**Big Data Compression Algorithms: Enhanced J-Bit Encoding Approach**

Mahtab Alam

Dept. of Computer Science & Engineering,

Noida International University,

alam12mahtab@gmail.com

***Abstract- Data compression is useful approach for storing the large volume of structured data in every application of information technology. It helps to increase and optimizes operational efficiencies, enable cost reductions, and reduce risks for the business operations. The size of databases is increases time to time and it needs to compress for storage and retrieval. The objective of this paper is to reduce the size of data by compression techniques using Enhance J-bit algorithms. In this paper we are proposing a new compression algorithm based on J-bit encoding techniques that will help to compress data and reduce the storage capacity. This algorithm is based on optimizing and splitting the input data into two parts 1 for “true” and 0 for “false”. Data compression algorithm compresses the data and reduces the size of database. Data warehouse has large volume of storage capacity that can be used by J-bit encoding for largest data bits. It is hard task to reduce the size of database and minimize the retrieval time. This proposed algorithm of data compression is named as Enhanced J-Bit Encoding (EJBE). The algorithm will controls each bit of data of a file and optimizes the size without losing any data subsequently using decoding which is known as lossless data compression. A lossless technique means that the restored data file is equal to the original form.***

**Keywords:** Burrow WheelAlgorithm**,** Data Compression, Lexicography Sorting, Data Encoding.

1. **INTRODUCTION**

The database plays a vital role of data mining for organization to provide the better result. Large volume or complex data always creates problem. Therefore, data compression is desperately required by all the organization for smooth, reliable and timely retrieval of the information. There is a number of data compression algorithm developed by researchers to compress the data and image files. A little work has been done in this research paper to reduce the size of the data and minimize the operational time of fetching the records from the data warehouse [1]. The data are stored in the buffer area of the computer memory. A buffer is a sequential section of computer memory that holds more than one instances of the same data type. A buffer overflow condition occurs when a program attempts to read or write outside the bounds of a block of allocated memory or when a program attempts to put data in a memory area past a buffer [2][3]. To avoid the buffer overrun problem industries desperately require data compression to store the data within stipulated space of buffer area.

This paper organized as section 1 covers the general introduction of data compression; section 2 covers the algorithm which are associated with our proposed algorithm related to data compression by earlier researchers, in section 3 we are proposing the multiple algorithm based on J-Bit, Burrow wheel, Zero Run-Length Coding and Huffman algorithm and introduce a new technique by combining all these algorithms, followed by the combination of all earlier proposed algorithm in Section 4. In section 5 result and conclusion are discussed. Finally, in section 6 conclusion part of this research work is described.

Data compression is the universal term for the numerous processes and programs developed to address this problem. A *compression program* is used to convert data from an easy-to-use presentation to one enhanced for compactness. Likewise, an *uncompressed program* returns the information to its original form. This technique is used on very large data contents in logical and physical database. In physical, database the data are stored in bits forms as input stream whereas on logical database, the particular data are stored in the forms of data contents in the output stream and they swap mutual data contents with a little bit of programming code. Logical method is compressing the data in database.

