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

Yinjun Wu, Zhen Chen, Yuhao Wen, Junwei Cao, Wenxun 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 Ternary 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], EWAH [15], PWAH [16] , COMPAX [17], GPU-WAH [18-19], GPU-PLWAH [20], SECOMPAX [21], PLWAH+[22], DFWAH [23], roaring 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.

RowID	type	bitmap index				
		tpye=1	type=2	type=3	type=4	type=5
1	2	0	1	0	0	0
2	1	1	0	0	0	0
3	4	0	0	0	1	0
4	3	0	0	1	0	0
5	1	1	0	0	0	0
6	1	1	0	0	0	0
7	5	0	0	0	0	1

Fig. 1 An example of bitmap index

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 *fill*s of the same kind.

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

B. 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 I. Terminology and corresponding explanations

Terminology	Explanations
unset bit	A bit that is “0”
set bit	A bit that is “1”
chunk	Consecutive 31 bits in a bit sequence
Fill	Defined as a *chunk* composed of the same kinds of bit.
	A *0-fill* is defined as a *fill* only composed of *unset bit*s.
	A *1-fill* is defined as a *fill* only composed of *set bit*s.
literal	A *chunk* that is not *fill*.
dirty byte	Defined as a byte in a *chunk* containing *set bit*s or *unset bits* exclusively.
	A *0-dirty* is defined as a byte in a *chunk* containing *set bits* exclusively.
	A *1-dirty* is defined as a byte in a *chunk* containing *unset bits* exclusively.
L	Defined as a *chunk* with one *dirty byte*
	A *0-L* is defined as a *chunk* with a *0-dirty*
	A *1-L* is defined as a *chunk* with a *1-dirty*
NI2-L	Defined as a *chunk* with two *dirty bytes*
	0-L is defined as a *chunk* with two *0-dirtys*
	1-L is defined as a *chunk* with two *1-dirtys*

C. 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 *chunk*s, *L* and *fill* (but without another *L* following), these two continuous *chunk*s 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 *chunk*s can be merged into a new one, called [NI2-LF]. The composition of [LF] and [NI2-LF] is also listed in Fig. 4 and Fig. 5 repectively.

D. 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

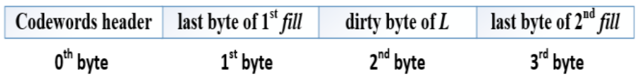

Fig. 2. [FLF] codeword in SECOMPAX

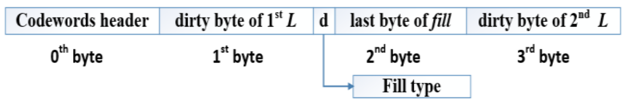

Fig. 3. [LFL] codeword in SECOMPAX

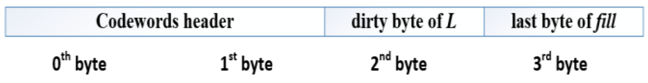

Fig. 4. [LF] codeword in COMBAT

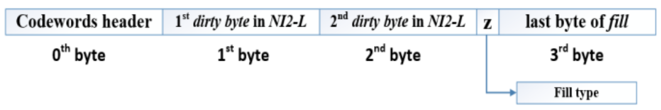

Fig. 5. [NI2-LF] codeword in COMBAT

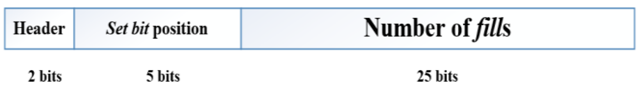

Fig. 6. [NL-F] codeword in CONCISE

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 *chunk*s 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 bit*s is denoted by a variable *d*.
- The variable *d* is independent from the distribution of *set bit*s 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 bit*s, the probability (denoted by p_1) will be

$$p_1 = (1-d)^{31} \approx 1 - 31d + 465d^2 \tag{1}$$

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)).

$$p_2 = d^{31} \approx 0 \tag{2}$$

After getting the value of p_1 and p_2, the probability of *literal* (denoted by p_7) can be written below:

$$p_7 = 1 - p_1 - p_2 = 1 - (1-d)^{31} - d^{31} \approx 31d - 465d^2 \tag{3}$$

- Special *Basic probabilities* for SECOMPAX and COMBAT:

In terms of the probability of *0-L* (denoted by p_3) it can be deduced below:

$$\begin{aligned} p_3 &= C_3^1(1-(1-d)^8)(1-d)^{23} + (1-(1-d)^7)(1-d)^{24} \\ &\approx 31d - 825d^2 \end{aligned} \tag{4}$$

The first term in equation (4) denotes the case that the *dirty byte* lies in the leftmost 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_4) can be denoted below.

$$p_4 = C_3^1(1-d^8)\,d^{23} + (1-d^7)\,d^{24} \approx 0 \tag{5}$$

The results of the probabilities of *0-NI2-L* and *1-NI2-L* (denoted by p_5 and p_6 separately) are shown below and their derivation process is similar to the former one.

$$p_5 \approx 360d^2 \tag{6}$$

$$p_6 = 3((1-d^8)^2 + 3(1-d^7)(1-d^8))\,d^{23} \approx 0 \tag{7}$$

- 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.

$$p_8 = C_{31}^1 d(1-d)^{30} \approx 31d - 930d^2 \tag{8}$$

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 $\overline{L_{COMBAT}}$) can be calculated out and detailed derivation processes of these probabilities are shown in Appendix (see Proof 1).

