A general analytical model for spatial and temporal performance of bitmap index compression algorithms in Big Data

Yinjun Wu, Zhen Chen, Yuhaow Wen, Junwei Cao, Wexsun Zheng, Ge Ma
Research Institute of Information Technology, Tsinghua University
Tsinghua National Lab for Information Science and Technologies (TNList),
Beijing, China
Wu-yy12@mails.tsinghua.edu.cn

Abstract—Bitmap indexing is flexible to conduct boolean operations in data retrieval. Besides, the query processing based on bitmap indexing is also very fast. Therefore it has been widely used in various big data analytics platforms, such as Druid and Spark etc. However, bitmap index can consume a large amount of memory, which leads to the invention of different kinds of bitmap index compression algorithms without sacrificing temporal performance. In practice, we are often discommoded by choosing a proper algorithm when handling specific problems. Besides, after devising a new algorithm that may outperform existing ones, it is essential to evaluate its performance in theory. Without appropriate theoretical analysis, the deficit of a new algorithm can only be spotted until final experimental results are drawn, thus wasting much time and effort. In this paper, we propose a general analytical model to analyze both the spatial and temporal performance for bitmap index compression algorithms, which can be applied to analyze all kinds of algorithms derived from WAH (word-aligned hybrid). In this model, two types of distributed bitmaps, uniformly distributed bitmaps and clustered bitmaps, are used separately. In order to illustrate this model, several bitmap index compression algorithms are analyzed and compared with each other. Algorithms herein are COMBAT (COMBining Binary And Terminary encoding), SECOMPAX (Scope Extended COMPAX) and CONCISE (Compressed ‘n’ Composable Integer Set), which are all derived from WAH. Evaluation results by MATLAB simulation about these algorithms are also presented. This paper paves the way for further researches on the performance evaluation of various bitmap index compression algorithms in the future.

Keywords—bitmap index; Big Data; COMBAT; SECOMPAX; CONCISE; data compression, performance evaluation.

I. INTRODUCTION

Nowadays streaming data, such as sensing data from IoT devices, network traffic and machines’ operational logs etc., are soaring and many applications are experiencing hardness in querying and searching such big data. In order to solve this problem, bitmap indexing [1-7] has been widely used in Big data platforms. An example of bitmap index is shown in Fig. 1. However, since bitmap indexing consumes a large amount of memory and disk space, a series of bitmap index compression algorithms have been proposed, such as BBC[8], WAH [9-10], UCB [11], RLH [12], VLC [13], PLWAH [14], EWHAH [15], PWAH [16], COMPAX [17], GPU-WAH [18-19], GPU-PLWAH [20], SECOMPAX [21], PLWAH+[22], DFWAH [23], roaming bitmap[24], BREAD[25], VAL-WAH [26], CONCISE[27], COMBAT[28] etc. A detailed survey is presented in [29].

In practice, a proper bitmap index compression algorithm often matches a specific problem and the process of choosing often discommodes us. Besides, a theoretical evaluation for a newly devised algorithm is indispensable. For this purpose, an theoretical model is developed for analyzing both the spatial and temporal performance of bitmap index compression algorithms by using two kinds of bitmaps, i.e. uniformly distributed bitmaps and clustered bitmaps. Based on appropriate assumptions, some indispensable procedures compose this analytical model, including calculating Basic Probabilities (defined in Section III), listing all compressible word combinations, calculating probabilities of each codewords and working out final expected values of compressed size and decompression time in each algorithm. This model is proved by analyzing spatial and temporal performance of SECOMPAX, COMBAT, and CONCISE.

This paper is organized as follows. In Section II, specific encoding schemes of SECOMPAX, COMBAT and CONCISE are given. In section III, specific analysis procedures of these algorithms referred above are presented, which are composed of both the spatial and temporal analyses in two kinds of bitmap indexes. In Section IV, evaluation results on memory consumption and decompression time are given to show their comparison explicitly.

II. ALGORITHM

A. Basic Definitions

For convenience in analysis, basic terms in bitmap indexing are introduced and are listed in TABLE I. Based on these definitions, all kinds of codewords in SECOMPAX, COMBAT as well as CONCISE are introduced. Since these coding schemes are all operated after WAH encoding, the codewords in WAH are introduced firstly. Two types of codewords are used in WAH, i.e. fill and literal. A Fill represents the number of consecutive fills of the same kind.

<table>
<thead>
<tr>
<th>RowID</th>
<th>type</th>
<th>bitmap index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2 0 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2 1 1 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4 2 0 0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3 4 0 0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1 1 1 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 1 1 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5 0 0 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1 An example of bitmap index

TABLE I. Basic Definition
A MSB recording the type of these consecutive fills in compressed fill after WAH encoding. A Literal is the same definition shown in TABLE I.

### Terminology

**dirty byte** aims at compressing a set bit in the bitmap. Besides, two continuous chunks are fill, L and fill separately, then they can be merged into a chunk belonging to [FLF] codeword. Likewise, the [LFL] and [FLF] codeword can also be defined. Specific compositions of these two codewords are shown in Fig. 2 and Fig. 3 separately.

