B_c between 6 and 31. We decide to use the formula in Equation 39 because it is in the middle of a range of possible choices. Since the average query processing cost for the interval-equality encoding is the same as that for the range-equality encoding, Equation 39 is also used for determining the number of coarse bins for the interval-equality encoding.

Based on our analysis of two-level encodings, the number of coarse level bins is quite small, therefore, there is no need to use another coarse level.

8 Optimal Encodings

In last three sections, we studied the performance characteristics of different encodings in isolation. Next, we summarize the key results and select some of them for further comparisons. In addition, we also point out the optimal scaling property of the three two-level encodings.

Among the multi-component methods, we identify two methods that are promising, the one-component equality encoding and the binary encoding. The query processing cost using a multi-component encoding generally increases as the number of component increases. A notable exception is the binary encoding, which because of its special design is able to keep its index size small and query processing cost low. It is implemented in some commercial DBMS products [17, 21]. So is the one-component equality encoding [2, 20]. Therefore, comparisons against these method are of practical interest to a large number of users.

For high-cardinality attributes, the one-component range encoding and interval encoding produce indexes that can be much larger than the base data, but their sizes decrease as the number of components increases. The versions that use the maximal number of components would have all bases of size 2, which can be optimized and turned into the binary encoding. In this regard, the one-component equality encoding and the binary encoding represent the two extremes of multi-component encodings, where the first uses the minimal number of components and the second uses the maximal number of components.

Among the multi-level encodings, we show that the optimal number of levels is 2. Among the two-level encodings, equality-equality encoding, range-equality encoding and interval-equality encoding produce compact indexes and also have low query processing costs.

In Table 3, we summarize the performance characteristics of the five encodings identified above. We only consider the case of uniform random attributes with high attribute cardinalities because the high-cardinality attributes are considered challenging for bitmap indexes, and uniform random data are among the hardest for WAH compression method to compress.

The specific numbers given in Table 3 are for $w = 32$, which is the word size on our test system. The rows labeled EQ, $1RQ$ and $2RQ$ are the query processing costs, which are defined to be the number of words accessed in order to answer an average query of the particular type. Currently, the basic bitmap index (marked equality in Table 3) and the binary encoded index are considered efficient indexes and used in commercial database systems. Through our analyses, we see that the two-level encodings can significantly outperform these two. For two-sided range queries ($2RQ$), the average query processing cost using the

Table 4: Labels of bitmap encoding schemes used in tests.

BN	binary encoding
E1	one-component equality encoding (the basic bitmap index)
EE	equality encoded coarse level, equality encoded fine level
RE	range encoded coarse level, equality encoded fine level
IE	interval encoded coarse level, equality encoded fine level

range-equality encoding or the interval-equality encoding is about 1/5th of that for the basic bitmap index. The query processing cost using the equality-equality encoding is about 1/3rd of that for the basic bitmap index.

Based on the query processing costs shown in Table 3, the range-equality encoding and interval-equality encoding would be faster than the binary encoding if the attribute cardinality is larger than five. This means that for most attributes, the two-level encodings are preferred. On attributes with cardinality 1 million, $\log_2(C) = \log_2(10^6) \approx 20$, the range-equality encoding and the interval-equality encoding are almost seven times faster at answering two-sided range queries than the binary encoding. The performance advantages of these two-level encodings is even more pronounced on one-sided range queries and equality queries.

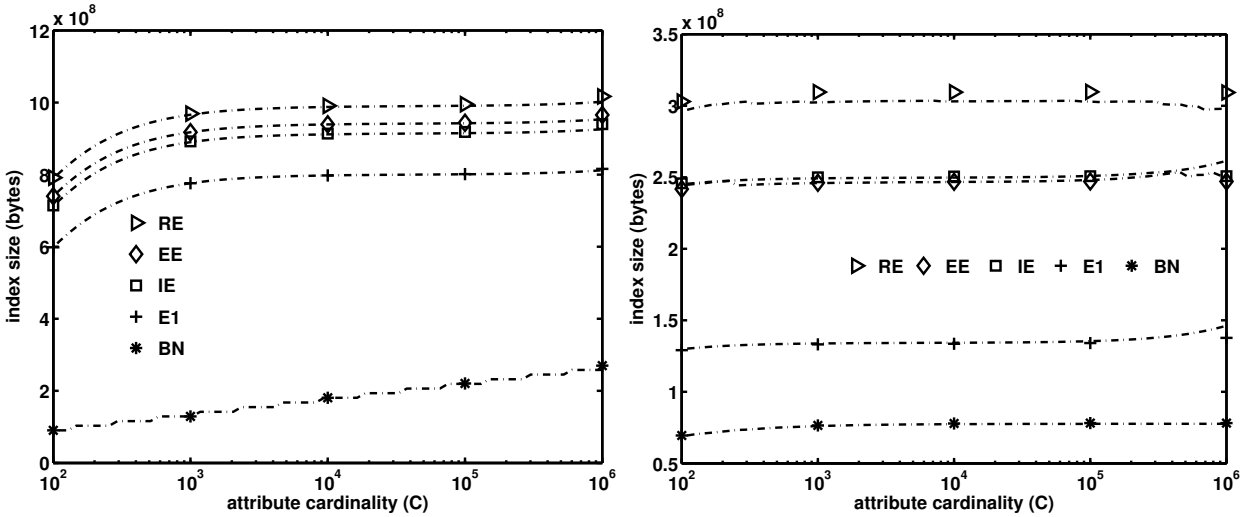
When a bitmap index is used to answer a query, a bitmap representing the selected records is produced. Let h denote the number of hits. Since each hit is identified, the process to identify and record the hits requires a minimal of $O(h)$ time. Any indexing method that can answer a query in $O(h)$ time is said to be optimal.

The three two-level encodings not only perform well in terms of average query processing cost, they are also optimal indexes because they answer queries on $O(h)$ time. In [34], the authors proved that the worst-case time complexity of the basic bitmap index compressed with WAH has $O(h)$ time complexity. By construction, the three two-level encodings contain the basic bitmap index at the fine level, and they never cost more than the basic bitmap index to answer any query. Therefore, equality-equality encoding, range-equality encoding and interval-equality encoding also have $O(h)$ time complexity.

9 Observed Performance

In this section we measure the size and the query processing costs of the five multi-level and multi-component bitmap encoding schemes identified in the previous section. A brief description of these encoding schemes and their short-hand notations are listed in Table 4. We used 100 million records with five different attribute cardinalities ranging from 100, 1000, to 10^6. For each cardinality, we also vary the data distribution by using three different Zipf exponents 0, 1 and 2, and three different clustering factors, 2, 4, and 8. The attributes are named cmzz or cmff, where m indicates the attribute cardinality being 10^m, z is the Zipf exponent, and f is the clustering factor. For example, the attribute named c5z1 has an attribute cardinality of 10^5 and a Zipf exponent of 1; c6f2 has an attribute cardinality of 10^6 and a clustering factor of 2. All our performance benchmarks were carried out on a 2.8 GHz Intel Pentium 4 with 2 GB of memory. The I/O subsystem is a RAID 0 disk array consisting of 4 disks with a measured I/O rate of some 80 MB/sec.

We present the index sizes before the query processing costs. The discussions on query processing cost progress from the most detailed query response time of each query, to the average cost per attribute, and finally to the average cost over the entire data set, which lead eventually to a high-level comparison among the five selected encodings.



(a) Uniform data (c*mz*0) (b) highly skewed data (c*mz*2)

Figure 9: Sizes of compressed bitmap indexes. Dash-dotted lines are predictions.