1. **ASSOCIATED ALGORITHMS**
2. **J-Bit Encoding**: J-bit encoding compress the file by reducing the bit size without losing the data. At the time of decompression the original data is obtain. The main objective of J-bit encoding is to combine the other compression algorithm and mitigate the compression ratio. In this paper we enhance the J-bit compression algorithm; the proposed algorithm is discussed in section III [4].
3. **Burrow-Wheel Forward Transform**: This algorithm is comprises in several stages and the input and output of the data are taken arbitrarily, but the size of the data is mentioned in the input stage. The same size of data is obtained after decompression process; it means it works on lossless data compression techniques. It consist three phases during the entire compression process namely, BWT (Burrow Wheel Technique), MTF (Move-To-Front) and EC (Entropy Encoding). In this proposed algorithm we reduced the complexities of the compression algorithm by removing the Entropy Encoding phase.
4. **Zero Run-Length Coding:** The Zero Run-Length Coding proposed the algorithm to compress the data by arranging the most attainable and least attainable data. There are so many techniques to decrease dimension in data warehouse by eliminating redundancy methods as Zero Run length Coding [5][6].The entire data is divided into two parts in such a way that their numbers of step are used. It is coded sequences of zeros to shorter strings. Now we explain the technique of the Zero Run-Length Coding step by step 233300000/9 (output of the example from the Move-To-Front Transform) as example input.
5. **Burrows-Wheeler Backward Transforms:** This technique is just reverse of Burrow Wheel Forward Transformation technique. In this technique, the decompression of the complex data is performed into simpler form. The algorithm reverses the process in different order of data compression. The input of the data contents are used in L-column i.e. left hand side and it covers the creative data input. The entire algorithm is discussed in section IV [7].
6. **Huffman Coding**: This is used for bits information and reduced the size of database by bits format. It has used fixed-lengthen coding like ASCII is convenient because the boundaries between characters are easily determined and the pattern used for each character is completely fixed (i.e. 'A' is always exactly 65) [8].
7. **Shannon Fano Encoding:** Shannon–Fano algorithm was developed by Claude Elwood Shannon and Robert Fano. It is a method to find the probabilities of prefix code which are based on set of symbols. According to this coding, the most probable symbols are getting privilege over least probable symbols, and then arranged into two sets whose total probabilities are as close as possible to being equal. All symbols then have the first digits of their codes assigned; symbols in the first set receive "0" and symbols in the second set receive "1". The Algorithm is proposed as under:
8. **PROPOSED ALGORITHM**

The proposed algorithm works on bits of data that should be reduced in size and enhance the input mechanism handle by J-bit encoding (JBE) based on other algorithm. This algorithm is to divide the input data into two parts where the first part will cover unique nonzero byte and the second part will cover bit value illumination location of nonzero and zero bytes. Both parts then can be compress distinctly with other data compression algorithm to attain maximum compression ratio. The compression mechanism can be defined as given step:

Step1-Input per byte that can be any kind of database.

Step 2-It control input byte by way of nonzero or zero byte.

Step 3- Output of nonzero byte into data I and write bit ‘1’ into temporary byte data, or only write bit ‘0’ into temporary byte data for zero input byte.

Step 4-Replication step 1-3 until temporary byte data filled with 16 bits of data.

Step 5- If temporary byte data filled with 16 bits then write the byte value of temporary byte data into data II.

Step 6- Clear temporary byte data.

Step 7-Replication step 1-6 until end of file is reached.

Step 8- Write combined output data

a) Write original input length.

b) Write data I.

c) Write data II.

Step 9-Data I and data II can be compress distinctly before mutual monitored by additional compression algorithm,

**Original**

16385=0100000000000001

16385=0100000000000001

0= 0000000000000000

0= 0000000000000000

16385=0100000000000001

16385=0100000000000001

0= 0000000000000000

0= 0000000000000000

0= 0000000000000000

0= 0000000000000000

0= 0000000000000000

0= 0000000000000000

16385=0100000000000001

16385=0100000000000001

16386=**100000000000010**

16386=**100000000000010**

**Data I**

16385=0100000000000001

16385=0100000000000001

16385=0100000000000001

16385=0100000000000001

16385=0100000000000001

16385=0100000000000001

16386=**100000000000010**

16386=**100000000000010**

**Temporary Byte**

1 0000000000000001

1 0000000000000001

0 0000000000000000

0 0000000000000000

1 0000000000000001

1 0000000000000001

0 0000000000000000

0 0000000000000000

0 0000000000000000

0 0000000000000000

0 0000000000000000

0 0000000000000000

1 0000000000000001

1 0000000000000001

1 0000000000000001

1 0000000000000001

**Data Byte II**

1100110000001111 = 52239

Figure 1: J- Bit Encoding Algorithm Mechanism

Figure 1 displays step-by-step compression mechanism pictorially for Enhanced J-bit Encoding (EJBE). The original input data appears into the first of the output will be used by method of information for data I and data II size. The data compression mechanism can be defined as following steps:

Step 1- Accept the original input database size.