TABLE II. Probability and corresponding compressed size of each word combination in COMBAT

word combination	compressed size	Symbol	Probability value
0-fill+0-fill+0-fill	1	q_1	1-93d+4278d^2
1-fill+0-fill+0-fill	2	q_2	0
0-fill+0-fill+1-fill	2	q_3	0
1-fill+1-fill+0-fill	2	q_4	0
0-fill+1-fill+1-fill	2	q_5	0
1-fill+1-fill+1-fill	1	q_6	0
fill + *L* + *fill* ([FLF])	1	q_7	31d - 2747d^2
L + *fill* + *L*([LFL])	1	q_8	961d^2
Literal(not L) + *L* + *fill*([LF])	2	q_9	961d^2
L + *fill* + *literal*(not *L*,[LF])	2	q_{10}	961d^2
fill + *fill* + *literal*	2	q_{11}	31d - 2387d^2
Literal (not *L*) + *fill* + *fill*	2	q_{12}	360d^2
L+ *fill* + *fill*	1	q_{13}	31d - 2747d^2
NI2-L + *fill* +*literal* ([NI2-LF])	2	q_{14}	360d^2
NI2-L + *fill* +*fill* ([NI2-LF])	1	q_{15}	360d^2
Any type + *NI2-L* + *fill* ([NI2-LF])	2	q_{16}	360d^2
Other cases	3	q_{17}	601d^2

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).

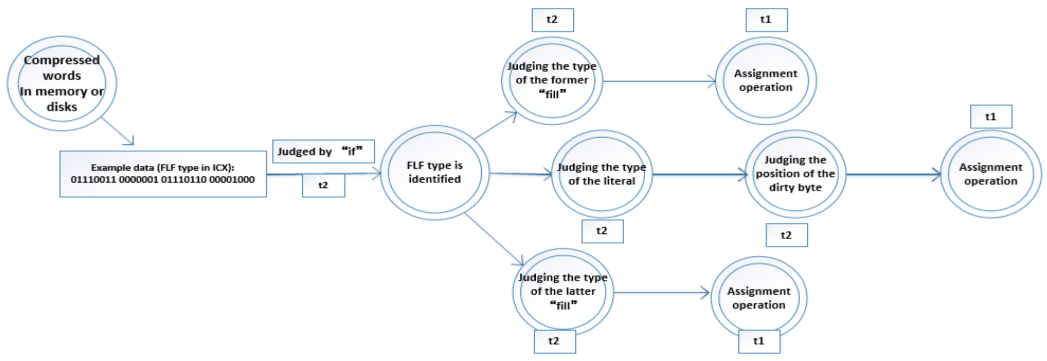
Compressed words In memory or disks
Example data (FLF type in ICX): 01110011 0000001 01110110 00001000
Judged by "if"
t2
FLF type is identified
Judging the type of the former "fill"
t2
Assignment operation
t1
Judging the type of the literal
t2
Judging the position of the dirty byte
t2
Assignment operation
t1
Judging the type of the latter "fill"
t2
Assignment operation
t1

Fig. 7. An example of time evaluation in the process of decompression

TABLE III. Probability and corresponding compressed size of each word combination in CONCISE

Word combination	compressed size	Symbol	Probability value
0-fill+0-fill+0-fill	1	q_1^c	$1 - 93d + 4278d^2$
1-fill+1-fill+1-fill	1	q_2^c	0
0-fill+0-fill+1-fill	2	q_3^c	0
1-fill+0-fill+0-fill	2	q_4^c	0
1-fill+1-fill+0-fill	2	q_5^c	0
0-fill+1-fill+1-fill	2	q_6^c	0
fill + fill + literal	2	q_7^c	$31d - 2387d^2$
literal(not N-fill) + fill + fill (the same type of *fill*)	2	q_8^c	$465d^2$
N-fill+0-fill+ 0-fill	1	q_9^c	$31d - 2852d^2$
N-fill+0-fill+ 1-fill	2	q_{10}^c	0
N-fill+1-fill+ 0-fill	2	q_{11}^c	0
N-fill+1-fill+1-fill	2	q_{12}^c	0
Any type of word + *N-fill + 0-fill*	2	q_{13}^c	$31d - 1891d^2$
Other cases	3	q_{14}^c	$2387d^2$

c) Step 3: calculating expected values of compressed size

According to TABLE II, the value of $\overline{L_{COMBAT}}$ can be derived below:

$$\overline{L_{COMBAT}} = q_1 + 2q_2 + 2q_3 + 2q_4 + 2q_5 + q_6 + q_7 + q_8 + 2q_9 + 2q_{10} + 2q_{11} + 2q_{12} + q_{13} + 2q_{14} + q_{15} + 2q_{16} + 3q_{17} \approx 1 + 31d + 496d^2 \quad (9)$$