**TABLE I. Terminology and corresponding explanations**

<table>
<thead>
<tr>
<th>Terminology</th>
<th>Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>unset bit</td>
<td>A bit that is “0”</td>
</tr>
<tr>
<td>set bit</td>
<td>A bit that is “1”</td>
</tr>
<tr>
<td>chunk</td>
<td>Consecutive 31 bits in a bit sequence</td>
</tr>
<tr>
<td>Fill</td>
<td>Defined as a chunk composed of the same kinds of bit.</td>
</tr>
<tr>
<td></td>
<td>A 0-fill is defined as a fill only composed of unset bits.</td>
</tr>
<tr>
<td></td>
<td>A 1-fill is defined as a fill only composed of set bits.</td>
</tr>
<tr>
<td>literal</td>
<td>A chunk that is not fill.</td>
</tr>
<tr>
<td>dirty byte</td>
<td>Defined as a byte in a chunk containing set bits or unset bits exclusively.</td>
</tr>
<tr>
<td></td>
<td>A 0-dirty is defined as a byte in a chunk containing set bits exclusively.</td>
</tr>
<tr>
<td></td>
<td>A 1-dirty is defined as a byte in a chunk containing unset bits exclusively.</td>
</tr>
<tr>
<td>L</td>
<td>Defined as a chunk with one dirty byte</td>
</tr>
<tr>
<td></td>
<td>A 0-L is defined as a chunk with a 0-dirty</td>
</tr>
<tr>
<td></td>
<td>A 1-L is defined as a chunk with a 1-dirty</td>
</tr>
<tr>
<td>NI2-L</td>
<td>Defined as a chunk with two dirty bytes</td>
</tr>
<tr>
<td></td>
<td>0-L is defined as a chunk with two 0-dirty</td>
</tr>
<tr>
<td></td>
<td>1-L is defined as a chunk with two 1-dirty</td>
</tr>
</tbody>
</table>

### Codewords in SECOMPAX

In SECOMPAX, new types of [LFL] and [FLF] codewords are devised for further compression after WAH encoding. If three consecutive chunks are fill, L and fill separately, then they can be merged into a chunk belonging to [FLF] codeword. Likewise, the [LFL] codeword can also be defined. Specific compositions of these two codewords are shown in Fig. 2 and Fig. 3 separately.

#### Table II. Codewords in SECOMPAX

<table>
<thead>
<tr>
<th>Codeword</th>
<th>Last byte of 1st fill</th>
<th>Dirty byte of L</th>
<th>Last byte of 2nd fill</th>
</tr>
</thead>
<tbody>
<tr>
<td>0th byte</td>
<td>1st byte</td>
<td>2nd byte</td>
<td>3rd byte</td>
</tr>
</tbody>
</table>

### Codewords in COMBAT

COMBAT is derived from SECOMPAX which includes all codewords of SECOMPAX and beyond. Furthermore, some special codewords are also introduced to improve the compression ratio. If there already exist two continuous chunks, L and fill (but without another L following), these two continuous chunks can be merged into a new compact one. This merge creates a new kind of codeword named [LF]. Besides, if an NI2-L just locates in front of a fill after WAH encoding, then these two chunks can be merged into a new one, called [NI2-L]. The composition of [LF] and [NI2-L] is also listed in Fig. 4 and Fig. 5 respectively.

#### Table III. Codewords in COMBAT

<table>
<thead>
<tr>
<th>Codeword</th>
<th>Dirty byte of L</th>
<th>Last byte of fill</th>
</tr>
</thead>
<tbody>
<tr>
<td>0th byte</td>
<td>1st byte</td>
<td>2nd byte</td>
</tr>
</tbody>
</table>

### Codewords in CONCISE

CONCISE is a bitmap index compression algorithm which is also extended from WAH. CONCISE introduces a new type of codeword based on fill (defined as NL-F). It aims at compressing a fill and a special literal which includes only one set bit in a literal. This compressible literal is defined as N-fill here. In a NL-F, besides the number of fill, the position of the only set bit in N-fill is also recorded. The [NL-F] codeword is shown in Fig. 6.

Intuitively, since COMBAT provides more possibilities to compress an uncompressed bit sequence comparing to SECOMPAX and CONCISE, it can be expected that the compressed size after COMBAT encoding is smaller than SECOMPAX and CONCISE. In terms of decompression time, due to shorter compressed bit sequence in COMBAT, it can take less time to load this compressed bitmap into memory and CPU. Thus shorter process time can also be expected. However, more process time can be expected when processing a single compressed chunk in COMBAT, because it uses more codewords. As a whole, these three algorithms can share nearly the same decompression time.

### III. Spatial and Temporal Analytical Model

#### A. Compression of uniformly distributed bitmaps

1) Assumptions

In this analysis model, we assume that only three chunks exist in raw bitmap indexes so that they can be compressed by COMBAT, SECOMPAX and CONCISE at the same time. It is convenient for the following analyses and this case can be generalized to other complex cases.

Before conducting analysis, some assumptions are listed below:

- The density of set bits is denoted by a variable $d$.
- The variable $d$ is independent from the distribution of set bits in uniformly distributed bitmaps.
- The value of $d$ can be considered very small, approaching to zero.
• For computational convenience, Taylor expansion can be applied since the value of \(d\) approaches zero. After Taylor expansion, only terms of the first and the second degree are retained.

• The execution time of an assignment statement and a conditional statement (if or else statement) in programs are consistent and denoted by variables \(t_1\) and \(t_2\) separately.

There are some explanations about the value of \(d\). According to listed assumptions, \(d\) is approaching zero, which is reasonable in real network environment. For example, since each byte composing an IP address ranges from 0 to 255, the average value of \(d\) is 1/256, less than 0.5% in each bitmap. In general network environment, the value of each byte is a random integer between 0 and 255 because a website can be accessed by millions even billions of different users with various IP address. So the value of \(d\) in each bitmap is very small, approaching to zero.

2) Size of compressed uniformly distributed bitmaps

a) Step 1: Calculating Basic Probabilities

Based on assumptions above, the probabilities of fill, literal, \(L\) and NI2-L (defined as Basic probabilities) can be calculated out separately. The probabilities of each codeword can be denoted by the value of Basic probabilities simplified through Taylor Expansion.