Step - It was compressed distinctly, decompress data I and data II.

Step 3- Accept data II per bit.

Step 4- It controls whether input bit is '0' or '1'.

Step 5- Display to output, if input bit is '1' then input and display data I to output, if input bit is '0' then display zero byte to output.

Step 6- Replication steps 2-5 until original input data is reached.

1. **ALGORITHM COMBINATION COMPARISON**

The data compression algorithm has used five combinations and find out which one combination is the best compression ratio. The combinations are:

1.ZRL+BWFT

2. BWFT+MTF+EC

3. BWFT+MTF+EC+SFE

4. ZRL+BWBT+EC+SFE+JBE

5. ZRL+BWT+MTF+JBE**+EJBE**

**1. Burrows-Wheeler TransformAlgorithm**

This algorithm will used for data compression. The algorithm can be refunded to their unique data with the decompression. It is to kind the characters of the input with the result that identical characters are nearby composed.

Input- BWT+MTF+EC-output

Process steps:

Step1- Order the length of the input n times among them, thus replace each row one character to the right compared to the previous row

Step2- Sort the rows lexicographical

The output of this phase is the L-column (Last column and First column) and the index value of the sorted matrix that contains the original input.

Now we describe the technique step by step with MISSISSIPI as example input.

**Step-1**

INPUT STREAM

0 M I S S I S S I P I

1 I M I S S I S S I P

2 P I M I S S I S S I

3 I P I M I S S I S S

4 S I P I M I S S I S

5 S S I P I M I S S I

6 I S S I P I M I S S

7 S I S S I P I M I S

8 S S I S S I P I M I

9 I S S I S S I P I M

**Step-2**

**LEXICOGRAPHY SORTING:-**

F-COL L-COL

0 I M I S I S P S I M

1 I P I S I S I S I S

2 I S I S I S I S I S

3 I S I S I S I S I S

4 M S S M P M I M P S

5 P S S P M I M P S P

6 S I S I S I S I S I

7 S I S I S I S I S I

8 S I M I S I S I S I

9 S I P I S P S I M I

**OUTPUT:- MSSSSPIIII/9**

The above given strings are arranged in such a way that all the identical characters are intact together, and this string is consider as the input for the next stages [8].

**2. Move-To-Front Transform Algorithm**

It shows the global index value. Additionally we add input data in global list Y.

Process steps:

Step1-The leading this character follows the input data separately since the index value to global list Y

Step2-Then If we allocation the secure character of the preceding step in the global list Y on index position 0 and allocation all characters one position to the right which is located in the global list Y Formerly the ancient location of the secure character.

Step3-Replication step 1 and 2 consecutively for the other characters of the input and use for all replications the adapted global list from the previous replication The output of this phase contains of all saved index positions and the index value of the sorted matrix from the Burrows-Wheeler Transform which covers the original input. This index value won't procedure in the Move-To-Front Transform [11].

Now we describe the process step by step with MSSSSPIIII/9 (output of the example from the Burrows-Wheeler Transform) as example input. We use Y = [I, M, P, S] as global list and don't a global list of all characters of the ASCII-Code, because the example is better to present and easier to understand with the smaller global list.