Similar processes can be conducted to work out the expected values of compressed size with SECOMPAX (denoted by $\overline{L_{SECOMPAX}}$). Since COMBAT includes all of the codewords of SECOMPAX, corresponding derivation processes are similar. In SECOMPAX, the probabilities, q_{11}, q_{12}, q_{16}, q_{17} and q_{18}, do not exist. Besides, the compressed size of q_{15} is 2 in SECOMPAX, because this word combination cannot be fully compressed due to lack of the codeword [LF]. Therefore the value of $\overline{L_{SECOMPAX}}$ is shown as follows.

$$\overline{L_{SECOMPAX}} \approx 1 + 62d - 210d^2 \quad (10)$$

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

$$\overline{L_{CONCISE}} \approx 1 + 62d + 1922d^2 \quad (11)$$

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 *fill*s 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).

TABLE IV. Decompression time estimation of all the codewords

Codewords	Time	Algorithms
Fill	$t_1 + t_2$	All
Literal	$t_1 + t_2$	All
[FLF]	$3t_1 + 5t_2$	COMBAT, SECOMPAX
[LFL]	$3t_1 + 6t_2$	COMBAT, SECOMPAX
[LF]	$2t_1 + 4t_2$	COMBAT
[NI2-LF]	$3t_1 + 6t_2$	COMBAT
NL-F	$2t_1 + 3t_2$	CONCISE

b) Step 2: calculating the expetected decompression time

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

$$\overline{T_{COMBAT}} \approx (1 + 124d)t_1 + (1 + 248d)t_2 \quad (12)$$

$$\overline{T_{SECOMPAX}} \approx (1+124d)t_1 + (1+248d)t_2 \tag{13}$$

$$\overline{T_{CONCISE}} \approx (1+124d)t_1 + (1+186d)t_2 \tag{14}$$

From equation (12) to (14), when d is approaching to zero, although the decompression 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 *chunk*s 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.

$$d = \frac{q}{p+q} \Leftrightarrow q = \left(\frac{d}{1-d}\right)\cdot p \tag{15}$$

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 bit*s is very sparse and continuous *set bit*s are also very rare. This assumption is logical since busty 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.

$$p_1 = (1-d)(1-q)^{30} = \frac{p}{p+q}(1-q)^{30} \approx 1\text{-}31q+466q^2\text{-}q\,r \tag{16}$$

$$p_2 = d\,(1-p)^{30} \approx 0 \tag{17}$$

$$p_7 = 1 - p_1 - p_2 \approx 31q - 466q^2 + q\,r \tag{18}$$

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.

$$\begin{pmatrix} p_n(0) \\ p_n(1) \end{pmatrix} = \begin{pmatrix} 1\text{-}q & 1\text{-}r \\ q & r \end{pmatrix} \begin{pmatrix} p_{n\text{-}1}(0) \\ p_{n\text{-}1}(1) \end{pmatrix} \tag{19}$$

Variables $p_n(0)$ and $p_n(1)$ 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):

$$\begin{pmatrix} p_n(0) \\ p_n(1) \end{pmatrix} = \begin{pmatrix} 1\text{-}q+q^2\text{-}q\,r & 1\text{-}q+q^2\text{-}q\,r \\ q-q^2+q\,r & q-q^2+q\,r \end{pmatrix} \begin{pmatrix} p_0(0) \\ p_0(1) \end{pmatrix} (n>2) \tag{20}$$

When $n = 8$, the result is:

$$\begin{pmatrix} p_8(0) \\ p_8(1) \end{pmatrix} = \begin{pmatrix} 1\text{-}q+q^2\text{-}q\,r \\ q-q^2+q\,r \end{pmatrix} \tag{21}$$

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 $p_0(n)$. 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 $(n\text{-}1)$-*th* bit is an *unset bit*, then probability is $p_0(n\,\text{-}1)\,(1\text{-}q)$. When the *(n-1)-th* bit is a *set bit*, at least one *set bit* has appeared and the probability is $p_{n\text{-}1}(1)\,p$. Since $p_{n\text{-}1}(1)$ equals $q-q^2+q\,r$, a recursive formula can be established as follows. And its general formula is also shown below.

$$p_0(n) = p_0(n\,\text{-}1)\,(1\text{-}q) + (q - q^2 + q\,r)\,p \tag{22}$$

Since the first term $p_0(2)$ equals $p\cdot q$ and p equals $1\text{-}\,r$, the general term of $p_0(n)$ can be written as follows after iterative process,.

$$p_0(n) = (1-r)(1-q+r) - (1-q)^{n\text{-}2}(1-r)(1-2q+r) \tag{23}$$

When n equals eight, the result of $p_0(n)$ is needed in the following calculation and shown below:

$$p_0(8) \approx 7q - 27q^2 - q\,r \tag{24}$$

Similarly, we can use variable $p_1(n)$ 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 $p_n(1)$, $q-q^2+q\,r$.