9.1 Index Size

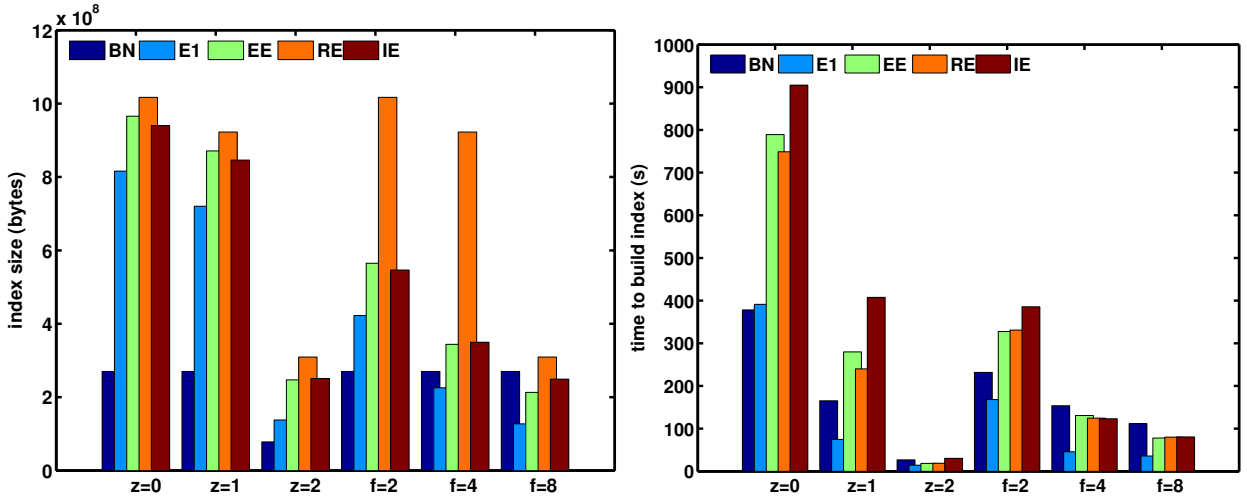
The sizes of the compressed bitmap indexes are shown in Figure 9 along with the predictions made in earlier sections. Each symbol shows the index size for one attribute and the dash-dotted lines are the predictions. There are five symbols on each line representing five different attribute cardinalities ranging from 100 to 10^6. Figure 9(a) shows the index sizes of uniform random data (attributes c*mz*0, i.e., those with Zipf exponents $z = 0$) and Figure 9(b) shows the index size of highly skewed data (attributes c*mz*2, i.e., Zipf exponents $z = 2$). In both cases, the actual measured index file sizes represented by the symbols fall on top of the prediction lines, indicating that the actual values agree with predictions.

The bitmap indexes used here are built with *no binning*. The basic bitmap index contains C bitmaps. With WAH compression, these bitmaps take about $2N$ words for large C. In our tests, N is 10^8 and a word has four bytes, therefore, the index size should approach 8×10^8 bytes as the attribute cardinality C increases. The sizes of E1 indexes indeed approach the expected value of 8×10^8 bytes.

The binary encoding (BN) produces the smallest indexes in each case and the range-equality encoding produces the largest index among the five encodings tested. Since the three two-level encodings contain the basic bitmap index at the fine level, their sizes are always larger than the basic bitmap indexes. Because the coarse level bitmaps are hard to compress, each coarse level bitmap is about the same size no matter what encoding is used. For this reason, the encoding that produces less bitmaps typically also produces smaller indexes. Based on the parameters described in the previous section, encoding IE produces 9 coarse level bitmaps, encoding EE produces 11 coarse level bitmaps and encoding RE produces 15 coarse level bitmaps. We observe that the RE indexes are indeed the largest. Because the equality encoded coarse level bitmaps produced by encoding EE are slightly more compressible than other coarse level bitmaps, EE encoded indexes are about the same as RE encoded ones in some cases.

Figure 9(a) and Figure 9(b) show data with different Zipf exponents. As the Zipf exponent increases, the data distribution becomes more skewed and the index size decreases as well. The coarse level bitmaps are hard to compress. This keeps the differences between the two-level encodings and the E1 encoding about the same in Figure 9(a) and Figure 9(b). On highly skewed data (with Zipf exponents $z = 2$), the relative difference between the two-level indexes and the one-level equality encoded index is larger. For example, the size of the IE encoded index is about 15% larger than the E1 index on c6z0 (where $C = 10^6$ and $z = 0$), but their relative difference increase to about 82% on c6z2 (where $C = 10^6$ and $z = 2$).

Figure 10(a) shows how the index sizes change with the Zipf exponent and the clustering factor. In this case the attribute cardinality is fixed at 10^6. This bar graph contains six groups for six different attributes



(a) Measured index sizes (b) Index building time

Figure 10: Sizes of compressed bitmap indexes and time needed to build the indexes. Attribute cardinality $C = 10^6$.

with $C = 10^6$. Within each group, the five bars correspond to five different encodings. The bars are ordered from left to the right in the same order as the legends.

For four out of six attributes, encoding BN produces the smallest indexes, while in the other two cases, E1 produces the smallest indexes. In terms of total sizes, indexes produced by BN is smaller. Another trend we observe is that the index sizes produced by BN is about the same in five out six cases. This is because BN usually produces bitmaps that are incompressible as expected. In all other cases, the index size decreases as the Zipf exponent or the clustering factor increases. The index sizes produced by E1 show the largest benefit as the Zipf exponent or the clustering factor increases.

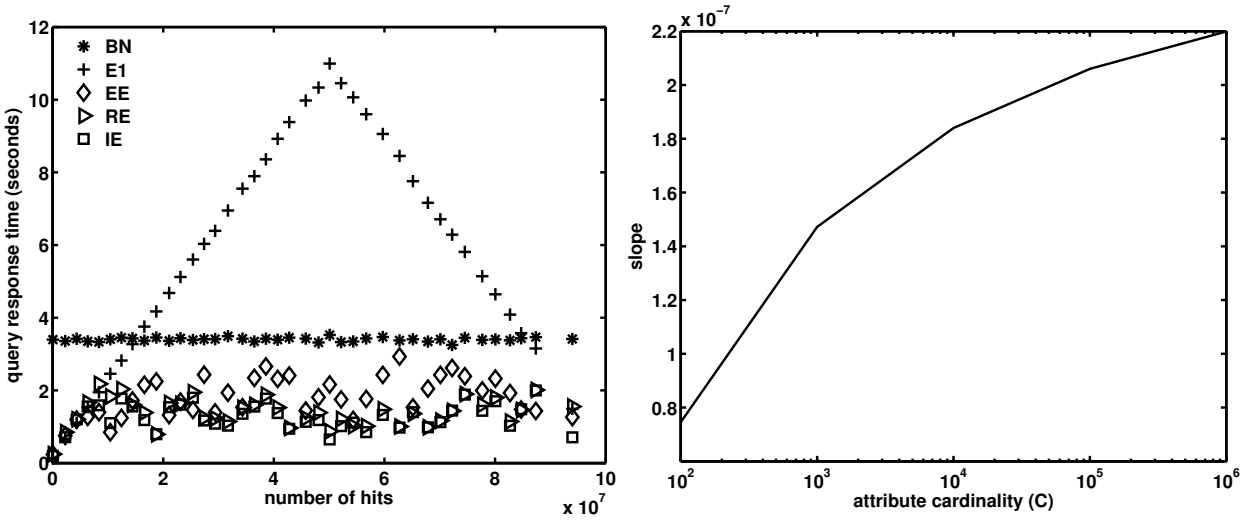
Figure 10(b) shows the time used to build the indexes for the attributes with cardinality 10^6. Generally, the larger indexes require more time to build. This is primarily because the current implementation uses the dynamic arrays from Standard Template Library to hold the words used in the compressed bitmaps. The larger indexes generally contain many large bitmaps. While creating the indexes, these bitmaps grow in size and have to be frequently reallocated to accommodate the increasing space requirement. Avoiding these dynamic memory allocation could reduce the variation in index creation time.

We conclude the discussion on index sizes by observing that all five encodings produce indexes that are no more 3 times the base data size. In our tests, the one attribute of the base data takes 4×10^8 bytes. In the worst case, shown in Figures 9(a) and 10(a), the observed index sizes are less than 12×10^8 bytes. Since the popular B-tree indexes are observed to take 3 – 4 times the size of base data, we regard all five of them as using acceptable amount of space.

9.2 Scaling with the number of hits

In Section 8, we show that four of the five encodings tested can answer queries in $O(h)$ time, which means in the worst case, the time required to answer a query is bounded by a linear function of the number of hits h. In Figure 11(a), we plot the time used to answer a query against the number of hits. In this case, we show only the time values measured with c6z0 because queries on the uniform random data take the longest time to answer. To make sure the full I/O cost is accounted for, we unmount the file system containing the data files and the index files. We measure the I/O cost through vmstat command, and measure the time through gettimeofday which reports the elapsed time in seconds. The queries we use here are subsets of canonical two-sided range queries. Each query set contains 300 randomly selected two-sided range queries. The queries we use only count the number of hits, but do not retrieve any records.

In Figure 11(a), the points marked E1 form a straight line when the number of hits is less than 50



(a) E1 scales linearly (b) slope approaches a constant

Figure 11: The query response time (seconds) plotted against the number of hits on uniform data.

million. For queries with more than 50 million hits, the query response time decreases as the number of hits increases. This agrees with prediction that the query processing time is bounded by a linear function of the number of hits. However, for the predicted linear relation to hold, the slope of the line formed by E1 must not depend on characteristics of the data such as the attribute cardinality. Wu et al. [34] gave extensive argument indicating the slope approaches a constant as attribute cardinality increases, here we give some empirical support. In Figure 11(b), we show how the slope of the timing curve varies with the attribute cardinality. We observe that as the attribute cardinality increases, the slope indeed approaches a constant. Therefore, the WAH compressed basic bitmap index does scale as $O(h)$.