**STEP 1:**

INPUT: MSSSSPIIII/9

Y= I M P S

**STEP 2: SAVE INDEX POSTION 1**

Y= I M**P**S

Y= MIPS

**STEP 3: SAVE INDEX POSTION 3**

Y= M IPS

Y=SM I P

**STEP 4: SAVE INDEX POSTION 3**

Y=S M I P

Y=PS M I

**STEP 5: SAVE INDEX POSTION 3**

Y=P S M I

Y=IP S M

**STEP 6: SAVE INDEX POSTION 0**

Y=I P S M

Y=I P S M

**STEP7: SAVE INDEX POSTION 0**

Y=I P S M

Y=I P S M

**STEP 8: SAVE INDEX POSTION 0**

Y=I P S M

Y=I P S M

**STEP 9: SAVE INDEX POSTION 0**

Y=I P S M

Y=I P S M

**STEP 10: SAVE INDEX POSTION 0**

Y=I P S M

Y=I P S M

**STEP 11: SAVE INDEX POSTION 0**

Y= I P S M

Y= I P S M

Output: 1333000000/9

**3. Zero Run-Length Coding**

The Zero Run-Length Coding coded sequences of zeros to shorter string. We describe the method of the Zero Run-Length Coding step by step 233300000/9 (output of the example from the Move-To-Front Transform) as example input.

1333000000/9

REPLACING 1 BY 2 AND 3 BY 4 the output string become 2444000000

OUTPUT = 244411 (Six zero will be coded as ‘11’).

Process Steps:

Step1- Increase all characters of the input which are greater as 0 by 1

Input: 1333000000 = 2444000000

Step2- Coding the sequences of zeros with string combinations of 0 and 1

We use at this point an example coding table.

Number of zeros coding string

1 0

2 1

3 00

4 01

5 10

6 11

**2444000000 = Output: 244411**

**4. Burrows-Wheeler Backward Transforms**

Suffix sorting is also an important problem in data compression, especially for compression schemes that are based on the Burrows-Wheeler Transform [12]

This transform process is reverse back algorithm. It identify input data left hand side value to be replace in L-column and the index value is sorted from F-column matrix which covers the creative input data. Following practice is:

Process steps:

Step 1-We are sorted data from the L-column input alphabetical to F-column input data.

Step 2- After that we save the character, then located in the F-column on the input index value.

Step 3- If we search the character in matrix index value i, whereby, the index value i is used the character in L-column which is similar to the protected character of the preceding step and this character has the same number of equal characters with lower index values in L-column such as the protected character of the preceding step in the F-column

Step 4- We can protect the character which is located in right hand side F-column on the index value i.

Step 5- Replication step 3 and 4 and break, when index value i is identical to the input data index. The output is contained of all protected characters.

**STEP-1**

**Save order Index F- column L-column**

* 0 I M
* 1 I S
* 2 I S
* 3 I S
* 4 M S
* 5 P P
* 6 S I
* 7 S I
* 8 S I
* 9 S I

**STEP-2**

**Save order Index F- column L-column**

* 0 I M
* 1 I S
* 2 I S
* 3 I S
* 4 M S
* 5 P P
* 6 S I
* 7 S I
* 8 S I
* 9 S I

STEP-3

**Save order Index F- column L-column**

* 0 I M
* 1 I S
* 2 I S
* 3 I S

1. 4 M S

* 5 P P
* 6 S I
* 7 S I
* 8 S I
* 9 S I

STEP-4

**Save order Index F- column L-column**

* 0 I M
* 1 I S
* 2 I S

1. 3 I S

1 4 M S

* 5 P P
* 6 S I
* 7 S I
* 8 S I
* 9 S I

STEP-5

**Save order Index F- column L-column**

1. 0 I M
2. 1 I S
3. 2 I S
4. 3 I S
5. 4 M S

9 5 P P

7 6 S I

6 7 S I

4 8 S I

3 9 S I

Finally, we replicate the processsteps, to find the other characters of the original input. We recognize that the rows are in lexicographical order; therefore the order of identical characters from the F-column is determined by the following characters of the rows. And the order of the identical characters from the L-column is determined by the characters of the F-column and the following characters of the rows. Then, we can accept that the order of the identical characters in both columns is same, because the same characters in both columns decided about the order of the identical characters [10].