In order to generalize this model to other algorithms based on WAH, the Basic probabilities are separated into two parts, some are general ones for all algorithms derived from WAH while others are special ones for specific algorithms

• General Basic probabilities for all algorithms based on WAH:

Since in a 0-fill 31 bits are all unset bits, the probability (denoted by \(p_1\)) will be

\[
p_1 = (1 - d)^{31} = 1 - 31d + 465d^2
\]  

(1)

Likewise, the probability of 1-fill (denoted by \(p_2\)) can deduced out according to symmetrical characteristic (replacing \(d\) with \(1-d\) in equation (1)).

\[
p_2 = d^{31} = 0
\]  

(2)

After getting the value of \(p_1\) and \(p_2\), the probability of literal (denoted by \(p_3\)) can be written below:

\[
p_3 = 1 - p_1 - p_2 = 1 - (1 - d)^{31} - d^{31} = 31d - 465d^2
\]  

(3)

• Special Basic probabilities for SECOMPAX and COMBAT:

In terms of the probability of 0-L (denoted by \(p_4\)) it can be deduced below:

\[
p_4 = C_4d^3(1 - d^8)(1 - d^7)(1 - d^4)
\]  

\[
\approx 31d - 825d^2
\]  

(4)

The first term in equation (4) denotes the case that the dirty byte lies in the lefmost position while the second term denotes the case that the dirty byte locates in one of the other three bytes in this 0-L.

Similarly, according to symmetrical characteristic, the probabilities of 1-L (denoted by \(p_5\)) can be denoted below:

\[
p_5 = C_4d^3(1 - d^8) + (1 - d^7)(1 - d^4) \approx 0
\]  

(5)

The results of the probabilities of 0-NI2-L and 1-NI2-L (denoted by \(p_4\) and \(p_6\) separately) are shown below and their derivation process is similar to the former one.

\[
p_4 \approx 360d
\]  

(6)

\[
p_6 = 31d - 360d^2
\]  

(7)

• Special Basic probabilities for CONCISE:

When it comes to the probability of \(N\)-fill in CONCISE, since only a set bit exists, the probability (denoted by \(p_8\)) can be denoted as follows by binomial theorem.

\[
p_8 = C_8d^0 + 3d(1 - d^3) \approx 31d - 930d^2
\]  

(8)

b) Step 2: Calculating probabilities of all compressible codewords

TABLE II lists probabilities of all compressible three-continuous-word combinations and corresponding compressed sizes after COMBAT encoding. Based on these probabilities, the expected value of compressed size in COMBAT (denoted by \(e_{\text{COMBAT}}\)) can be calculated out and detailed derivation processes of these probabilities are shown in Appendix (see Proof 1).

<table>
<thead>
<tr>
<th>word combination</th>
<th>compressed size</th>
<th>Symbol</th>
<th>Probability value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-fill+0-fill+0-fill</td>
<td>1</td>
<td>(q_1)</td>
<td>1-93d+278d^2</td>
</tr>
<tr>
<td>1-fill+0-fill+0-fill</td>
<td>2</td>
<td>(q_2)</td>
<td>0</td>
</tr>
<tr>
<td>0-fill+0-fill+1-fill</td>
<td>2</td>
<td>(q_2)</td>
<td>0</td>
</tr>
<tr>
<td>1-fill+1-fill+0-fill</td>
<td>2</td>
<td>(q_0)</td>
<td>0</td>
</tr>
<tr>
<td>0-fill+1-fill+1-fill</td>
<td>2</td>
<td>(q_0)</td>
<td>0</td>
</tr>
<tr>
<td>1-fill+1-fill+1-fill</td>
<td>1</td>
<td>(q_8)</td>
<td>0</td>
</tr>
<tr>
<td>fill + L + fill (FLF)</td>
<td>1</td>
<td>(q_{10})</td>
<td>961d^2</td>
</tr>
<tr>
<td>L + fill + L (LFL)</td>
<td>1</td>
<td>(q_{10})</td>
<td>961d^2</td>
</tr>
<tr>
<td>Literal(not L) + L + fill (LFL)</td>
<td>2</td>
<td>(q_{10})</td>
<td>961d^2</td>
</tr>
<tr>
<td>L + fill + literal (not L) (LFL)</td>
<td>2</td>
<td>(q_{10})</td>
<td>961d^2</td>
</tr>
<tr>
<td>fill + fill + literal</td>
<td>2</td>
<td>(q_{11})</td>
<td>31d - 2387d^2</td>
</tr>
<tr>
<td>Literal (not L) + fill + fill</td>
<td>2</td>
<td>(q_{12})</td>
<td>360d^2</td>
</tr>
<tr>
<td>L + fill + fill</td>
<td>1</td>
<td>(q_{13})</td>
<td>31d - 2374d^2</td>
</tr>
<tr>
<td>NI2-L + fill + literal (NI2-LFL)</td>
<td>2</td>
<td>(q_{14})</td>
<td>360d^2</td>
</tr>
<tr>
<td>NI2-L + fill + fill (NI2-LFL)</td>
<td>1</td>
<td>(q_{14})</td>
<td>360d^2</td>
</tr>
<tr>
<td>Any type + NI2-L + fill (NI2-LFL)</td>
<td>2</td>
<td>(q_{16})</td>
<td>360d^2</td>
</tr>
<tr>
<td>Other cases</td>
<td>3</td>
<td>(q_{17})</td>
<td>601d^2</td>
</tr>
</tbody>
</table>

Since all kinds of codewords in SECOMPAX are covered by COMBAT, there is no need to create an independent table for SECOMPAX.

Similarly, TABLE III lists probabilities and corresponding compressed sizes of all possible compressed codewords in CONCISE. Detailed derivation processes of them can be seen in Appendix (see Proof 2).
Detailed derivation processes are similar. In SECOMPAX, the probabilities, corresponding derivation (denoted by $q$), expected size $L$ of each word combination in CONCISE can be calculated and shown below.

$$L_{\text{CONCISE}} \approx 1 + 62d - 210d^2$$  \hspace{1cm} (10)$$

According to the analysis above, the value of $L_{\text{SECOMPAX}}$ can be calculated and shown below.

$$L_{\text{SECOMPAX}} \approx 1 + 62d + 1922d^2$$  \hspace{1cm} (11)$$

Detailed derivation processes of the equation (10) and (11) are shown in Appendix (see Proof 3).