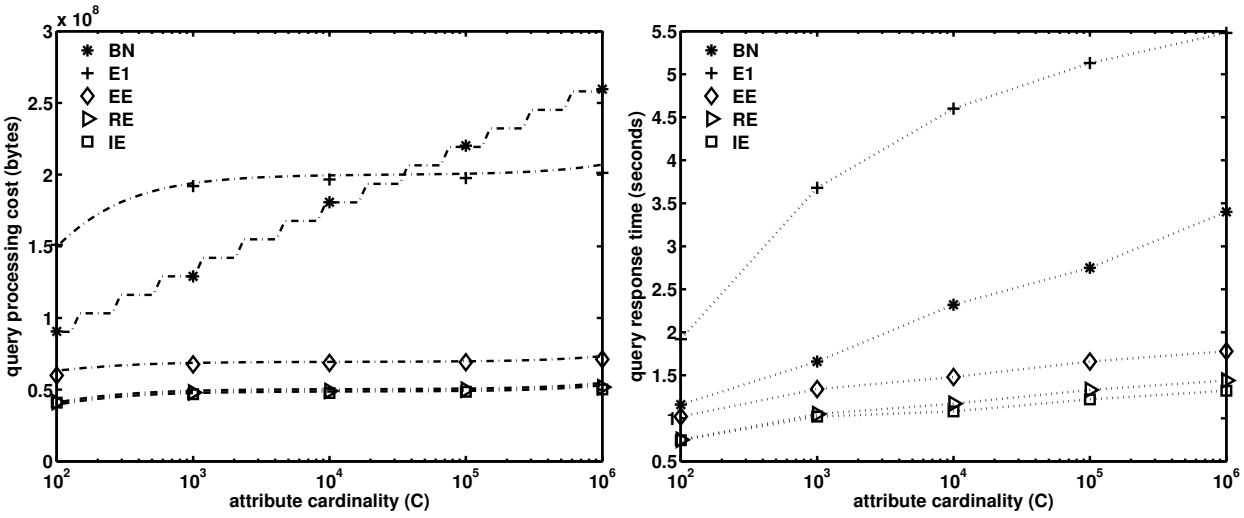
As expected, the time used by the two-level indexes are never more than that of the basic bitmap index (E1). In fact, when the number of hits is close to 50 million ($N/2$), the two-level methods are 10 times faster. Figure 11(a) shows that the query response time of the three two-level indexes are close to that of E1 if the number of hits is less than 10 million. Therefor, their query response time is bounded by the same linear function for E1. This verifies that the three two-level indexes are not only faster but also theoretically optimal.

9.3 Average cost per attribute

The next set of measurements are averaged over all random two-sided range queries for each attribute. We measure these both the I/O cost and the query response time.

Figure 12(a) shows the actual query processing costs against their predicted values, where each symbol represent the average I/O cost over 300 random queries on a single attribute, and the dashed lines represent the predictions computed earlier. Each prediction line has five symbols on them representing five attributes with different attribute cardinalities. We observe that the predictions match the actual observed values well. Though we only show the case for the uniform data, similar agreement between predicted costs and actual costs is also observed for other attributes. This agreement indicates that our analyses captured the most important I/O costs involved in answering a query.

Figure 12(b) shows the average query response time used for each attribute of the test data set. We choose to show the time values involving the uniform data to reduce the clutter of the graph. Overall, we see that that the basic bitmap index (marked with E1) requires the longest time. The three two-level encodings are faster than both the basic bitmap index and the binary encoded index. Among the three two-level encodings, encoding EE takes longer time to answer a query than encoding RE and IE. Time used by encoding RE and IE are nearly the same, with IE uses slightly less time in some cases.



(a) I/O cost

(b) query response time (seconds)

Figure 12: The average cost to answer random two-sided range queries on uniform random data.

Comparing Figures 12(a) and (b), we see that the relative differences among the three two-level encodings are about the same, but the same is not true between E1 and BN. According the I/O costs, E1 should be more efficient than BN for attributes with $C > 2^{(w-1)/2} = 32768$. However, E1 always takes more time to answer a query than BN no matter what is the attribute cardinality. This is primarily due to two reasons. Using encoding E1, we need to perform $C/4$ operations on compressed bitmaps on the average. This requires considerable amount of CPU time. In many cases, the CPU time is close to the time required for I/O operations using encoding E1, while the CPU time using encoding BN is typically negligible. This means that the total elapsed time using E1 is about twice that using BN as shown in Figures 12(b). The second reason that using E1 takes more time than using BN is that it requires more read operations using E1 than using BN. In our test software, we always reads a binary encoded index in one read operation, while to answer a query using E1, we need at least two read operations, once to read the metadata and once to read the bitmaps involved. This introduces more I/O overhead. The three two-level encodings require even more I/O operations than E1, but they require less CPU time than E1. When $C = 10^6$, the average query response time using IE is about 1/3rd of that using BN even though the I/O cost of IE is about 1/5th of that of BN.

9.4 Relative speed

To measure the relative speed of the five encodings, we compute their average speedup against a simple base line, namely the projection index [21]. The projection index is efficient for ad hoc range queries used in our tests [21], and it requires a simple fixed time, about 4.88 seconds to answer a query on our test system, which make the comparisons simple. Figure 13 shows the speedup of the five different bitmap indexes over the projection index. We compute the speedup value based on the average time to answer 300 queries on an attribute, and each bar in this figure represents the average of speedup values for five different attributes with the same Zipf exponent or clustering factor, but different attribute cardinalities. There are six groups of bars representing six types of attributes. Within each group, the five encodings are shown from left to right in the same order as they are listed in top of the figure.

From Figure 13, we see that as the Zipf exponent increases the speedup value also increases. Similarly, as the clustering factor of the uniform Markov process increases, the speedup value also increases. Encoding E1 shows the largest increase; its speedup value goes from just above 1 for $z = 0$ to above 30 for $z = 2$. In contrast, encoding BN shows smallest increase; its speedup value goes from about 2 for $z = 0$ to about 4 for $z = 2$. The three two-level encodings are generally faster E1, but the relative differences decrease as the Zipf exponent increases or the clustering factor increases. The average speedup values over all queries are

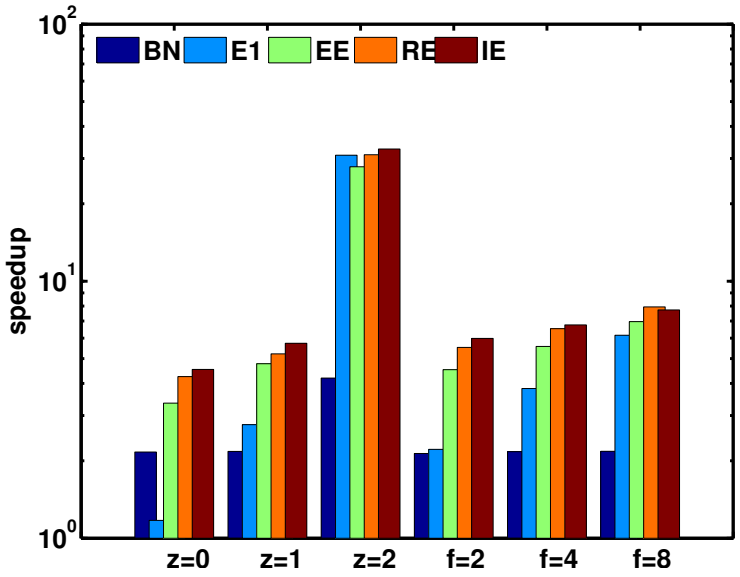


Figure 13: Speedup values of bitmap indexes against the projection index.

2.5 for BN, 7.8 for E1, 8.8 for EE, 10.1 for RE and 10.6 for IE.

9.5 Space-time trade-off

We summarize all measurements in a graph plotting the overall average query response time against the overall average index sizes in Figure 14. In this figure, the horizontal axis shows the average index size overall all 30 attributes in our synthetic data set, and the vertical axis shows the average query response time over all 9000 queries (300 queries $\times$ 30 attributes). In Figure 14, we show five symbols, one for each of the five encodings tested. We observe that the overall average index size of IE is about 3 times that of BN, but the query response time of BN is about exactly 3 times that of IE. The dashed line passes through all points where increasing the index by a factor of x reduces the query response time by a factor of $1/x$. If we regard this line as representing a *fair* space-time trade-off, on our particular data set, BN and IE have this fair trade-off. Encoding IE has the best space-time trade-off among the three two-level encodings.