**5. HUFFMAN CODINGALGORITHM**

The ASCII set has 256 characters with identical occurrence. In this table has used 90 and unique characters. This algorithm precedes assistance of the difference between occurrences and practices little bit space for the frequently occurring characters at the amount of consuming to use more space for each of the more infrequent characters. For instance as *variable-length* encoding—it is involved 2 or 3 bits and other characters are involved 7, 10, or 12 bits. The reserves character from not consuming to practice a full 8 bits for the most corporate characters makes up for consuming to practice more than 8 bits for the infrequent characters and the overall result is that the file virtually always desires little bit space.

**ASCII Encoding Statement**

In this instance, we are successful to procedure through this input data is encoding the certain string **"MISSISSIPPI STATE”** By the usual ASCII encoding, this 17 character string needs 13\*8 = 104 total bits. Following ASCII table shows below:

**Character ASCII Value Binary Value**

M 77 01001101

I 73 01001001

S 83 01010011

P 80 01010000

T 84 01010100

A 65 01000001

E 69 01000101

Space 32 00100000

Instance **"MISSISSIPPI STATE"** string would be encoded in ASCII standard through**77738383738383 738080733283 84 65 84 69**. It is not basically strong by individuals; it would be reproduced as the subsequent stream of bits (each byte is divided into boxes):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 01001101 | 01001001 | 01010011 | 01010011 | 01001001 | 01010011 | 01010011 | 01001001 |
| 01010000 | 01010000 | 01001001 | 00100000 | 01010011 | 01010100 | 01000001 | 01010100 |
| 01000101 |

**A data compressed encoding**

The primary it can communicate roughly ASCII standards encoding is that with 8 bits per character can be exceptionally extensive data. Even if it permits for the option of indicating 256 dissimilar characters, we only have seven dissimilar characters in the expression. We are annoying to encode, and therefore could distinguish between these designs with little bits. We might be design unusual coding table objective for this expression with 3 bits for each character. Producing as an encoding is trivial: we produce a list of the exclusive characters, and then go complete and allocate each a dissimilar encoded number from 0 to N-1. For instance, as possible 3-bit encoding (i.e.7 possible arrangements);

**Character Digit Bit Design**

M 0 000

I 1 001

S 2 010

P 3 011

T 4 100

A 5 101

E 6 110

Space 7 111

Consuming this table, **"MISSISSIPPI STATE"** is encoded as **0 1 2 2 1 2 2 1 3 3 1 7 2 4 5 4 6**

Denoted as binary numbers

000 001 010 010 001 010 010 001 011 011 001 111 010 100 101 100 110

Using three bits per character, the encoded string requires 51 bits in its place of the unique 136 bits, compressing to 62.5% to its original size.

Yet, this binary coded illustration to decode first would requirement to recognize the distinct mapping used, meanwhile consuming 000for ‘**M’**is not regular preparation and in statistic, in this pattern, each compressed string practices its own special-purpose representing that is not certainly corresponding any other bits. Different types of header and sorted supplementary file would have to be devoted or involved by the encoded illustration that delivers the mapping data. That header would take up some supplementary space that would cut into our compression reserves. For a huge sufficient file, though, the reserves from trimming down per character cost would prospective offset the outflow of the supplementary table space.

**A variable-length encoding**

Do we fall the requirement that all characters take up the same quantity of bits? Through consuming little bits to encode characters like **'p'**, **'h'**, and space that ensue regularly and supplementary to encode characters like **'y'** and **'o'** that ensue fewer regularly, we could be competent to compress data smooth advance. We would future display in what way we produced the table below, but for at the present just take our word for it that is denotes an execute Huffman encoding for the string **"MISSISSIPPI STATE"**:

**Character Weightage Bit Pattern**

M 1 000

I 4 10

S 5 01

P 2 100

T 2 001

A 1 111

E 1 110

Space 1 101

Each character has an exclusive bit design encoding, then not all characters practice the identical quantity of bits. The string **"MISSISSIPPI STATE"** encoded consuming the above variable-length code table is:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 000 | 10 | 01 | 01 | 10 | 01 | 01 | 10 | 100 | 100 |
| 10 | 101 | 01 | 001 | 111 | 001 | 110 |