3) Decompression time complexity

The practical executing time of decompression process will be influenced by many factors like CPU load, free memory size and the programming language etc. Therefore the numbers of statements in decompression are used to approximate the decompression time.

a) Step 1: listing estimated decompression time in all kinds of codewords

A decompression process is usually composed of two steps: one is judgment and the other is assignment. In practice, assignment statements and conditional statements will be executed alternatively during the decompression process.

Let’s take an example to illustrate this process, if a compressed word (e.g. [FLF] codeword in COMBAT) is ready to be decoded, it would take $3t_1 + 5t_2$ because three assignment statements are needed to split it into three separate words and 5 judgment statements are needed to judge the type of two fills and the both the type and position of the dirty byte in this $L$. This decompression process is shown in Fig. 7.

Similarly, estimated decompression time of all the codewords in COMBAT, SECOMPAX and CONCISE can be calculated and listed in TABLE IV (“All” means that this codeword’s decompression time is the same for all the three algorithms).

b) Step 2: calculating the expected decompression time

Taking the probabilities calculated in Section 1) into consideration, we can calculate the expected values of decompression time (denoted by $T_{\text{COMBAT}}$, $T_{\text{SECOMPAX}}$ and $T_{\text{CONCISE}}$ separately) in a three-word combination and detailed derivation process is shown in Proof 4 in Appendix.

$$T_{\text{COMBAT}} \approx (1 + 124d)t_1 + (1 + 248d)t_2$$  \hspace{1cm} (12)$$
\[
T_{\text{SECOMPAX}} \approx (1 + 124d)t_1 + (1 + 248d)t_2
\]
\[
T_{\text{CONCISE}} \approx (1 + 124d)t_1 + (1 + 186d)t_2
\]

From equation (12) to (14), when \( d \) is approaching to zero, although the decomposition time consumed in CONCISE is less than that in SECOMPAX and COMBAT, however, it can be considered that querying time of the three algorithms at least stay in the same order of magnitude.

B. Compression of clustered bitmaps

1) Assumptions

Further assumptions are made that bitmap indexes follow two-state Markov process. Still only three chunks exist in the raw bit sequence and Taylor expansion can be applied to simplify our analysis. Besides, the first bit in a chunk is considered to be independent from the former chunk.

Variable \( p \) is introduced to represent the probability that a set bit is followed by an unset bit and \( q \) represents the probability that an unset bit is followed by a set bit.

Then the relationship between \( d \), \( p \) and \( q \) can be established because two type of bits are both likely to appear preceding a set bit. After simplification, an equation about variable \( d \), \( p \) and \( q \) can be written as follows.

\[
d = \frac{q}{p + q} \Rightarrow q = \left( \frac{d}{1-d} \right) \cdot p
\]

According to equation (15), it is obvious that the value of \( q \) approaches zero since \( d \) approaches zero. It can be assumed that \( p \) approaches one since the distribution of set bits is very sparse and continuous set bits are also very rare. This assumption is logical since busy network traffic is actually rare in general case.

In order to be more convenient, another variable \( r \) is introduced and it is assigned with the value of \( (1 - p) \). So \( r \) approaches zero. It can be further assumed that when \( q \) and \( r \) approach zero, they share the same convergent speed.

2) Size of compressed clustered bitmap

a) Step 1: Calculating Basic probabilities

In order to calculate the expected values of compressed size, Basic probabilities should be recalculated.

- \( p_1 \), \( p_2 \), \( p_7 \)

It is easy to calculate the values of \( p_1 \), \( p_2 \) and \( p_7 \), which are shown below.

\[
p_1 = (1 - d) \cdot (1 - q) \approx \frac{p}{p + q} \cdot (1 - q) \approx 1 - 31q + 466q^2 - q r
\]

\[
p_2 = d \cdot (1 - p) \approx 0
\]

\[
p_7 = 1 - p_7 \cdot p_2 = 31q - 466q^2 + q r
\]

The derivation processes of equation (16) and (17) are shown in Proof 5 in Appendix.

- \( p_3 \) AND \( p_4 \)

It would be much more complex to calculate the values of \( p_3 \) and \( p_4 \) because dirty byte may be dependent on the former bits and have influence on the following ones.

In this case, three kinds of subcases should be considered according to the position of dirty byte. In the following analyses, 0-dirty and 0-L will be considered at first. And in order to be more convenient to analyze, the bytes in a chunk are numbered. The leftmost byte is numbered as the zeroth byte and from leftmost to rightmost bytes, the number is in ascending order.

Case (1): In this case, the first or second byte in a chunk is 0-dirty. That means this 0-dirty would be influenced by the zeroth byte in this chunk and have impact on the third one.

Before calculating probabilities in this case, some auxiliary probabilities should be calculated at first.

According to the properties of Markov process, a basic recursive formula can be established as follows.

\[
\frac{p_1(0)}{p_1(1)} = \frac{1 - q}{q + r}
\]

Variables \( p_1(0) \) and \( p_1(1) \) represent the probability when the \( n \)-th bit in a chunk is unset bit or set bit separately. After iterative process and omitting terms of higher degree, the results will be (detailed derivation process is shown in Proof 6 in Appendix):

\[
\frac{p_1(0)}{p_1(1)} = \frac{1 - q + q^2 - q r}{q - q^2 + q r}
\]

When \( n = 8 \), the result is:

\[
\frac{p_1(0)}{p_1(1)} = \frac{1 - q^2 + q^2 r}{q - q^2 + q r}
\]

Another auxiliary variable is introduced here. If a group of bits ends with an unset bit and contains at least one set bit, the probability will be denoted by \( p_2(n) \). Variable \( n \) represents the number of containing bits.