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

$$p_{dirty,\,1} = C_2^1(1\text{-}d)\,(1-q)^{21}(p_0(8)\,(1\text{-}q) + p_1(8)\,p) \tag{25}$$

$$\approx 16q - 422q^2 - 2q\,r$$

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 $p_n(0)$, $p_n(1)$, $p_0(n)$ and $p_1(n)$ still work, but their values get changed.

Because this *0-dirty* is not affected by the former bytes, the values of $p_8(0)$ and $p_8(1)$ are *d* and 1 - *d* separately. In order to be distinct, $p_0(n)$ is redefined as $p^*_0(n)$ in this case. When *n* equals seven, the value of $p^*_0(n)$ is:

$$p^*_0(7) \approx 6q - 21q^2 \qquad (26)$$

The value of $p_1(7)$ still equals $q - q^2 + q\,r$. Based on these values, the probability of *0-dirty* (denoted by $p_{dirty,\,2}$) is:

$$p_{dirty,\,2} = p_0(7)(1-q)^{24} + p_1(7)p(1-q)^{23} \approx 7q - 189q^2 + q\,r \qquad (27)$$

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 $p_k(n)$ 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 $p_k(1)$ equals q, its general term is also shown below.

$$p_k(n) = p_k(n-1) + (1-q)^{n-1}q \qquad (28)$$

$$p_k(n) = 1 - (1-q)^n \qquad (29)$$

So the probability $p_{dirty,\,3}$ in this case is:

$$p_{dirty,\,3} = (1-d)\,(1-q)^{22}p_k(8) \approx 8q - 212q^2 \qquad (30)$$

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

$$p_3 = p_{dirty,\,1} + p_{dirty,\,2} + p_{dirty,\,3} \approx 31q - 823q^2 - q\,r \qquad (31)$$

Similarly, the value of p_4 can be calculated out. But it is obvious that p_4 can be divided by the expression r^{22} which is an item of high degree. So its approximate value is:

$$p_4 \approx 0 \qquad (32)$$

- P_5 AND P_6

The values of p_5 and p_6 will be calculated here. Likewise, p_5 will be considered at first. And several cases are discussed below. In order to distinct 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 $p_{2\text{-}dirty,\,1}$) can be derived as follows.

$$\begin{aligned} p_{2\text{-}dirty,\,1} = {} & p^*_0(7)\,(1-q)^8(p_0(8)\,(1-q)^8 + p_1(8)\,p\,(1-q)^7) \\ & + p^*_1(7)\,p(1-q)^7(p_0(8)\,(1-q)^8 + p_1(8)\,p\,(1-q)^7) \approx 56q^2 \end{aligned} \qquad (33)$$

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

$$p_{2\text{-}dirty,\,2} = (1-d)(1-q)^7(p_0(8)\,(1-q)^8 + p_1(8)\,p\,(1-q)^7)p_d^8 \qquad (34)$$

$$\approx 64q^2$$

Case (2) - two *dirty bytes* are adjacent:

This is the case that two *dirty bytes* are adjacent in a *0-NI2-L* codeword. A probability should be calculated at first. In order to be distinct, the variable $p_0(n)$ is redefined as $p'_0(n)$ here. Obviously $p'_0(n)$ shares the same recursive formula with $p_0(n)$ but they have different first terms. The first term, $p'_0(2)$ is $(1-p)\,p$. After iterative process, the general term is:

$$p'_0(n) = (1-r)\,(1-q+r) - (1-r)\,(1-q)^{n-1} \qquad (35)$$

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

$$p'_0(8) = 6q - 21q^2 + r - 6q\,r \qquad (36)$$

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 $p_{2\text{-}dirty,\,3}$) is shown below and detailed derivation process is shown in Appendix (see Proof 8):

$$p_{2\text{-}dirty,\,3} \approx 55q^2 + q\,r \qquad (37)$$

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

$$\begin{aligned} p_{2\text{-}dirty,\,4} = {} & (1-d)\,(1-q)^8(p_0(8)\,(p_0(8)(1-q)^8 + p_1(8)\,p(1-q)^7) \\ & + p_1(8)\,(p'_0(8)\,(1-q)^8 + p_1(8)\,p(1-q)^7)) \approx 63\mathrm{q}^2 \end{aligned} \qquad (38)$$

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

$$p^*_k(n) = 1 - (1-q)^n + r - q \qquad (39)$$

When *n* equals eight, the value is:

$$p^*_k(8) = 7q - 28q^2 + r \qquad (40)$$

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 $p_{2\text{-}dirty,\,5}$ and it is shown as follows.

$$\begin{aligned} P_{2\text{-}dirty,\,5} = {} & (1-d)(1-q)^{15}(p_0(8)\,p_d^8 + p_1(8)\,p_d^{8^*}) \\ & \approx 63q^2 + q\,r \end{aligned} \qquad (41)$$