Total of 42 bits are required in encoded expression, we are cutting an insufficient additional bits from the fixed-length description. What is delicate around a variable-length code is that we not at all may certainly decide the limitations between characters in the encoded stream of bits when decoding. We enter the character in boxes and show the bit pattern. Then lacking this utility, we can surprise how we will recognize whether the original character is encoded through the two bits **01** or the three bits **010** or possibly impartial the original bit **0**? If we aspect at the encoding in the table overhead, we will realize that individual one of these possibilities is probable. It is not character that is encodes to the single bit **0** and none character that is encodes to the arrangement**010** or **0100** or **01000** designed for that substance. Yet, a character that encodes to **01** and that is **'S'**. One of the vital structures of the table created by Huffman coding is the prefix property: *C*haracters are not encoding of prefix or new one (i.e. if **'S'** is encoded by**01** then no other character’s encoding will twitchby**01** and character is not encoded to impartial**0**). By this assurance, there is no uncertainty in defining wherever the character limitations are. We twitch interpretation from the commencement, collecting bits in an order until we find a match. That directs the culmination of a character and we transfer on to decoding the succeeding character.

**6. Shannon Fano Encoding**

Shannon and Fano invented coding procedure to generate a binary code decimal tree [11]. Let the symbols Xi are coming from somewhere, such that 1 ≤ i ≤ 6, in the BCD (Binary Coding Decimal) format with probabilities P(Xi), is given in Table 1, at a rate Rs = 9.6 kbaud (baud=symbol/second).

State (i) the information rate and (ii) the data rate of the source.

**Table 1 Binary Coding Decimal probabilities.**

|  |  |  |
| --- | --- | --- |
| Xi | P(Xi) | BCD word |
| A | .12 | 000 |
| E | .10 | 001 |
| I | .28 | 010 |
| M | .02 | 011 |
| P | .14 | 100 |
| S | .34 | 101 |

We have to calculate the information rate and data rate before and after applying the Shanon Fano algorithm [11].

Before Shannon Fano Algorithm (With the help of Entropy)

1. **Entropy of source:**

*H = −Log2P(xi)*

= -.10 \* Log2 0.10 + -.12 \* Log2 0.12 + -.28 \* Log2 0.28 + -.02 \* Log2 0.02 + -.14 \* Log2 0.14 + -.34 \* Log2 0.34

= -.10 \* -3.3219 + -.12 \* -3.0588 + -.28 \* -1.8365 + -.02 \* -5.6438 + -.14 \* -2.8365 + -.34 \* -1.5563

= .33219 + .31265 + .51422 + .11287 + .39711 + .52914

***H= 2.19818 bits/symbol***

Information Rate R = H.Rs = 2.19818 \* 9600 = 21102.53 bits/sec = 21103 bits/sec

**Data Rate = 3 \* 9600 = 28800 (Before Shanon Fano)**

**ii) Shanon Fano Coding**

**Table 2 Binary Coding Decimal Codes**

|  |  |  |  |
| --- | --- | --- | --- |
| X | P(X) | Steps | Code |
| S | .34 | 0 | 0 |
| I | .28 | 1 0 | 10 |
| P | .14 | 1 1 0 | 110 |
| A | .12 | 1 1 1 0 | 1110 |
| E | .10 | 1 1 1 1 0 | 11110 |
| M | .02 | 1 1 1 1 1 1 | 111111 |

As we have taken only 6 characters (Bits) of possible code length, Therefore the maximum code word length = n – 1 = 5, where n = number of characters.