Comparing IE with E1, we observe that IE uses about 1/3rd more space (4.8×10^8 bytes vs. 3.6×10^8 bytes), but it reduces the query response time by a factor of about 2.4 (0.71 seconds vs. 1.7 seconds). Clearly, the two-level interval-equality encoding (IE) has a favorable space-time trade-off compared with the basic bitmap index (E1).

10 Summary and Future Work

One important contribution of this paper is the development of precise closed-form formulas that predict the index sizes and query processing costs of compressed bitmap indexes. These formulas are highly accurate as shown by extensive experimental measurements. Therefore, they are very useful for query planning and cost estimation. In addition, through the analyses we also discover the correct parameters that significantly improve the performance of multi-level indexes.

Among the multi-component indexes, we identify the bit-sliced index [21, 31] and the basic bitmap index [20] as the best. Without compression, Chan and Ioannidis [6] showed that the optimal number of components was two. In this paper, we prove that using either the minimal number of components (such as the one-component equality encoding that produces the basic bitmap index) or the maximum number of components (i.e., the binary encoding that produce the bit-sliced index) minimizes the index size. It is possible for the one-component range encoding and interval encoding to use less time to answer a range

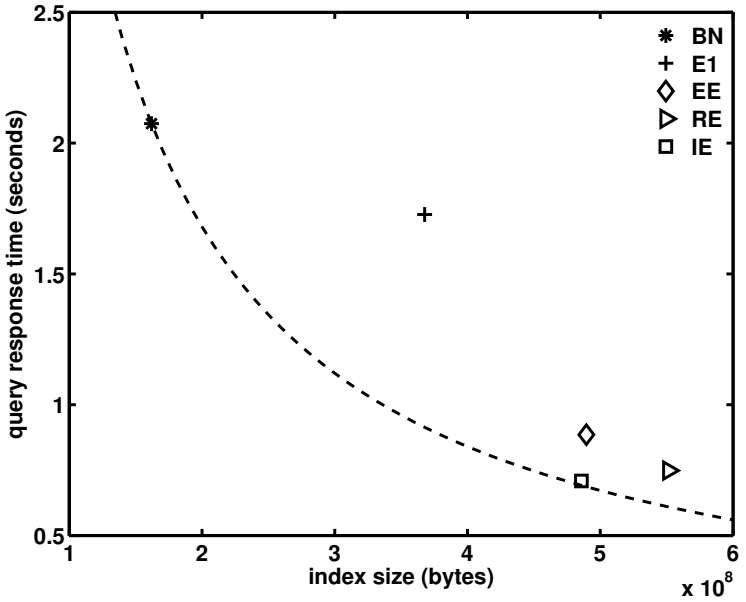


Figure 14: The average query response time (in seconds) plotted against the average index sizes (in bytes) over all test data. On the dashed line that goes through BN and IE, the product of the index size and the query response time is the same.

query, but their index sizes can be arbitrarily large as shown in Equations 15 and 19. For example, for a 4-byte integer attribute with 1 million distinct values, a range encoded index could be more than 30,000 times larger than the base data. Therefore, they are not suitable as general-purpose indexing methods. Using more than one component reduces the numbers of bitmaps needed, and therefore index sizes. However, due to its unique construction, the binary encoding has many advantages over all other multi-component encodings, with the only possible exception of the one-component equality encoding which is highly amenable to compression. We include both the bit-sliced index and the basic bitmap index in our experimental study also because they are implemented in commercial products.

Among the multi-level encodings, we show that the optimal number of levels is two. Among the two-level encodings, the best ones all use equality encoding at the fine level. They are equality-equality encoding [35, 27], range-equality encoding [35], and interval-equality encoding. Through our analyses, we are able to compute the optimal number of coarse bins to use in each case. Both analyses and timing measurements show that using the optimal number of coarse bins can significantly improve their performance. Because the optimal numbers of coarse level bins are small (11 – 16), there is no need to use more than two levels. Without considering compression, the predicted number of coarse level bins is much larger and using more than two level might have been useful.

We prove that four out of the five above encodings (except the binary encoding) are theoretically optimal because they require $O(h)$ time to answer a query, where h is the number of hits of a query. Some of the multi-level indexes have been studied before, however, we are the first to identify the appropriate parameters to ensure that queries can be answered in $O(h)$ time. We not only prove this in theory, but also verify it through experimental measurements.

In addition to being theoretically optimal, the two-level encodings also answer queries much faster than both the bit-sliced index and the basic bitmap index. In terms of overall average query response time, the two-level interval-equality encoding is about 3 times faster than the binary encoding and 2.4 times faster than equality encoding. In some cases, the two-level indexes are 10 times faster than the two multi-component methods.

The two-level indexes generally use more space than the bit-sliced index and the basic bitmap index. The overall average index size with the two-level interval-equality encoding is about 1/3rd larger than the

basic bitmap index, and about 3 times larger than the bit-sliced index. However, compared with the popular B-tree indexes, these two-level indexes are still relatively small. For example, the average size of all interval-equality encoded indexes in our tests is about 1.2 times the base data size, while B-tree indexes are observed to be 3 – 4 times the base data size.

Overall, the binary encoding produces the most compact indexes, while interval-equality encoding costs the least to answer an average range query. Therefore, the binary encoding is more suitable if the disk space is very limited, while interval-equality encoding is the best if disk space is less of a concern. In most applications where B-tree indexes are currently used, switching to one of the five bitmap indexes reduces both the disk usage and the query response time. In this case, we recommend using interval-equality encoding.

In this paper, we analyze encoding methods with WAH compression. Our analyses concentrate on random data, which produce bitmap indexes that are the hardest for WAH to compress. In this regard, we have studied the worst case scenario — all indexes studied will perform no worse on real application data than on random data. However, it is possible that their relative performances may differ on a set of application data than on random data, or the optimal number of coarse bins may differ. Considering an alternative compression method could be of interest when the attribute cardinality C is close to N, because WAH compression has a considerable overhead in this case. We also did not consider how binning the base data might affect the overall query response time. These are all interesting topics for future work.

11 Acknowledgements

The authors gratefully acknowledge the help received from Dr. Ekow Otoo and Professor Patrick O'Neil for reviewing this manuscript.

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

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

[2] G. Antoshenkov and M. Ziauddin. Query processing and optimization in ORACLE RDB. *VLDB Journal*, 5:229–237, 1996.

[3] Gunter Bloch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains*. John Wiley & Sons, New York, NY, 1998.

[4] A. Bookstein and S. T. Klein. Using bitmaps for medium sized information retrieval systems. *Information Processing & Management*, 26:525–533, 1990.

[5] Abraham Bookstein, Shmuel T. Klein, and Timo Raita. Simple bayesian model for bitmap compression. *Information Retrieval*, 1(4):315–328, 2000.

[6] C.-Y. Chan and Y. E. Ioannidis. Bitmap index design and evaluation. In *Proceedings of the 1998 ACM SIGMOD: International Conference on Management of Data*, pages 355–366, New York, NY, USA, 1998. ACM press.

[7] C. Y. Chan and Y. E. Ioannidis. An efficient bitmap encoding scheme for selection queries. In Delis et al. [10], pages 215–226.

[8] Surajit Chaudhuri, Umeshwar Dayal, and Venkatesh Ganti. Database technology for decision support systems. *Computer*, 34(12):48–55, December 2001.

[9] Douglas Comer. The ubiquitous B-tree. *Computing Surveys*, 11(2):121–137, 1979.

[10] A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors. *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, New York, NY, USA, 1999. ACM Press.

[11] C. Faloutsos and S. Christodoulakis. Signature files: An access method for documents and its analytical performance evaluation. *ACM Trans. of Information Systems*, 2(4):267–288, 1984.

[12] Ying Hu, Seema Sundara, Timothy Chorma, and Jagannathan Srinivasan. Supporting RFID-based item tracking applications in oracle DBMS using a bitmap datatype. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 1140–1151. ACM, 2005.

[13] Theodore Johnson. Performance measurements of compressed bitmap indices. In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 278–289, San Francisco, 1999. Morgan Kaufmann. A longer version appeared as AT&T report number AMERICA112.

[14] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison Wesley, 2nd edition, 1998.

[15] Nick Koudas. Space efficient bitmap indexing. In *Proceedings of the ninth international conference on Information knowledge management CIKM 2000 November 6 - 11, 2000, McLean, VA USA*, pages 194–201. ACM, 2000.

[16] X. Lin, Y. Li, and C. P. Tsang. Applying on-line bitmap indexing to reduce counting costs in mining association rules. *Information Sciences*, 120(1-4):197–208, 1999.

[17] Roger MacNicol and Blaine French. Sybase IQ multiplex - designed for analytics. In Nascimento et al. [19], pages 1227–1230.