This probability can be further divided into two parts according to the type of the \( (n-1) \)-th bit. When the \( (n-1) \)-th bit is an unset bit, then probability is \( p_2(n - 1) \cdot q \). When the \( (n-1) \)-th bit is a set bit, at least one set bit has appeared and the probability is \( p_2(n - 1) \cdot p \). Since \( p_2(1) \) equals \( q - q^2 + q r \), a recursive formula can be established as follows. And its general formula is also shown below.

\[
p_2(0(n)) = p_2(n - 1)(1 - q) \cdot (q - q^2 + q r) \cdot p
\]

Since the first term \( p_2(2) \) equals \( p \cdot q \) and \( p \) equals \( 1 - r \), the general term of \( p_2(n) \) can be written as follows after iterative process.

\[
p_2(n) = (1 - r)(1 - q + r) - (1 - q)^{n + 2}(1 - r)(1 - 2q + r)
\]

When \( n \) equals eight, the result of \( p_2(n) \) is needed in the following calculation and shown below:

\[
p_2(8) \approx 7q - 27q^2 - q r
\]

Similarly, we can use variable \( p_2(n) \) to represent the probability of a group of bits ending with an unset bit and containing at least one set bit. Obviously, this probability equals \( p_2(1) \cdot q - q^2 + q r \).

Based on these auxiliary variables, the probability of 0-L in this case (denoted by \( p_{\text{dirty}}(1) \)) is shown below and the derivation process is presented in Proof 7 in Appendix:

\[
p_{\text{dirty}}(1) = \left(1 - d \right) \cdot (1 - q) \cdot (p_1(0) \cdot (1 - q) + p_1(8) \cdot p)
\]
Case (2): If 0-dirty is the zeroth byte in a 0-L, then the first bit will not be influenced by the former bytes. In this case, variable $p_n(0)$, $p_n(1)$, $p_n(n)$ and $p_n(n)$ still work, but their values get changed.

Because this 0-dirty is not affected by the former bytes, the values of $p_n(0)$ and $p_n(1)$ are $d$ and $1 - d$ separately. In order to be distinct, $p_n(n)$ is redefined as $p_n'(n)$ in this case. When $n$ equals seven, the value of $p_n'(n)$ is:

$$p_n'(7) \approx 6q - 21q^2$$

(26)

The value of $p_n'(7)$ still equals $q - q^2 + q r$. Based on these values, the probability of 0-dirty (denoted by $p_{d \text{rty}, 2}$) is:

$$p_{d \text{rty}, 2} = p_n'(7)^{24} + p_n'(7)p(1 - q)^{23} \approx 7q - 189q^2 + q r$$

(27)

Case (3): The third byte in 0-L is 0-dirty. This 0-dirty will have no influence on the following bytes. Before calculating the probability, another variable $p_3(n)$ is introduced to denote the probability that a group of bits (composed of $n$ bits) contain at least one set bit. Then another recursive formula will be established as follows. Since the first term $p_3(1)$ equals $q$, its general term is also shown below.

$$p_3(n) = p_3(n - 1) + (1 - q)^n q$$

(28)

So the probability $p_{d \text{rty}, 3}$ in this case is:

$$p_{d \text{rty}, 3} = (1 - d)(1 - q)^2 p_3(8) \approx 8q - 212q^2$$

(30)

Finally, the value of $p_3$ is the sum of probabilities in the three cases discussed above:

$$p_3 = p_{d \text{rty}, 1} + p_{d \text{rty}, 2} + p_{d \text{rty}, 3} \approx 31q - 823q^2 - q r$$

(31)

Similarly, the value of $p_3$ can be calculated out. But it is obvious that $p_3$ can be divided by the expression $r^2$, which is an item of high degree. So its approximate value is:

$$p_3 \approx 0$$

(32)

- $p_3$ AND $p_6$

The values of $p_3$ and $p_6$ will be calculated here. Likewise, $p_3$ will be considered at first. And several cases are discussed below. In order to distinguish the two dirty bytes existing in a chunk, they are named as left 0-dirty (or 1-dirty) and right 0-dirty (or 1-dirty) according to their positions in this chunk.

Case (1) - two dirty bytes are not adjacent:

Subcase (1): If the zeroth byte is the left 0-dirty in this chunk, this case is just extended from subsection A. So the probability (denoted by $p_{d \text{rty}, 1}$) can be derived as follows. $p_{d \text{rty}, 1}$ is:

$$p_{d \text{rty}, 1} = p_n'(7)(1 - q)^3 p_3(8)(1 - q)^4 + p_n'(8)p(1 - q)^7 + p_n'(7)p_3(8)(1 - q)^5 + p_3(8)p(1 - q)^7 \approx 56q^2$$

(33)

Subcase (2): if the first byte in this chunk is the left 0-dirty, the probability (denoted by $p_{d \text{rty}, 2}$) is:

$$p_{d \text{rty}, 2} = (1 - d)(1 - q)^3 p_3(8)(1 - q)^4 + p_3(8)p(1 - q)^7 \approx 6q^2$$

(34)

Case (2) - two dirty bytes are adjacent:

This is the case that two dirty bytes are adjacent in a 0-NIL-1 codeword. A probability should be calculated at first. In order to be distinct, the variable $p_3(n)$ is redefined as $p_n'(n)$ here. Obviously $p_n'(n)$ shares the same recursive formula with $p_n(n)$ but they have different first terms. The first term, $p_n'(2)$ is $(1 - p) p$. After iterative process, the general term is:

$$p_n'(n) = (1 - r)(1 - q + r) - (1 - r)(1 - q)^{n - 1}$$

(35)

When $n$ equals eight, the value is (after simplification):

$$p_n'(8) - 6q - 21q^2 + r - 6q r$$

(36)

Subcase (1): If the zeroth byte is the left 0-dirty, the derivation process is similar to the former cases. Then after simplification, the probability (denoted by $p_{d \text{rty}, 3}$) is shown below and detailed derivation process is shown in Appendix (see proof 8):

$$p_{d \text{rty}, 3} = 55q^2 + q r$$

(37)

Subcase (2): When the first byte is the left 0-dirty similarly, the following equation can be drawn. The probability is denoted by $p_{d \text{rty}, 4}$:

$$p_{d \text{rty}, 4} = (1 - d)(1 - q)^7 p_3(8)(1 - q)^4 + p_3(1 - q)^7 \approx 63q^2$$