Average code word length with 5 buffer size is:

d = .34 \* 1 + .28 \* 2 + .14 \* 3 + .12 \* 4 + .10 \* 5 = .34 + .56 + .42 + .48 + .50 = 2.30 [bits/symbol]

Data Rate: ***d***\*RS = 2.30 \* 9.6 kbaud(Given) Kbaud = Kilobits/second

2.30 \* 9600 = 22080 (Bits/Second) (After Shanon Fano)

**Compression factor:** 3 [bits]/d [bits] = = 1.304

**Coding efficiency before Shannon-Fano:**

CE = = = 73.27%

**Coding efficiency after Shannon-Fano:**

CE = = = = 95.57% = 96%

Hence Shannon Fano Algorithm brought the coding efficiency closer to 100%.

**Table 3Access Time Performance of Database**

|  |  |  |
| --- | --- | --- |
| **Database (n)** | **Database Size** | **Evolution Time(Milliseconds) (X)** |
| NIU\_CSE | 170050 | 2500 |
| NIU\_ECE | 195000 | 2580 |
| NIU\_CE | 205560 | 2600 |
| NIU\_ME | 218310 | 2759 |
| NIU\_BIO | 220830 | 2801 |
| NIU\_EE | 230940 | 2950 |
| NIU\_EEE | 240050 | 2995 |
| NIU\_SBM | 245000 | 3001 |
| NIU\_SLA | 250500 | 3035 |
| NIU\_SOS | 260500 | 3050 |
| NIU\_LAW | 280500 | 3080 |
| NIU\_NURSING | 290500 | 3100 |
| NIU\_ARCH | 291000 | 3150 |
| NIU\_EDU | 292000 | 3160 |
| NIU\_FINEART | 293000 | 3190 |
|
| NIU\_MEDIA | 310500 | 3200 |
| NIU\_BDS | 320000 | 3250 |
|
| NIU\_MBBS | 330000 | 3290 |
|
| NIU\_BSW | 340500 | 3320 |
| NIU\_MSW | 345000 | 3380 |

**Figure 5 Access Time Performance of Database**

A database supplies huge volumes of departmental statistics and all particulars of a division. The above statistics can be applied to resolve a number of queries, and it can be applied to increase the functionality of an association in a short time. Not only users are able to access the database contents, but this data is also providing integrity. But also users are able to manage this information with current data. All the above types of problem can be resolved by proposed algorithm. The figure-5 depicts that as the size of database increases the performance related to access time is reducing consequently.

1. **Result of combination**

**Table 4 Combination of 24 Bit Ratio algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.no.** | **Types of combination** | **Result** | **Record** |
| 1 | ZRL+BWFT | 85.65 | 100 |
| 2 | BWFT+MTF+EC | 85.73 | 100 |
| 3 | BWFT+MTF+EC+SFE | 66.76 | 100 |
| 4 | ZRL+BWBT+EC+SFE+JBE | 56.73 | 100 |
| 5 | ZRL+BWT+MTF+JBE+**EJBE** | 55.42 | 100 |

**Figure 6 Combination of 24 Bit Ratio algorithms**

This Figure indicates that 24-bit bitmap images are compressed by decent compression ratio through algorithms that combined with Enhanced J-bit encoding.

**Table 5 Combination of 64 Bit Ratio algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.no.** | **Types of combination** | **Result** | **Record** |
| 1 | ZRL+BWFT | 89.4 | 100 |
| 2 | BWFT+MTF+EC | 89.52 | 100 |
| 3 | BWFT+MTF+EC+SFE | 80.51 | 100 |
| 4 | ZRL+BWBT+EC+SFE+JBE | 78.61 | 100 |
| 5 | ZRL+BWT+MTF+JBE+**EJBE** | 62.52 | 100 |

**Figure 7 Combination of 64 Bit Ratio algorithms**

This Figure indicates that 64-bit bitmap images are compressed by improved compression ratio through algorithms that combined by Enhanced J-bit encoding. A 64 bit bitmap image is most composite data than 24 bit it is enhanced more color. A 64 bit bitmap image are achieved greatest compression ratio, even supposed that will reduction excellence of the innovative image.