[18] A. Moffat and J. Zobel. Parameterised compression for sparse bitmaps. In N. Belkin, P. Ingwersen, and A. M. Pejtersen, editors, *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval, Copenhagen, June 1992*, pages 274–285. ACM Press, 1992.

[19] Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors. *Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*. Morgan Kaufmann, 2004.

[20] P. O'Neil. Model 204 architecture and performance. In *2nd International Workshop in High Performance Transaction Systems, Asilomar, CA*, volume 359 of *Lecture Notes in Computer Science*, pages 40–59. Springer-Verlag, September 1987.

[21] P. O'Neil and D. Quass. Improved query performance with variant indices. In Joan Peckham, editor, *Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 38–49. ACM Press, 1997.

[22] P. E. O'Neil and G. Graefe. Multi-table joins through bitmapped join indices. *SIGMOD Record*, 24(3):8–11, 1995.

[23] Patrick O'Neil. Informix indexing support for data warehouses. *Database Programming and Design*, 10(2):38–43, February 1997.

[24] Patrick O'Neil and Elizabeth O'Neil. *Database: pronciples, programming, and performance*. Morgan Kaugmann, 2nd edition, 2000.

[25] Torben Bach Pedersen and Christian S. Jensen. Research issues in clinical data warehousing. In Maurizio Rafanelli and Matthias Jarke, editors, *10th International Conference on Scientific and Statistical Database Management, Proceedings, Capri, Italy, July 1-3, 1998*, pages 43–52. IEEE Computer Society, 1998.

[26] Doron Rotem, Kurt Stockinger, and Kesheng Wu. Optimizing candidate check costs for bitmap indices. In *CIKM 2005*. ACM Press, 2005.

[27] R. R. Sinha, S. Mitra, and M. Winslett. Bitmap indexes for large scientific data sets: A case study. In *Proc. IEEE International Parallel & Distributed Processing Symposium (20th IPDPS'06)*, Rhodes Island, Greece, April 2006. IEEE Computer Society.

[28] Kurt Stockinger and Kesheng Wu. *Bitmap Indices for Data Warehouses*, chapter VII, pages 179–202. Idea Group, Inc., 2006. LBNL-59952.

[29] Kurt Stockinger, Kesheng Wu, and Arie Shoshani. Strategies for processing ad hoc queries on large data warehouses. In *Proceedings of DOLAP'02*, pages 72–79, McLean, Virginia, USA, 2002. A draft appeared as tech report LBNL-51791.

[30] Kurt Stockinger, Kesheng Wu, and Arie Shoshani. Evaluation strategies for bitmap indices with binning. In *International Conference on Database and Expert Systems Applications (DEXA 2004), Zaragoza, Spain*. Springer-Verlag, September 2004.

[31] H. K. T. Wong, H.-F. Liu, F. Olken, D. Rotem, and L. Wong. Bit transposed files. In *Proceedings of VLDB 85, Stockholm*, pages 448–457, 1985.

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

[33] Kesheng Wu. Fastbit: an efficient indexing technology for accelerating data-intensive science. *J. Phys.: Conf. Ser.*, 16:556–560, 2005. doi:10.1088/1742-6596/16/1/077.

[34] Kesheng Wu, Ekow Otoo, and Arie Shoshani. An efficient compression scheme for bitmap indices. *ACM Transactions on Database Systems*, 31:1–38, 2006.

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

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

[37] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. On the performance of bitmap indices for high cardinality attributes. In Nascimento et al. [19], pages 24–35.

[38] M.-C. Wu. Query optimization for selections using bitmaps. In Delis et al. [10].

[39] M.-C. Wu and A. P. Buchmann. Encoded bitmap indexing for data warehouses. In *Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, pages 220–230. IEEE Computer Society, 1998.

[40] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, 1998.

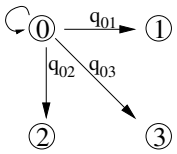


Figure 15: A 4-state Markov model. To avoid clutter, only transition probabilities from state 0 to others are marked. The transition probability of staying in state 0 is $q_{00} \equiv 1 - q_{01} - q_{02} - q_{03}$. The transition probabilities from other states can be similarly defined.

A Index Sizes on More Realistic Data

In the body of the article, we show the performance characteristics of Zipfian data and Markovian data in graphs but without given the exact formulas for them. In this appendix, we give a brief description of these two types of data and give the exact formulas for the expected index sizes under the three basic encodings. We will not repeat the formulas for the query processing costs to answer queries because they can be computed using the general formulas given before by plugging in the specific formula for bitmap sizes of different data.

Similar to Section 5, we start by discussing how the synthetic data is generated, following by discussing the formula for compute the size of each individual bitmap, then discussing the sizes of various indexes. We give detailed formulas for the three one-component encodings because they are the building blocks for other more complex encoding methods. We also discuss the formulas for the multi-component equality encoded indexes.

A.1 Model data

The two types of more realistic data are Zipfian data and Markovian data. The Zipfian data contains records that are independent from each other as the uniform random data, but the probability distribution of each value follows the well-known Zipf distribution, where the value i has a probability that is proportional to i^{-z}, with z known as the Zipf exponent [**?**].

The rows of uniform random data and Zipfian data are not correlated with each other. However, in real applications, data records are often related. We use the Markov model to capture this dependency [3]. Under a Markov model, a new record is generated based on the most recent record as illustrated in Figure 15. This particular Markov model has four states, which will generate an attribute with cardinality 4. In general, an attribute with cardinality C is generated with a C-state Markov model.

A.2 Size of Markovian bitmap

In a bitmap index on Zipfian data, all bitmaps are random bitmaps with sizes given in Equation 6. We don't need to produce a separate formula for compute the sizes of their bitmaps. However, we do need another formula for computing the sizes of bitmaps on Markovian data.

For Markovian data, each bitmap in the bitmap index is a Markov bitmap as described in [34]. In addition to parameters N and d used to describe a random bitmap, a Markov bitmap requires one more parameter called the *clustering factor* f.

The Markov bitmaps may contain longer 1-fills than random bitmaps, this makes the bitmaps more compact. Given a large enough dataset, the bit density of a bitmap corresponding to one value approaches the probability of the state corresponding to the value in the stable distribution of the Markov model [3]. The clustering factor f of the bitmap is the clustering factor of the Markov model. For the example given in Figure 15, the clustering factor of the bitmap corresponding to value (state) 0 is determined by q_{00}. According to the analysis in [34], we have $f = 1/(1 - q_{00})$.

The number of words required to store a Markov bitmap with bit density d and clustering factor f is as follows,

$$\begin{aligned} m_M(d,f) &= \left\lfloor \frac{N}{w-1} \right\rfloor + 2 - \\ &\quad \left(\left\lfloor \frac{N}{w-1} \right\rfloor - 1 \right) \left((1-d)(1-q_{01})^{2w-3} + d(1-q_{10})^{2w-3} \right) \\ &\approx \frac{N}{w-1} \left(1 - (1-d) \left(1 - \frac{d}{(1-d)f} \right)^{2w-3} - d \left(\frac{f-1}{f} \right)^{2w-3} \right). \end{aligned} \tag{40}$$

The above formula was derived using the same two-word grouping used for the random bitmaps, the details of which can be found in [34].

To simplify the discussion for Markovian data, we limit the Markov model to have a simple transition matrix; all its off-diagonal entries of the transition matrix must be the same. In the example shown in Figure 15, we restrict $q_{01} = q_{02} = q_{03} = q_{10} = q_{12} = q_{13} = \ldots = q$. The stable distribution of this Markov model is a *uniform distribution*. The clustering factor of a bitmap corresponding to a single value is

$$f = 1/(1-q_{ii}) = 1/(1-(1-(C-1)q)) = 1/(C-1)q. \tag{41}$$

Since bitmap indexes on uniformly distributed data are always harder to compress than non-uniform data of the same cardinality, the data produced by this particular Markov model represents the worst cases among all datasets generated by Markov models.