(38)

Subcase (3): Then the case that the second byte is the left 0-dirty is discussed. Before this discussion, another variable $p_8(n)$ is introduced. It shares the same recursive formula with $p_3(n)$. But it denotes the case that a group of bits follow a set bit. The value of first term $p_8'(1)$ is 1 - $p$. So the general term is:

$$p_8(n) = 1 - (1 - q)^n + r - q$$

(39)

When $n$ equals eight, the value is:

$$p_8'(8) = 7q - 28q^2 + r$$

(40)

So when the left 0-dirty is the second byte, corresponding probability can be derived in the same way. The probability in this case is denoted by $p_{d \text{rty}, 5}$ and it is shown as follows.

$$p_{d \text{rty}, 5} = (1 - d)(1 - q)^5 p_3(8)p_8^4 + p_3(1 - q)^7 \approx 63q^2 + q r$$

(41)

Finally, the value of $p_8$ is:

$$p_8 = p_{d \text{rty}, 1} + p_{d \text{rty}, 2} + p_{d \text{rty}, 3} + p_{d \text{rty}, 4} + p_{d \text{rty}, 5} \approx 307q^2 + 2q r$$

(42)

In terms of $p_8$, there exist at least 14 continuous set bits. This is a term of high degree, which can be divided by the expression $r^4$. When the value of $r$ approaches zero, this value can be also considered to approach zero.

- $p_8$

Since only a set bit exists in an $N$-fill, according to the position of the only set bit, the calculation of $p_8$ should be divided into three cases and the results are shown below.

$$p_8 = d^q - 10C_2d^{10}(1 - d)p q(1 - q)^8 + 4(1 - d) q(1 - q)^9$$

(43)
\( q - 28q^2 \)

The first term denotes that the only set bit lies in the leftmost bit and the third denotes that the only set bit lies in the rightmost bit. The second term denotes other possible positions.

b) Step 2: calculating probabilities of each codeword

After calculating Basic probabilities, similar to Subsection A, the probabilities listed in TABLE II and TABLE III can be recalculated and listed in TABLE V and TABLE VI respectively.

### TABLE V. Probabilities in COMBAT

<table>
<thead>
<tr>
<th>Denotation</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1 )</td>
<td>( 1-93q+4281q^2-3q )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( 31q - 2745q^2 - q )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( 961q^2 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( 961q^2 )</td>
</tr>
<tr>
<td>( q_5 )</td>
<td>( 31q - 2388q^2 + q )</td>
</tr>
<tr>
<td>( q_6 )</td>
<td>( 357q^2 + 2q )</td>
</tr>
<tr>
<td>( q_7 )</td>
<td>( 31q - 2745q^2 - q )</td>
</tr>
<tr>
<td>( q_8 )</td>
<td>( 301q^2 )</td>
</tr>
<tr>
<td>( q_{10} )</td>
<td>( 301q^2 )</td>
</tr>
<tr>
<td>( q_{11} )</td>
<td>( 301q^2 )</td>
</tr>
<tr>
<td>( q_{12} )</td>
<td>( 415q^2 - 4q )</td>
</tr>
</tbody>
</table>

### TABLE VI. Probabilities in CONCISE

<table>
<thead>
<tr>
<th>Denotation</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q'_1 )</td>
<td>( 1-93q+4281q^2-3q )</td>
</tr>
<tr>
<td>( q'_2 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( q'_3 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( q'_4 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( q'_5 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( q'_6 )</td>
<td>( 31q - 2388q^2 + q )</td>
</tr>
<tr>
<td>( q'_7 )</td>
<td>( 435q^2 + 30q )</td>
</tr>
<tr>
<td>( q'_8 )</td>
<td>( 31q - 2823q^2 - 2q )</td>
</tr>
<tr>
<td>( q'_{10} )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( q'_{11} )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( q'_{12} )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( q'_{13} )</td>
<td>( 31q - 1862q^2 - 29q )</td>
</tr>
</tbody>
</table>

c) Step 4: calculating expected values of compressed size

Similar to subsection A, the expected values of compressed size using COMBAT, SECOMPAX and CONCISE can be calculated and shown as follows respectively. And the derivation process is similar to that in subsection A.

\[
\bar{T}_{\text{COMBAT}} \approx 1 + 31q + 362q^2 - q \tag{44}
\]
\[
\bar{T}_{\text{SECOMPAX}} \approx 1 + 62q - 218q^2 + 6q \tag{45}
\]
\[
\bar{T}_{\text{CONCISE}} \approx 1 + 62q + 899q^2 + 62q \tag{46}
\]

As shown in equations of (44) to (46), the coefficient of monomial term in COMBAT is also smaller than that in SECOMPAX and CONCISE, when \( q \) and \( r \) approach zero. It can be predicted that COMBAT can provide more spatial savage comparing to SECOMPAX and CONCISE.

c) Step 4: calculating expected values of compressed size

Similar to subsection A, the expected values of compressed size can be also estimated and values in TABLE IV still hold. The values of \( \bar{T}_{\text{COMBAT}} \), \( \bar{T}_{\text{SECOMPAX}} \) and \( \bar{T}_{\text{CONCISE}} \) can be recalculated and their results are also shown below. Their derivation processes are similar to those in subsection A.

\[
\bar{T}_{\text{COMBAT}} = t_1(1+124q)+t_2(1+252q) \tag{47}
\]
\[
\bar{T}_{\text{SECOMPAX}} = t_1(1+124q)+t_2(1+188q) \tag{48}
\]
\[
\bar{T}_{\text{CONCISE}} \approx t_1(1+124q)+t_2(1+126q) \tag{49}
\]

By comparing results above, the decompression time is at least in the same order of magnitude. Although the coding scheme in COMBAT contains more operations, querying time is still acceptable.

### IV. EVALUATION RESULTS

In this section, evaluation results including both the spatial and the temporal performance are presented.