Finally, the value of p_5 is:

$$\begin{aligned} p_5 = {} & p_{2\text{-}dirty,\,1} + p_{2\text{-}dirty,\,2} + p_{2\text{-}dirty,\,3} + p_{2\text{-}dirty,\,4} + p_{2\text{-}dirty,\,5} \\ & \approx 307q^2 + 2q\,r \end{aligned} \qquad (42)$$

In terms of p_6, there exist at least 14 continuous *set bits*. This is a term of high degree, which can be divided by the expression r^{14}. 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.

$$p_8 = d\,q^{30} + C^1_{29}(1-d)p\,q\,(1-q)^{28} + (1-d)\,q\,(1-q)^{29} \qquad (43)$$

$\approx q - 28q^2$

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

Denotation	Probability
q_1	$1-93q+4281q^2-3q\,r$
q_2	$31q - 2745q^2- q\,r$
q_3	$961q^2$
q_4	$961q^2$
q_5	0
q_6	$31q - 2388q^2+q\,r$
q_7	$357q^2+2q\,r$
q_8	$31q -2745q^2- q\,r$
q_9	$301q^2$
q_{10}	$301q^2$
q_{11}	$301q^2$
q_{12}	$415q^2- 4q\,r$

TABLE VI. Probabilities in CONCISE

Denotation	Probability
q_1^c	$1- 93q + 4281q^2-3q\,r$
q_2^c	0
q_3^c	0
q_4^c	0
q_5^c	0
q_6^c	0
q_7^c	$31q - 2388q^2+q\,r$
q_8^c	$435q^2+30q\,r$
q_9^c	$31q -2823q^2-29q\,r$
q_{10}^c	0
q_{11}^c	0
q_{12}^c	0
q_{13}^c	$31q -1862q^2-29qr$

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.

$$\overline{L_{COMBAT}} \approx 1 + 31q + 362q^2 - q\,r \tag{44}$$
$$\overline{L_{SECOMPAX}} \approx 1 + 62q - 218q^2 + 6q\,r \tag{45}$$
$$\overline{L_{CONCISE}} \approx 1 + 62q + 899q^2 + 62q\,r \tag{46}$$

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.

3) Decompression time complexity

Similar to subsection A, decompression time can be also estimated and values in TABLE IV still hold. The values of $\overline{T_{COMBAT}}$, $\overline{T_{SECOMPAX}}$ and $\overline{T_{CONCISE}}$ can be recalculated and their results are also shown below. Their derivation processes are similar to those in subsection A.

$$\overline{T_{COMBAT}} \approx t_1(1+124q)+t_2(1+252q) \tag{47}$$
$$\overline{T_{SECOMPAX}} \approx t_1(1+124q)+t_2(1+188q) \tag{48}$$
$$\overline{T_{CONCISE}} \approx t_1(1+124q)+t_2(1+126q) \tag{49}$$

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

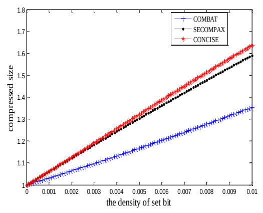

Fig. 8. Simulation results with uniformly distributed bitmaps

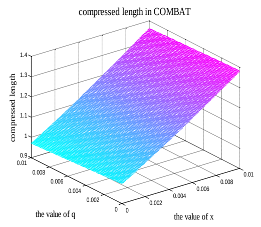

Fig. 9. Compressed size in COMBAT

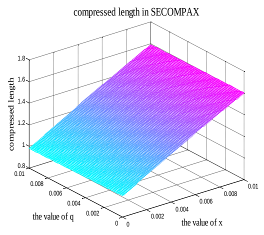

Fig. 10. Compressed size in SECOMPAX

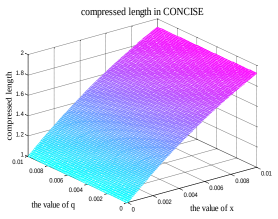

Fig. 11. Compressed size in CONCISE

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.

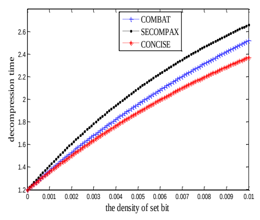

Fig. 12. Simulation results with uniformly distributed bitmaps

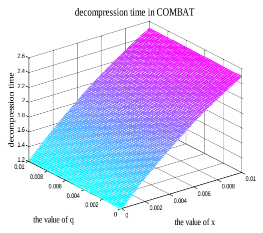

Fig. 13. Decompression time evaluation in COMBAT

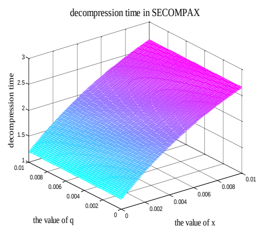

Fig. 14. Decompression time evaluation in SECOMPAX

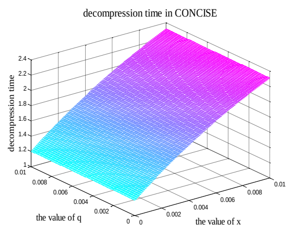

Fig. 15. Decompression time evaluation in CONCISE

B. *Evaluation results on temporal performance*

When evaluating the temporal performance, since final results contains up to four variables, i.e. q, r, t_1, t_2, some variables should be set constant values to facilitate our evaluation in advance. 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, $0.2t$. 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, 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

[1] I. Spiegler and R. Maayan, Storage and retrieval considerations of binary data bases, Information Processing and Management, vol. 21, no. 3, pp. 233-254, 1985.