As a consistency check, we observe that the uniform random numbers can be generated with the Markov model using a special choice of $q = 1/C$. This choice of transition matrix means that the another of the C states can be reached from any state with equal probability, therefore this special Markov model is equivalent to the uniform random process used earlier. In this case, the clustering factor f equals $1/((C-1)/C) = C/(C-1)$. For any bitmap from a bitmap index for both the uniform random data and the Markov model, the bit density $d = 1/C$. Substituting the bit density into Equation 6, to get the size of a bitmap as

$$m_r(1/C) \approx N(1-(1-1/C)^{2w-2} - 1/C^{2w-2})/(w-1). \tag{42}$$

At the same time, we can substitute the bit density and the clustering factor into Equation 40, we compute the size of a bitmap to be

$$\begin{aligned} m_M(\frac{1}{C}, \frac{C}{C-1}) &\approx \frac{N}{w-1} \left(1 - (1-\frac{1}{C}) \left(1 - \frac{1/C}{(1-1/C)(C/(C-1))} \right)^{2w-3} \right. \\ &\quad \left. -\frac{1}{C} \left(\frac{C/(C-1)-1}{C/(C-1)} \right)^{2w-3} \right) \\ &= N(1-(1-1/C)^{2w-2} - 1/C^{2w-2})/(w-1). \end{aligned} \tag{43}$$

We see that Equations 42 and 43 give the same result. In fact, it is straightforward to verify that the exact version of Equation 6 and 40 are also the same if d is substituted with $1/C$ and f is substituted with $C/(C-1)$. This shows that these formulas for evaluating sizes of bitmaps are consistent with each other.

A.3 Equality encoding

In Section 5.3, we give the formula for the total size of one-component equality encoded bitmap index on uniform random data. Now, we consider the expected total size of bitmaps of the same index on Zipfian and Markovian data.

A.3.1 Zipfian data

Let α_z denote the scaling constant of the probability distribution, we have $\alpha_z = (\sum_1^C i^{-z})^{-1}$. The probability for the ith value, d_i, is $\alpha_z i^{-z}$. Since each record is generated independently of others, we have random bitmaps in the index for an attribute following the Zipf distribution. The total size of such an index under WAH compression is given by the following summation,

$$s_z^E = \sum_{i=1}^{C} m_r(\alpha_z i^{-z}). \tag{44}$$

For $z = 0$, we have $\alpha_0 = 1/C$ and $d_i = 1/C$. In which case, $s_{z=0}^E \equiv s_U^E$.

If the attribute cardinality C is large and the Zipf exponent is small, say $z \leq 1$, then α_z is relatively small and the probability for each value is small. In this case, the bit density d_i of a bitmap in the bitmap index is also small, which means the following approximations are valid, $d_i^{2w-2} \approx 0$ and $(1-d_i)^{2w-2} \approx 1-(2w-2)d_i$. In this case, we can simplify the formula for the size of a bitmap to be $m_r(\alpha_z i^{-z}) \approx N((2w-2)\alpha_z i^{-z})/(w-1) = 2N\alpha_z i^{-z}$. By the definition of α_z, we obtain

$$s_z^E \approx 2N,$$

which gives the same approximate index size as the uniform random data. This is what we expected as the Zipf exponent approaches 0. However, in general, as z increases, the total index size would be smaller.

For $z > 1$, the summation $\sum_{i=1}^{C} i^{-z}$ and α_z approach their asymptotic maximum as C increases. For a large enough C, the first few values appear with nearly fixed probabilities no matter what is the actual value of C. For example, if $z = 2$, for any C larger than 20, the value of α_z is about 1.6 no matter what is the exact value of C. The probabilities for values greater than 20 are so small that their corresponding bitmaps in an index would be much smaller than those corresponding to the first 20 values. In this particular case, one reasonable way to estimate the index size would be to assume the first 20 bitmaps are not compressible, therefore the index size would be close to $20N/(w-1)$ words, which is much smaller than $2N$ words.

A.3.2 Markovian data

For an attribute produced by the Markov model, the bitmap index contains C Markov bitmaps with bit density $d = 1/C$ and clustering factor f (same as the clustering factor of the Markov model). The total size of these bitmaps is given by the following equation,

$$\begin{aligned} s_M^E &= C m_M(\frac{1}{C}, f) \\ &\approx \frac{NC}{w-1}\left(1-\left(1-\frac{1}{C}\right)\left(1-\frac{1}{(C-1)f}\right)^{2w-3}\right. \\ &\qquad \left. -\frac{1}{C}\left(1-\frac{1}{f}\right)^{2w-3}\right). \end{aligned} \tag{45}$$

By definition, f cannot be less than 1. For large C, say $C \geq 100$, we can assume $(1-1/((C-1)f))^{2w-3} \approx 1-(2w-3)/((C-1)f)$. In this case, $(1-1/C)(1-1/((C-1)f))^{2w-3} \approx (1-1/C)(1-(2w-3)/((C-1)f)) = 1-1/C-(2w-3)/(Cf)$. This leads to the following approximation for the index size,

$$\begin{aligned} s_M^E &\approx \frac{NC}{w-1}\left(\frac{1}{C}+\frac{2w-3}{Cf}-\frac{1}{C}\left(1-\frac{1}{f}\right)^{2w-3}\right) \\ &= \frac{N}{w-1}\left(1+\frac{2w-3}{f}-\left(1-\frac{1}{f}\right)^{2w-3}\right). \end{aligned} \tag{46}$$

We see that the index size is independent of the actual value of C for large enough C. This is similar to the case of uniform random numbers. In most applications, the values of f are smaller than w. In this case, the total index size is nearly inversely proportional to the clustering factor f.

A.4 Range encoding

The bitmaps produced using the one-component range encoding are effectively the precomputed answers to one-sided range queries. A key advantage of this encoding is that it can answer range queries quickly, but the bitmaps are not amenable to compression. In this subsection, we quantify the index size and query processing cost to see if it offers a good space-time trade-off.

A.4.1 Zipfian data

Under Zipf distribution, the probability corresponding to value j is $\alpha_z j^{-z}$. Under the range encoding, the bit density of the ith bitmap is $d_i = \sum_{j=1}^{i} \alpha_z j^{-z}$. Since each such bitmap is a random bitmap, we can use Equation 6 to compute its size. Summing up the size of each such bitmap, we obtain the formula for the total size of bitmaps as

$$s_z^R \approx \sum_{i=1}^{C-1} \frac{N}{w-1} \left(1 - \left(1 - \sum_{j=1}^{i} \alpha_z j^{-z} \right)^{2w-2} - \left(\sum_{j=1}^{i} \alpha_z j^{-z} \right)^{2w-2} \right). \quad (47)$$

We know that WAH compression does not reduce the sizes of bitmaps with bit densities between 0.05 and 0.95. Figure 3 shows the densities of bitmaps for Zipfian data with attribute cardinality 100. For $z = 0$, about 90% of the bitmaps are not compressible. For $z = 1$, about 80% of the bitmaps are not compressible. For $z = 2$, about 10% of the bitmaps are not compressible. Generally, when $z > 0$, the fraction of compressible bitmaps increases as attribute cardinality increases. As more bitmaps become compressible, the total index size will decrease.

A.4.2 Markovian data

To evaluate the sizes of indexes in Markovian data, we need to also compute the clustering factor of the bitmaps. Let f_i denote the clustering factor of the ith bitmap. The clustering factor the first bitmap f_1 is the same as the clustering factor f of the Markov model that generated the data. Based on Equation 41 we say that the clustering factor is the inverse of the probability of the Markov model going from a state represented by a bitmap to a state not represented by the bitmap. The ith bitmap in range encoding represents the first i values; the probability of going from one of these values to $(C-i)$ others is $(C-i)q$ according to the definition of the Markov model. Therefore, the clustering factor of the ith bitmap is

$$f_i = \frac{1}{(C-i)q} = \frac{C-1}{C-i} f. \quad (48)$$

Plug in the formulas for the bit density $d_i = i/C$ and the clustering factor f_i into Equation 40, we obtain the following expression for the index size for Markovian data

$$s_M^R \approx \sum_{i=1}^{C-1} \frac{N}{w-1} \left(1 - \left(1 - \frac{i}{C} \right) \left(1 - \frac{i(C-i)}{(C-1)^2 f} \right)^{2w-3} - \frac{i}{C} \left(1 - \frac{(C-i)}{f(C-1)} \right)^{2w-3} \right). \quad (49)$$

A.5 Interval encoding

Interval encoding produces about half as many bitmaps as equality encoding and range encoding. It can answer the queries just as efficient as range encoding. Next, we quantify these performance characteristics for the basic one-component interval encoded index to see whether it presents a better space-time trade-off than the range encoded index.