**Table 6 Comparison of 64 Bit Ratio algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.no.** | **Types of combination** | **Result** | **Record** |
| 1 | ZRL+BWFT | 64.81 | 100 |
| 2 | BWFT+MTF+EC | 65.82 | 100 |
| 3 | BWFT+MTF+EC+SFE | 54.51 | 100 |
| 4 | ZRL+BWBT+EC+SFE+JBE | 48.51 | 100 |
| 5 | ZRL+BWT+MTF+JBE+**EJBE** | 37.51 | 100 |

**Figure 8 Comparison of 64 Bit Ratio algorithms**

Above Figure indicates that text files are compressed by improved compression ratio through algorithms that combined by Enhanced J- bit encoding. It is reduced the size of database (37.51) by EJBE algorithm.

1. **CONCLUSION**

In this paper shows data compression algorithm compared to other algorithm and enhance J-Bit encoding algorithm. By research consuming 5 categories of files with 50 various dimensions for each category was showed, 5permutation algorithms has been confirmed and associated. This algorithm stretches improved compression ratio when inserted between move to front transform (MTF) and Zero Run length coding (ZRLC),Huffman coding etc. These algorithm compress 37.51% data has been compressed and we use other decompress algorithm burrow wheel back ward and Front backward algorithm. For the purpose that roughly files involve of hybrid contents as audio video and text etc. the capability to identify substances irrespective the file category, divided it then compresses it distinctly with suitable algorithm to the substances is possible for supplementary research in the upcoming to attain improved compression ratio. Burrow wheel transform algorithm is optimized used for the lossless data compression in data warehouse. We observed at the structure of database and the numerous phases of compression and the decompression, in which the phases are track in opposite direction compared to the compression. HEC Using Huffman coding, we can translate the communication into a string of bits and send it to you. However, you cannot decompress the communication, because you don't have the encoding tree that we used to direct the communication.

**References:**

[1] Capo-chichi, E. P., Guyennet, H. and Friedt, J. K-RLE, “***A New Data Compression Algorithm for Wireless Sensor Network”*** In Proceedings of the 2009 Third International Conference on Sensor Technologies and Applications.

[2]. Jonathan Pincus, Brandon Baker, “***Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns****”*, IEEE Coputer Society, 2004, pp.20-27.

[3] MahtabAlam, PrashantJohri, RiteshRastogi,  *“****Buffer Overrun: Techniques of Attack and Its Prevention***”, International Journal of Computer Science & Emerging Technologies (E-ISSN: 2044-6004)1 *Volume 1, Issue 3, October 2010*

[4] I Made Agus DwiSuarjaya A New Algorithm for Data Compression Optimization (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 3, No.8, 2012

[5] Ajit Singh and YogitaBhatnagar, ***”Enhancement of Data Compression Using Incremental Encoding****”*, International Journal of Scientific & Engineering Research, Volume 3, Issue 5, May-2012 1 ISSN 2229-5518.

[6] *Nirmal Sharma, S. K. Gupta, “****Design and Data Compression Techniques to Reduced Time in Data Warehouse with Tested Algorithms****”,*International Conference on Advanced Computing (ICAC-2016)*College of Computing Sciences and Information Technology (CCSIT)*, pp: 46-48, 2016

[7] Nelson, M. 1996. Data Compression with Burrows-Wheeler Transform. Dr. Dobb's Journal.

[8] Bell, T.C., Witten, I.H., Cleary, J.G.: Calgary Corpus: .Modeling for text Compression. Computing Surveys 21(4), 557-591, 1989.

[9] Campos, A. S. E. Move to Front. Available: http://www.arturocampos.com/ac\_mtf.html (last accessed July 2012).

[10] Daniel Schiller “***The Burrows-Wheeler Algorithm***” Research report, August 5, 2012

[11] Julie Zelenski, Keith Schwarz,”Huffman Encoding and Data Compression”, Spring May 2012

[12] Burrows, M.; Wheeler, D.J. A Block-Sorting Lossless Data Compression Algorithm; Research Report 124; Digital Equipment Corporation: Palo Alto, CA, USA, 1994