[2] P. E. O'Neil, Model 204 architecture and performance, in High Performance Transaction Systems. Springer Berlin Heidelberg, 1989, pp.39-59

[3] P. Cheng, "bitmap index techniques and its research advancement," Science and technologies information, Vol. 026, pp.134-135, 2010.

[4] J. Li, Research in bitmap index in data warehouse, (in Chinese), PhD diss, Shandong University, 2007.

[5] Z. Huang, W. Lv, and J. Huang, "Improved BLAST algorithm based on bitmap indexes and B+ tree," Computer Engineering and Applications, 49(11), pp.118-120, 2013.

[6] B. Yang, Y. Qi, Y. Xue, and J. Li, "Bitmap data structure: Towards high-performance network algorithms designing," Computer Engineering and Applications, 45(15), 2009.

[7] H. Garcia-Molina, J. D. Ullman, and J. Widom, Database System implementation, Second Edition, Prentice Hall, 2009.

[8] G. Antoshenkov, "Byte-aligned bitmap compression," Data Compression Conference, 1995.

[9] K. Wu, Ekow J. Otoo , and A. Shoshani, "Compressing bitmap indexes for faster search operations." In Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on, pp. 99-108. IEEE, 2002.

[10] K. Wu, Ekow J. Otoo , and A. Shoshani, "Optimizing bitmap indexes with efficient compression," in ACM Transactions on Database Systems (TODS), 31(1), 2006, pp.1-38.

[11] C. Guadalupe, M. Gibas, and H. Ferhatosmanoglu, "Update conscious bitmap indexes," 19th IEEE International Conference on Scientific and Statistical Database Management SSBDM'07, pp. 15-15, 2007.

[12] M. Stabno, and R. Wrembel. "RLH: Bitmap compression technique based on run-length and Huffman encoding," Information Systems 34, no. 4, 2009, pp.400-414.

[13] F. Corrales, D. Chiu, and J. Sawin, "Variable Length Compression for Bitmap Indexes," in DEXA'11, Springer-Verlag, pp.381-395, 2011.

[14] F. Deli`ege and T. B. Pedersen, "Position list word aligned hybrid: optimizing space and performance for compressed bitmaps," In Proceeding of the 13th International Conference on Extending Database Technology, 2010.

[15] D. Lemire, O.Kaser, and K. Aouiche, "Sorting improves word-aligned bitmap indexes," Data & Knowledge Engineering, 69(1), pp.3-28, 2010.

[16] S. J. van Schaik and O. de Moor, "A memory efficient reachability data structure through bit vector compression," In Proceedings of the

2011 ACM SIGMOD International Conference on Management of data, pp. 913-924. ACM, 2011.

[17] F. Fusco, M. P. Stoecklin, and M.Vlachos, "Net-fli: on-the-fly compression, archiving and indexing of streaming network traffic," Proceedings of the VLDB Endowment, 3(1-2), pp.1382-1393, 2010.

[18] W.Andrzejewski, and R.Wrembel, "GPU-WAH: Applying GPUs to compressing bitmap indexes with word aligned hybrid," In Database and Expert Systems Applications, Springer Berlin Heidelberg, January, pp. 315-329, 2010.

[19] F. Fusco, M. Vlachos, X. Dimitropoulos, and L. Deri, "Indexing million of packets per second using GPUs," In Proceedings of the 2013 conference on Internet measurement conference, pp.327-332. ACM, 2013.

[20] W. Andrzejewski, and R. Wrembel, "GPU-PLWAH: GPU-based implementation of the PLWAH algorithm for compressing bitmaps," Control & Cybernetics, 40(3), pp. 627-650, 2011.

[21] Y. Wen, Z. Chen, G. Ma, J. Cao, W. Zheng, G. Peng, and W. L. Huang, "SECOMPAX: A bitmap index compression algorithm," In 23rd International Conference on Computer Communication and Networks (ICCCN), IEEE, pp. 1-7, 2014.

[22] J. Chang, Z. Chen, W. Zheng, Y. Wen, J. Cao, and W. L. Huang, "PLWAH+: a bitmap index compressing scheme based on PLWAH," In Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems, ACM, pp. 257-258, 2014.

[23] A. Schmidt, D. Kimmig, and M. Beine, "DFWAH: A Proposal of a New Compression Scheme of Medium-Sparse Bitmaps," in the Third International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2011), pp. 192-195.