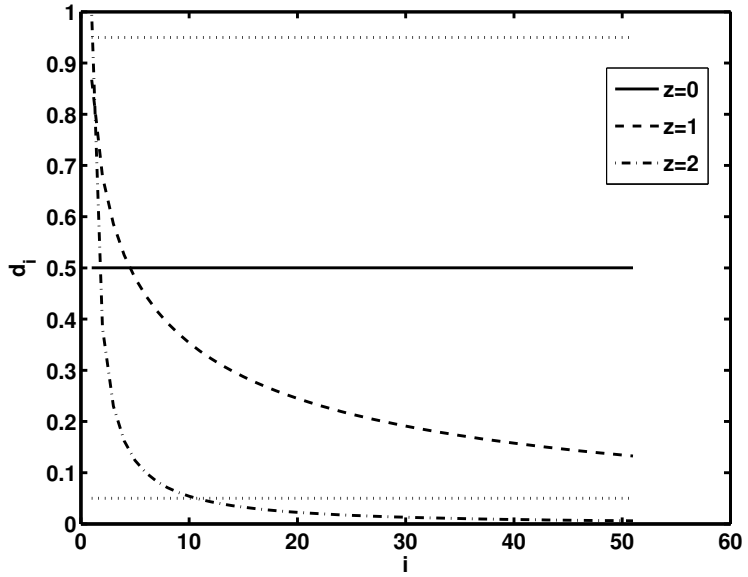


Figure 16: The bit density d_i of bitmaps from an interval encoded index on Zipfian data ($C = 100$). WAH is only able to compress random bitmaps with bit density less than 0.05 or more than 0.95.

A.5.1 Zipfian data

The bit density of the ith bitmap in an interval encoded bitmap index on a Zipf attribute is $d_i = \sum_{j=i}^{i+\lceil C/2 \rceil - 1} \alpha_z j^{-z}$. Plugging in this bit density into Equation 6, we obtain the total size of bitmaps as

$$s_z^I = \sum_{i=1}^{C-1} m_r d_i \approx \frac{N}{w-1} \sum_{i=1}^{C-1} \left(1 - (1-d_i)^{2w-2} - d_i^{2w-2}\right).$$

To simplify the above formula, we again take advantage of the fact that WAH is not able to reduce the size of bitmaps with densities between 0.05 and 0.95. To see the exact values of d_i, we plot them from indexes on different Zipf distributed data in Figure 16. In this case, all attributes have attribute cardinality of 100. There are 51 bitmaps in each index. For $z = 0$, the bit density of every bitmap is exactly 0.5 as expected. For $z > 0$, the lines formed by the bit densities are monotonically decreasing. For $z = 1$, we see that none of the bitmaps are compressible because their bit densities are all between 0.05 and 0.95. This indicates that the index size for z between 0 and 1 is about the same as in the uniform random case, that is, $s_z^I \approx NC/2(w-1)$. Compared with equality encoding, interval encoded index can be much larger in size. However, the interval encoded index is about half the size of a range encoded index.

As z increases, more bitmaps may have densities less than 0.05. In fact, for $z = 2$, 40 out of the 51 bitmaps have bit density less than 0.05, which means about 80% of the bitmaps are compressible. We also note that as attribute cardinality increases, a larger fraction of bitmaps becomes compressible as well. As more bitmaps become compressible, their total size decreases.

A.5.2 Markovian data

To compute the expected index sizes on Markovian data, we can reuse the same formula for bit density for the uniform random data, but, we need to compute the clustering factors of the bitmaps. To do this, we use the fact that the clustering factor of a bitmap is the inverse of the probability of the Markov model going from a value represented by the bitmap to a value not represented by the bitmap. Since each bitmap under interval encoding represents $\lceil C/2 \rceil$ values, the transition probability that the next value is one of the other $C - \lceil C/2 \rceil$ values is $(C - \lceil C/2 \rceil)q$. According to Equation 41 that defines the relationship between q and

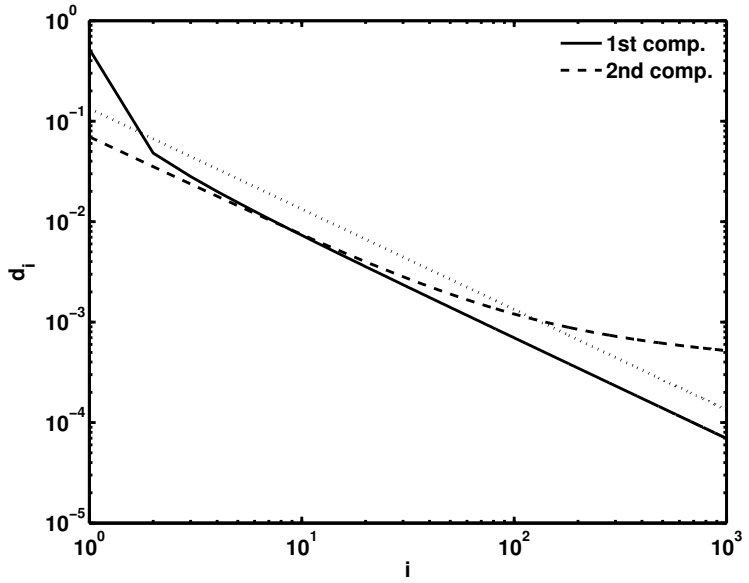


Figure 17: The bit density d_i of bitmaps from a two-component equality encoding of a Zipf attribute ($C_1 = 1000$, $C_2 = 1000$, $C = 10^6$, $z = 1$).

the clustering factor f of the Markov model, we can express the clustering factor of a bitmap as

$$f_i = f(C-1)/(C-\lceil C/2\rceil) \approx 2f. \tag{50}$$

Therefore, the total size of bitmaps in an interval encoded index can be written as follows,

$$\begin{aligned} s_M^I &= (C-\lceil C/2\rceil+1)m_M(\lceil C/2\rceil/C, f(C-1)/(C-\lceil C/2\rceil)) \\ &\approx \frac{C}{2}m_M(\frac{1}{2}, 2f) \\ &\approx \frac{NC}{2(w-1)}\left(1-\left(1-\frac{1}{2f}\right)^{2w-3}\right). \end{aligned} \tag{51}$$

In most cases, where the clustering factor f is modest, say f is between 1 and 10, the value of $(1-(2f)^{-1})^{2w-3}$ is much less than 1, for example, $(1-20^{-1})^{62} = 0.04$. Therefore, the total size of bitmaps is approximately $NC/2(w-1)$, which is the same as in the case of uniform random data. However, in rare cases where f is much larger than w, we have $(1-1/2f)^{2w-3} \approx 1-(2w-3)/2f$. Further assuming $(2w-3)/(2w-2) \approx 1$, we can simplify the expression for s_M^I to be $s_M^I \approx NC/2f$. In these cases, the index size is inversely proportional to the clustering factor f.

A.6 Multi-component equality encoding

A.6.1 Zipfian data

For Zipfian data, the probability for the ith value is $\alpha_z i^{-z}$, and the probability that the jth component has value i_j is (note that $i \equiv \sum_{g=1}^{k} i_g \prod_{l=g+1}^{k} C_l$)

$$d_{i_j}^{Ek} = \alpha_z \sum_{i_1=1}^{C_1} \cdots \sum_{i_{j-1}=1}^{C_{j-1}} \sum_{i_{j+1}=1}^{C_{j+1}} \cdots \sum_{i_k=1}^{C_k} \left(\sum_{g=1}^{k} i_g \prod_{l=g+1}^{k} C_l\right)^{-z}. \tag{52}$$

For the first component, the bit density of a bitmap $d_{i_1}^{Ek}$ is the sum of probabilities of C/C_1 consecutive values, i.e., $d_{i_1}^{Ek} = \sum_{l=1+(i_1-1)C/C_1}^{i_1 C/C_1} \alpha_z l^{-z}$. Because i^{-z} is a monotonically decreasing function (when $z > 0$),

$d_{i_1}^{Ek}$ is also a monotonically decreasing function. Furthermore the decrease is sharper in $d_{i_1}^{Ek}$ than in i_1^{-z}, as illustrated in Figure 17. In this figure, we plotted the bit densities of all bitmaps from a two-component equality encoded index on a Zipf attribute with attribute cardinality 1 million and Zipf exponent 1. We choose to have two components of the same size, $C_1 = C_2 = 1000$. The dotted line in the figure shows a reference distribution for $C = 1000$. In this case, we see the bit density of the first bitmap d_1 is about a half, while d_i for most other bitmaps are less than 0.05. This means the bitmaps in the first component are easy to compress.

For the last component in a multi-component encoding, the expression above expression for the bit density can also be significantly simplified, $d_{i_k}^{Ek} = \sum_{l=0}^{C/C_k-1} \alpha_z (i+k+l*C/C_k)^{-z}$. Because i^{-z} is monotonically decreasing, $d_{i_k}^{Ek}$ is also monotonically decreasing as i_k increases. In fact, the bit densities of bitmaps in each component form a decreasing line, but the rate of decrease reduces as the component number j increases. Therefore, this decreases the likelihood that any bitmap has density less than 0.05 or greater than 0.95, which means more bitmaps are not compressible. In Figure 17, we see that the majority of the bitmaps in the second component have a higher bit density than the simple Zipf distribution (shown as the dotted line).