A. Evaluation results on spatial performance

When bitmap indexes are uniformly distributed, Fig. 8 shows evaluation results where the density interval ranges from 0 to 0.01. When bitmap indexes are clustered, corresponding evaluation results are shown in Fig. 9 to Fig. 11.

As shown in these figures, COMBAT has a higher compression ratio comparing to the other two algorithms, which confirms the analysis results in Section III. It is concluded that COMBAT has a better spatial performance than the other two algorithms whether a bitmap index is uniformly distributed or clustered.
The decompression time values for the three bitmap algorithms, i.e., COMBAT, SECOMPAX, and CONCISE, are shown in Fig. 12. The results indicate that CONCISE achieves the best improvement in spatial performance, followed by SECOMPAX and then COMBAT. This is consistent with the theoretical analysis and evaluation results presented in Section III.

When evaluating the temporal performance, the results contain up to four variables, i.e., $q$, $r$, $t_1$, and $t_2$, some of which should be set constant values to facilitate the evaluation. For example, the values of $t_1$ and $t_2$ can be constant within a CPU. It is necessary to set proper values for them.

For convenience, $t_1$ is set as one unit of time, denoted as $t$ and $t_2$ as 0.2 unit of time, that is, 0.2$t$. These values are reasonable because a certain conditional statement will be often followed by more than one non-conditional statements. More execution time on assignment operations is expected. Based on this assumption, evaluation results in uniformly distributed bitmaps are presented in Fig. 12. When $d$ varies in the interval of $(0.01)$, temporal performance is actually almost the same among the three algorithms, at least in the same order of magnitude, which is in accordance with analytical results in Section III.

Corresponding results in clustered bitmap are shown in Fig. 13-15. As shown in these figures, with the change of the values of $r$ and $q$, the three algorithms share the nearly the same trend as well as similar decompression time values, which also concurs with analytical results in Section III.

V. CONCLUSION

In this paper, a general analytical model is proposed to analyze spatial and temporal performance of three bitmap index compression algorithms, i.e., COMBAT, SECOMPAX, and CONCISE. When a bitmap index is sparse, COMBAT achieves the best improvement in spatial performance, which is proved by both the theoretical analysis and evaluation results. Besides, this analytical model can not only be applied in these three algorithms, but also be used to any bitmap index compression algorithms that are derived from WAH. Based on appropriate assumptions, the general analysis procedures are similar, including listing all kinds of possible compressed word combinations, calculating the probabilities of each codeword and then working out the expected values of compressed size after encoding and corresponding decompression time. This model can contribute to analyzing both the spatial and temporal performance results of a new invented bitmap compression algorithm before conducting real experiments.

REFERENCES

2011 ACM SIGMOD International Conference on Management of

compression, archiving and indexing of streaming network traffic,”

compressing bitmap indexes with word aligned hybrid,” In Database
and Expert Systems Applications, Springer Berlin Heidelberg,

millions of packets per second using GPUs,” In Proceedings of the
2013 conference on Internet measurement conference, pp.327-332.
ACM, 2013.

implementation of the PWAH algorithm for compressing bitmaps,”

Huang. “SECOMPAX: A bitmap index compression algorithm.” In
23rd International Conference on Computer Communication and
Networks (ICCCN), IEEE, pp. 1-7, 2014.

“PWAH: a bitmap index compressing scheme based on PWAH.”
In Proceedings of the tenth ACM/IEEE symposium on Architectures
for networking and communications systems, ACM, pp. 257-258,
2014.

Compression Scheme of Medium-Sparse Bitmaps,” in the
Third International Conference on Advances in Databases,
Knowledge, and Data Applications (DBKDA 2011), pp. 192-195.

(2014).

"BreadZip: a combination of network traffic data and bitmap index
encoding algorithm." In Systems, Man and Cybernetics (SMC), 2014

[26] R. Slechta, J. Sawin, B. McCamish, D. Chiu and G. Canahuate,
A tunable compression framework for bitmap indices, in Data
Engineering (ICDE 2014), pp. 484-495. IEEE.

composable integer set,” In Information Processing Letters, 110(16),

[28] Yinjun Wu, Zhen Chen, Yu-hao Wen, Junwei Cao, “COMBAT: a
new bitmap index compression algorithm for Big Data”, in
submission.

Hakmoun, G. Peng. “A Survey of Bitmap Index Compression
Algorithms for Big Data,” Tsinghua Science and Technology, 20(1),
February 2015

APPENDIX

Proof 1. This is to prove the probabilities listed in TABLE II.

\[
q_1 - p_1 p_2 \approx 1 - 93d + 2487d^2
q_2 - q_3 = p_1 p_2 p_2 \approx 0
q_4 = q_5 - q_6 p_2 \approx 0
q_6 = p_2 p_2 \approx 0
q_7 = (p_1 p_2) (p_1 p_2) \approx 31d - 247d^2
q_8 = (p_1 p_2) (p_1 p_2) \approx 961d^2
q_9 = (p_1 p_2) (p_1 p_2) \approx 961d^2
q_{10} = (p_1 p_2) (p_1 p_2) \approx 961d^2
q_{11} = p_1 p_2 p_2 \approx 31d - 2387d^2
q_{12} = p_1 p_2 p_2 \approx 30d \leq 360d^2
q_{13} = (p_1 p_2) (p_1 p_2) \approx 31d - 247d^2
q_{14} = (p_1 p_2) (p_1 p_2) \approx 360d^2 (1 - 3d + 6d^2) (3d - 6d^2) \approx 0
q_{15} = (p_1 p_2) (p_1 p_2) \approx 60d^2
q_{16} = (p_1 p_2) (p_1 p_2) \approx 60d^2 (1 - 3d + 6d^2) \approx 36d^2
q_{17} = 1 - \Sigma q_i \approx 16d^2
\]

Proof 2. This is to prove the probabilities in TABLE III.