[24] S. Chambi, D. Lemire, O. Kaser, and R. Godin, "Better bitmap performance with Roaring bitmaps," arXiv preprint arXiv:1402.6407 (2014).

[25] G. Ma, Z. Guo, X. Li, Z. Chen, J. Cao, Y. Jiang, and X. Guo. "BreadZip: a combination of network traffic data and bitmap index encoding algorithm." In Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on, pp. 3235-3240. IEEE, 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.

[27] A. Colantonio, and R. Di Pietro, "Concise: Compressed 'n' composable integer set," In Information Processing Letters, 110(16), 2010, pp.644-650.

[28] Yinjun Wu, Zhen Chen, Yuhao Wen, Junwei Cao, "COMBAT: a new bitmap index compression algorithm for Big Data", in submission.

[29] Z. Chen, Y. Wen, J. Cao, W. Zheng, J. Chang, Y. Wu, G. Ma, M. Hakmaoui, 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.

$q_1 = p_1p_1p_1 = (1 - 31d + 465d^2)^3 \approx 1 - 93d + 4278d^2$ $q_2 = q_3 = p_1p_1p_2 \approx 0$ $q_4 = q_5 = p_1p_2p_2 \approx 0$ $q_6 = p_2p_2p_2 \approx 0$

$q_7 = (p_1+p_2)(p_3+p_4)(p_1+p_2) \approx 31d - 2747d^2$ $q_8 = (p_3+p_4)(p_1+p_2)(p_3+p_4) \approx 961d^2$ $q_9 = (p_3+p_4)(p_1+p_2)p_7 \approx 961d^2$

$q_{10} = (p_3+p_4)(p_1+p_2)(p_7 - p_3 - p_4) \approx 961d^2$ $q_{11} = p_7p_1p_1 \approx 31d - 2387d^2$ $q_{12} = (p_7 - p_3 - p_4)p_1p_1 \approx 360d^2$

$q_{13} = (p_1+p_2)(p_3+p_4)(p_1+p_2) \approx 31d - 2747d^2$ $q_{14} = (p_1+p_2)(p_5+p_6)p_7 = 360d^2(1 - 31d + 465d^2)(31d - 465d^2) \approx 0$

$q_{15} = (p_5+p_6)(p_1+p_2)(p_1+p_2) \approx 360d^2$ $q_{16} = (p_5+p_6)(p_1+p_2) = 360d^2(1 - 31d + 465d^2) \approx 360d^2$ $q_{17} = 1 - \sum_{i=1}^{18} q_i \approx 165d^2$

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

$q_1^c = p_1p_1p_1 \approx (1 - 31d + 465d^2)^3 \approx 1 - 93d + 4278d^2$ $q_2^c = p_2p_2p_2 = d^{93} \approx 0$ $q_3^c = q_4^c = p_1p_1p_2 = d^{31}(1 - d)^{62} \approx 0$

$q_5^c = q_6^c = p_1p_2p_2 = d^{62}(1 - d)^{31} \approx 0$ $q_7^c = (p_1+p_2)(p_1+p_2)p_7 \approx 31d - 2387d^2$ $q_8^c = (p_1+p_2)(p_1+p_2)(p_7 - p_8) \approx 465d^2$

$q_9^c = p_1p_1p_8 \approx 31d - 2852d^2$ $q_{10}^c = q_{11}^c = p_8p_2p_1 = (31d - 930d^2)d^{31}(1 - d)^{31} \approx 0$ $q_{12}^c = p_8p_2p_2 = (31d - 930d^2)d^{62} \approx 0$

$q_{13}^c = p_8(p_2 + p_1) \approx (31d - 930d^2)(1 - 31d + 465d^2) \approx 31d - 1891d^2$

Proof 3. This is to prove the equation of (10) and (11)

$\overline{L_{SECOMPAX}} = q_1 + 2q_2 + 2q_3 + 2q_4 + 2q_5 + q_6 + q_7 + q_8 + 2q_{11} + 2q_{12} + 2q_{13} + 3(1 - q_1 - q_2 - q_3 - q_4 - q_5 - q_6 - q_7 - q_8 - q_{11} - q_{12} - q_{13}) \approx 1 + 62d - 210d^2$

$\overline{L_{CONCISE}} = q_1^c + q_2^c + 2q_3^c + 2q_4^c + 2q_5^c + 2q_6^c + 2q_7^c + 2q_8^c + q_9^c + 2q_{10}^c + 2q_{11}^c + 2q_{12}^c + 2q_{13}^c + 3q_{14}^c \approx 1 + 62d + 1922d^2$