Given the above bit densities, we can evaluate the total size of the bitmaps by summing up the expected sizes of bitmaps given by Equation 6, $s_z^{Ek} = \sum_j \sum_{i_j} m_r(d_{i_j}^{Ek})$. This summation can be evaluated numerically using a computer program.

Next, we examine a few special cases where the index size can be simplified to compact expressions. The first special case is for $z \leq 1$. In previous analyses, we see that the index sizes of such Zipfian data are just about the same as the uniform random data. If the same is true for every component, then we again have $s_z^{Ek} \approx 2kN$. Since the first component is expected to be compressed better than a Zipf dataset with Zipf exponent z and cardinality C_1, the total index size is likely to be less than $2kN$, but we do not have a compact expression for the exact difference.

For Zipfian data with large Zipf exponents, say, $z \geq 2$, we know that the probability of the first few values are nearly constant no matter how many other values are observed. In this case, these first few values will determine the total index size. Let g_z denote the number of such values. When they are encoded using the multi-component equality encoding, each of them will be encoded into a different bitmap in the last component if $C_k \geq g_z$. In this case, the bitmaps in the kth component have nearly the same bit densities as the first C_k bitmaps in the one-component equality encoded index. Therefore the total size of bitmaps in kth component is about the same as the total size of all bitmaps in the one-component equality encoded index. The total size of bitmaps in other components may be small because the first bitmap in each of these components have a bit density close to 1 and all bitmaps are very easy to compress. Altogether, the multi-component index takes more space than the one-component version, though the difference could be small.

A.6.2 Markovian data

For the Markovian data, the bit densities are the same as for the uniform random data. To compute the index sizes, we need to evaluate the clustering factors. In a bitmap from the jth component, a bit being 1 represents the Markov model is in one of some C/C_j states. The probability of going from one of these states to others is $Cq(1-1/C_j)$. Using Equation 41, we can evaluate the clustering factor of the bitmap as

$$f_j^{Ek} = (Cq(1-1/C_j))^{-1} = \frac{(C-1)C_j}{C(C_j-1)} f. \tag{53}$$

For relatively large C and C_j, we see that the clustering factors of bitmaps produced by the multi-component equality encoding is nearly the same as the clustering factor of the Markov model.

Given the above clustering factors and the bit densities, using the formula for the size of a Markov bitmap given in Equation 40, we can express the total size of the bitmaps as follows

$$s_M^{Ek} = \sum_{j=1}^{k} C_j m_m(\frac{1}{C_j}, f_j^{Ek})$$

$$\approx \frac{N}{w-1}\sum_{j=1}^{k} C_j \left(1-\left(1-\frac{1}{C_j}\right)\left(1-\frac{C}{(C-1)C_j f}\right)^{2w-3} - \frac{1}{C_j}\left(1-\frac{C(C_j-1)}{(C-1)C_j f}\right)^{2w-3}\right). \quad (54)$$

As in the case of the one-component equality encoding, the clustering factor of the Markov model has a strong influence on the index size. To illustrate this influence, we assume that all C_j are larger than 100. In this case, we can use the same approximation that was used to produced Equation 46, and give the following expression for the total index size (note we also assume $C \approx C-1$).

$$s_M^{Ek} \approx \frac{N}{w-1}\sum_{j=1}^{k}\left(1+\frac{2w-3}{f}-\left(1-\frac{C_j-1}{C_j f}\right)^{2w-3}\right).$$

If we further assume either $(1-\frac{C_j-1}{C_j f})^{2w-3} \approx (1-1/f)^{2w-3}$ or $(1-\frac{C_j-1}{C_j f})^{2w-3} \approx 0$, then $s_M^{Ek} \approx k s_M^E$. Similar to the case of uniform random data, the index size increases as the number of components increases for small k. As k increases, the values of C_j will become smaller, in which case, we can approximate the index size by considering each bitmap to be incompressible. Since the number of bitmaps decreases as k increases, the total index will eventually decrease as k further increases. Again, the minimal index size would be achieved with either the minimal k ($k=1$) or the maximum k.

A.7 Multi-component range encoding and multi-component interval encoding

A.7.1 Zipfian data

As indicated before, the bitmaps for the multi-component range encoding and the multi-component interval encoding can be constructed from those for a multi-component equality encoding. In the jth component, the i_jth bitmap in the multi-component range encoding is a bitwise OR of the first i_j bitmaps in jth component of the multi-component equality encoding. Therefore, the bit density is simply the sum of those of the first i_j bitmaps and can be computed from the expressions for the bit densities in the multi-component equality encoding as shown in Equation 52, i.e.,

$$d_{i_j}^{Rk} = \sum_{g_j=1} i_j d_{g_j}^{Ek}.$$

Similarly, the bit density of the i_jth bitmap in the jth component of a multi-component interval encoding can be expressed as

$$d_{i_j}^{Ik} = \sum_{g_j=i_j}^{i_j+\lceil C_j/2\rceil-1} d_{g_j}^{E_k}.$$

Since we do not have concise expression for Equation 52, we not able to produce concise formulas for $d_{i_j}^{Rk}$ or $d_{i_j}^{Ik}$ either. Instead, we use a computer program to evaluate these expressions to compute the bit densities, use Equation 6 to compute sizes of each bitmap, and then sum up the total size of the bitmaps to give the size information plotted in Figures 6 and 7. From this process, we observe that most of the bitmaps are incompressible using WAH compression.

A.7.2 Markovian data

In Sections A.4.2 and A.5.2, we explained how the clustering factors of bitmaps under range encoding and interval encoding are related to the clustering factors of the bitmaps under equality encoding. The same can be applied here. Under multi-component range encoding, the bitmaps in the jth component have clustering factor given in Equation 53. Substituting the clustering factor f on the right-hand size of Equation 48, we

obtain the expression for the clustering factor of the i_jth bitmap in the jth component under multi-component range encoding,

$$f_{i_j}^{Rk} = \frac{C_j - 1}{C_j - i_j} f_j^{Ek} = \frac{(C-1)C_j}{C(C_j - i_j)} f.$$

Similarly, the clustering factor of the bitmaps in the jth component of a multi-component interval encoding can be derived from Equations 50 and 53 as follows,

$$f_j^{Ik} = \frac{(C_j - 1)}{(C_j - \lceil C_j/2 \rceil)} \frac{(C-1)C_j}{C(C_j - 1)} f = \frac{(C-1)C_j}{C(C_j - \lceil C_j/2 \rceil)} f \approx 2f.$$

Note that the bit densities for Markov data are the same as those for the uniform random data. With these quantities, we can compute the sizes of each bitmap using Equation 40, and then compute the sum as the total size of bitmaps in an index. This procedure generated the values that go into Figure 6 and 7. Note that with clustering factor $f = 2$, the total index size is only slightly smaller than the uniform random case (marked with $z = 0$).

A.8 Sizes of coarse-level bitmaps in multi-level encodings

In Section 7.1, we mentioned that for both the uniform random data and Zipfian data, the coarse level bitmaps can be treated as random bitmaps with bit density $1/B_c$. For Markovian data, the bit densities of the coarse level bitmaps are the same, i.e., equal to $1/B_c$. Next, we briefly explain how to compute their clustering factors.

As before, we can compute the clustering factor for equality encoded bitmaps as the inverse of the probability for the Markov model to transfer from one of the values in the coarse bin to a value outside. It is clear that each coarse bin contains C/B_c values which represent C/B_c states of the Markov model. Let q denote the probability for the Markov model to transfer from one state to another, then the probability that the Markov model transfer from one of the C/B_c states to another one outside the coarse bin is $qC(1 - 1/B_c)$. Thus the clustering factor of an equality-encoded bitmap is $(qC(1-1/B_c))^{-1} = f(1-1/C)/(1 - 1/B_c)$, where f is the clustering factor of the Markov model, see Equation 41 for further details. Given this clustering factor for equality encoded bitmaps, the corresponding clustering factor for range and interval encoded bitmaps can be determined as in Sections A.4.2 and A.5.2. Substitute the above clustering factor into Equation 48, we obtain the clustering factor of the ith coarse bitmap with range encoding as

$$f_i = \frac{B_c - 1}{B_c - i} \frac{1 - 1/C}{1 - 1/B_c} f = \frac{B_c(C-1)}{(B_c - i)C} f.$$

Similarly, using Equation 50, we obtain the clustering factor of coarse bitmaps with interval encoding as

$$\frac{B_c - 1}{B_c - \lceil B_c/2 \rceil} \frac{1 - 1/C}{1 - 1/B_c} f = \frac{B_c(C-1)}{(B_c - \lceil B_c/2 \rceil)C} f \approx 2f.$$

With the values of bit density and clustering factor, we can compute size of each individual bitmap using Equation 40 and then derive the expression for the index sizes and query processing costs.