\[
q_1 = p_1 p_2 p_1 \approx 1 - 93d + 2487d^2
q_2 = p_2 p_2 \approx d^2 \approx 0
q_3 = q_4 = q_5 = q_6 = q_7 = q_8 = q_9 = q_{10} = q_{11} = q_{12} = q_{13} \approx 1 - 62d - 210d
\]

\[
T_{\text{CONCISE}} = q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + q_7 + q_8 + q_9 + q_{10} + q_{11} + q_{12} + q_{13} + 1 + 62d - 210d
\]

Proof 4. This is to prove the equation of (12)

\[
T_{\text{SECOMPAX}} = q_1 (t_1 + t_2) + 2q_1 (t_1 + t_2) + 2q_1 (t_1 + t_2) + 2q_1 (t_1 + t_2) + 2q_1 (t_1 + t_2) + q_1 (t_1 + t_2) + q_1 (t_1 + t_2) + q_1 (t_1 + t_2) + q_1 (t_1 + t_2) + q_1 (t_1 + t_2)
\]

Proof 5. This is to prove the equation of (16) and (17)

In a 0-fill, the first bit is an unset bit and it can be considered an independent bit. So the value of it is (1 - d). Then the rest 30 continuous bits are also unset bits. Since they all follow an unset bit, the probability is (1 - q)^{30}. So the final result of p_1 is (1 - d) (1 - q)^{30}. Using the equation of (15) and replacing p with r, the following equation can be deduced: \( p_2 = \frac{1 - r^2}{1 - r + q}\ (1 - q)^{30}\). After using Taylor expansion and omitting terms of high degree, the final approximate result is \( p_2 \approx 31q + 466q^2 - q r \).
In a 1-fill, the first bit is set bit and this bit is followed by 30 continuous set bits. The probability is $d (1 - p)^{30}$. After replacing $d$ with variable $q$ and $r$ and replacing $p$ with $r$, the result is $\frac{1}{1 + r + q} r^{30}$. Obviously, $r^{30}$ is a term of high degree and $\frac{1}{1 + r + q}$ is a finite value. So when $r$ approaches zero, the result can be seen as $p_2 \rightarrow 0$

Proof 6. This is to prove the equation of (20)

Mathematical induction is applied to prove this conclusion. The matrix $\begin{pmatrix} 1 - q & 1 - r \\ q & r \end{pmatrix}$ is denoted by $A$

When $n$ equals 3, $A^3 = \begin{pmatrix} 1 - q & 1 - r \\ q & r \end{pmatrix}^3 = \begin{pmatrix} 1 - q + q^2 + q^3 - q r + 2 q^2 r - q r^2 & 1 - q + q^2 + q^3 - q r + 2 q^2 r - q r^2 \\ q - q^2 + q^3 - q r + 2 q^2 r - q r^2 & q - q^2 + q^3 - q r + 2 q^2 r - q r^2 \end{pmatrix}$

After omitting items of higher degree, the result is $\begin{pmatrix} 1 - q + q^2 - q r & 1 - q + q^2 - q r \\ q - q^2 + q^3 - q r & q - q^2 + q^3 - q r \end{pmatrix}$, which is in agreement with the conclusion.

Now assuming that when $n$ equals $k$ (an integer which is bigger than three), the conclusion is still established.

Then when $n$ equals $k+1$, $A^{k+1} = A^k A = \begin{pmatrix} 1 - q + q^2 - q r & 1 - q + q^2 - q r \\ q - q^2 + q^3 - q r & q - q^2 + q^3 - q r \end{pmatrix} \begin{pmatrix} 1 - q & 1 - r \\ q & r \end{pmatrix} = \begin{pmatrix} 1 - q + q^2 - q r & 1 - q + q^2 - q r \\ q - q^2 + q^3 - q r & q - q^2 + q^3 - q r \end{pmatrix}$

So this equation is proved.

Proof 7. This is to prove the equation of (25)

In this case, 0-dirty can locate in either the first byte or the second byte. It can be assumed that this 0-dirty is the first byte in the following analyses and other bytes are all composed of unset bits. In the zeroth byte, except that the first bit is independent, other bits all follows an unset bit. So the probability is $(1 - d) (1 - q)^6$. In the second byte, the first bit of it must be influenced by the last bit of the first byte which is 0-dirty. When the 0-dirty ends with an unset bit, the probability for the first bit to be unset bit is $p_0(8) (1 - q)$. When the case is opposite, the probability is $p_1(8) p$. Then the probability of the rest 15 bits to be unset bit is $(1 - q)^{15}$. The final result of $p_1^{\text{dirty}}$ is $C_2(1-d) (1 - q)^{21} (p_0(8) (1 - q) + p_1(8) p)$. According to the value of $p_0(8)$ and $p_1(8)$, the simplified result is $p_1^{\text{dirty}} \approx 16q - 422q^2 - 2q r$

Proof 8: This is to prove the equation of (37)

According to the type of the last bit in this dirty byte, this probability can be divided into two parts. If the last bit is an unset bit, then according to the type of the last bit in the following dirty byte, the probability can be further divided into two subparts. Similar to Proof 8, the probability in this part is $p_1^* (7) (p_0(8) (1 - q)^{16} + p_1(8) p (1 - q)^{15})$. Likewise, if the last bit is a set bit, this part of probability is $p_1^* (7) (p_0^* (8) (1 - q)^{16} + p_1^* (8) p (1 - q)^{15})$. So the value of $p_3^{\text{dirty}}$ is the sum of these two parts of probability and it is shown below.

$\begin{align*}
    p_3^{2\text{dirty}} &= p_1^* (7) (p_0(8) (1 - q)^{16} + p_1(8) p (1 - q)^{15}) + p_1^* (7) (p_0^* (8) (1 - q)^{16} + p_1^* (8) p (1 - q)^{15}) \\
    \text{Then after specific value is assigned and simplification process is done, the value of } p_3^{2\text{dirty}} \text{ is: } p_3^{2\text{dirty}} &\approx 55q^2 + q r
\end{align*}$