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

$$\begin{aligned}\overline{T_{COMBAT}} = {} & q_1(t_1 + t_2) + 2q_2(t_1 + t_2) + 2q_3(t_1 + t_2) + 2q_4(t_1 + t_2) + 2q_5(t_1 + t_2) + q_6(t_1 + t_2) + q_7(3t_1 + 5t_2) + q_8(3t_1 + 6t_2) \\ & + q_9(2t_1 + 4t_2 + t_1 + t_2) + q_{10}(t_1 + t_2 + 2t_1 + 4t_2) + q_{11}(t_1 + t_2 + t_1 + t_2) + q_{12}(t_1 + t_2 + t_1 + t_2) + q_{13}(2t_1 + 4t_2) + q_{14}(3t_1 + 6t_2 + t_1 + t_2) \\ & + q_{15}(3t_1 + 6t_2) + q_{16}(t_1 + t_2 + 3t_1 + 6t_2) + 3q_{17}(t_1 + t_2) \approx (1 + 124d)t_1 + (1 + 248d)t_2\end{aligned}$$

$$\begin{aligned}\overline{T_{SECOMPAX}} = {} & q_1(t_1 + t_2) + 2q_2(t_1 + t_2) + 2q_3(t_1 + t_2) + 2q_4(t_1 + t_2) + 2q_5(t_1 + t_2) + q_6(t_1 + t_2) + q_7(3t_1 + 5t_2) + q_8(3t_1 + 6t_2) \\ & + q_{11}(t_1 + t_2 + t_1 + t_2) + q_{12}(t_1 + t_2 + t_1 + t_2) + q_{13}(2t_1 + 2t_2) \\ & + 3(1 - q_1 - q_2 - q_3 - q_4 - q_5 - q_6 - q_7 - q_8 - q_9 - q_{10} - q_{13} - q_{14} - q_{15})(t_1 + t_2) \approx (1 + 124d)t_1 + (1 + 248d)t_2\end{aligned}$$

$$\begin{aligned}\overline{T_{CONCISE}} = {} & q_1^c(t_1 + t_2) + q_2^c(t_1 + t_2) + 2q_3^c(t_1 + t_2) + 2q_4^c(t_1 + t_2) + 2q_5^c(t_1 + t_2) + 2q_6^c(t_1 + t_2) + 2q_7^c(t_1 + t_2) + 2q_8^c(t_1 + t_2) \\ & + q_9^c(2t_1 + 3t_2) + q_{10}^c(3t_1 + 3t_2) + q_{11}^c(3t_1 + 3t_2) + q_{12}^c(2t_1 + 2t_2) + q_{13}^c(3t_1 + 4t_2) \\ & + (1 - q_1^c - q_2^c - q_3^c - q_4^c - q_5^c - q_6^c - q_7^c - q_8^c - q_9^c - q_{10}^c - q_{11}^c - q_{12}^c - q_{13}^c)(3t_1 + 3t_2) \approx (1 + 124d)t_1 + (1 + 186d)t_2\end{aligned}$$

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 bit*s. Since they all follows 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: $p_1 = \frac{1 - r}{1 - r + q}(1 - q)^{30}$. After using Taylor expansion and omitting terms of high degree, the final approximate result is $p_1 \approx 1 - 31q + 466q^2 - q\,r$.

In a *1-fill*, the first bit is *set bit* and this bit is followed by 30 continuous *set bit*s. The probability is $d\,(1-p)^{30}$. After replacing d with variable q and r and replacing p with r, the result is $\frac{1-r}{1-r+q}\,r^{30}$. Obviously, r^{30} is a term of high degree and $\frac{1-r}{1-r+q}$ is a finite value. So when r approaches zero, the result can be seen as $p_2 \to 0$

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

After omitting items of higher degree, the result is $\begin{pmatrix} 1-q+q^2-q\,r & 1-q+q^2-q\,r \\ q-q^2+q\,r & q-q^2+q\,r \end{pmatrix}$, 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, $A^{k+1} = A^k A = \begin{pmatrix} 1-q+q^2-q\,r & 1-q+q^2-q\,r \\ q-q^2+q\,r & q-q^2+q\,r \end{pmatrix} \begin{pmatrix} 1-q & 1-r \\ q & r \end{pmatrix} = \begin{pmatrix} 1-q+q^2-q\,r & 1-q+q^2-q\,r \\ q-q^2+q\,r & q-q^2+q\,r \end{pmatrix}$

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 bit*s. 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^{dirty} is $C_2^1(1-d)\,(1-q)^{21}(p_0(8)\,(1-q)+p_1(8)\,p)$. According to the value of $p_0(8)$ and $p_1(8)$, the simplified result is $p_1^{dirty} \approx 16q-422q^2-2q\,r$

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 $p^*{}_0(7)\,(p_0(8)\,(1-q)^{16}+p_1(8)\,p\,(1-q)^{15})$. 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 $p_3^{2\text{-}dirty}$ is the sum of these two parts of probability and it is shown below.

$p_3^{2\text{-}dirty} = p^*{}_0(7)\,(p_0(8)\,(1-q)^{16}+p_1(8)\,p\,(1-q)^{15}) + p^*{}_1(7)\,(p'{}_0(8)\,(1-q)^{16}+p'{}_1(8)\,p\,(1-q)^{15})$

Then after specific value is assigned and simplification process is done, the value of $p_3^{2\text{-}dirty}$ is: $p_3^{2\text{-}dirty} \approx 55q^2+q\,r$