IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 11, Issue 6 (May. - Jun. 2013), PP 89-96
www.iosrjournals.org

Affable Compression through Lossless Column-Oriented Huffman Coding Technique

Punam Bajaj[1], Simranjit Kaur Dhindsa[2]
Computer Science Engineering Department, Chandigarh Engineering Collage, Landran, Mohali, Punjab

Abstract: *Compression is a technique used by many DBMSs to increase performance. Compression improves performance by reducing the size of data on disk, decreasing seek times, increasing the data transfer rate and increasing buffer pool hit rate [1]. Column-Oriented Data works more naturally with compression because compression schemes capture the correlation between values; therefore highly correlated data can be compressed more efficiently than uncorrelated data. The correlation between values of the same attribute is typically greater than the correlation between values of different attributes. Since a column is a sequence of values from a single attribute, it is usually more compressible than a row [4].*

In this paper we proposed the Lossless method of Column-Oriented Data-Image Compression and Decompression using a simple coding technique called Huffman Coding. This technique is simple in implementation and utilizes less memory [2]. A software algorithm has been developed and implemented to compress and decompress the created Column-oriented database image using Huffman coding techniques in a MATLAB platform.

Keywords- *Compression, Column-Oriented Data-Image Compression and Decompression, Huffman coding.*

I. Introduction:

Column-oriented DBMS's are currently under development. Column oriented DBMS's differ from Row-Oriented DBMS's in the layout of data on disk [4]. In Column Oriented each value of an attribute (column) is stored contiguously on disk; in a row store the values of each attribute in a tuple are stored contiguously. Compression is a technique used by many DBMSs to increase performance. Compression improves performance by reducing the size of data on disk, decreasing seek times, increasing the data transfer rate and increasing buffer pool hit rate [1]. Intuitively, data stored in columns is more compressible than data stored in rows. Column-oriented Compression algorithms perform better on data with low information entropy (high data value locality) [3]. Eg. Imagine a database table containing information about customers (name, phone number, e-mail address, e-mail address, etc.). Storing data in columns allows all of the names to be stored together, all of the phone numbers together, etc. Certainly phone numbers will be more similar to each other than surrounding text fields like e-mail addresses or names [4]. Further, if the data is sorted by one of the columns, that column will be super-compressible. Column data is of uniform type; therefore, there are some opportunities for storage size optimizations available in column-oriented data that are not available in row-oriented data. This has advantages for data warehouses and library catalogues where aggregates are computed over large numbers of similar data items [5].

Therefore, Column-Oriented Compression are better than traditional Row-oriented Compression as applications require higher storage and easier availability of data, the demands are satisfied by better and faster techniques [7].

II. Column-Oriented Compression

Compression is possible for data that are redundant or repeated in a given test set. Compression is a technique used by many DBMSs to increase performance. Compression improves performance by reducing the size of data on disk, decreasing seek times, increasing the data transfer rate and increasing buffer pool hit rate [1]. Intuitively, data stored in columns is more compressible than data stored in rows.
Compression is usually of three types:

- Data Compression
- Image Compression
- Graphical Compression

But in our paper, we are performing Data Compression by embedding that data into Images i.e. by using Column-Oriented Image Compression.
Column data is of uniform type; therefore, there are some opportunities for storage size optimizations available in Column-oriented data that are not available in Row-oriented data. Compression is useful because it helps reduce the consumption of expensive resources, such as hard disk space or transmission bandwidth.

Infobright is an example of an open source Column-Oriented DBMS built for high-speed reporting and analytical queries, especially against large volumes of data. Data that required 450GB of storage using SQL Server required only 10GB with Infobright, due to Infobright's massive compression and the elimination of all indexes. Using Infobright, overall compression ratio seen in the field is 10:1. Some customers have seen results of 40:1 and higher. Eg.1TB of raw data compressed 10 to 1 would only require 100 GB of disk capacity [5].

Customer's Test	**Alternative**	**Infobright**
Analytic Queries	2+ hours with MySQL	<10 seconds
1 Month Report (15MM Events)	43 min with SQL Server	23 seconds
Oracle Query Set	10 seconds- 15 minutes	0.43-22 seconds

Table 1 Performance Output Difference

Therefore, we can conclude that Column-Oriented Data Compression performs better than traditional Row-oriented Compression as applications require higher storage and easier availability of data, the demands are satisfied by better and faster techniques [7].

III. Image Compression

A digital image obtained by sampling and quantizing a continuous tone picture requires an enormous storage. For instance, a 24 bit color image with 512x512 pixels will occupy 768 Kbyte storage on a disk, and a picture twice of this size will not fit in a single floppy disk. To transmit such an image over a 28.8 Kbps modem would take almost 4 minutes. The purpose for image compression is to reduce the amount of data required for representing sampled digital images and therefore reduce the cost for storage and transmission. Image compression plays a key role in many important applications, including image database, image communications, and remote sensing.

The image(s) to be compressed are gray scale with pixel values between 0 to 255. There are different techniques for compressing images [6]. They are broadly classified into two classes called lossless and lossy compression techniques. As the name suggests in lossless compression techniques, no information regarding the image is lost. In other words, the reconstructed image from the compressed image is identical to the original image in every sense. Whereas in lossy compression, some image information is lost, i.e. the reconstructed image from the compressed image is similar to the original image but not identical to it. In this work we will use a lossless compression and decompression through a technique called Huffman coding (i.e. Huffman encoding and decoding) [6].

It's well known that the Huffman's algorithm is generating minimum redundancy codes compared to other algorithms. The Huffman coding has effectively used in text, image, video compression, and conferencing system such as, JPEG, MPEG-2, MPEG-4, and H.263etc.. The Huffman coding technique collects unique symbols from the source image and calculates its probability value for each symbol and sorts the symbols based on its probability value. Further, from the lowest probability value symbol to the highest probability value symbol, two symbols combined at a time to form a binary tree. Moreover, allocates zero to the left node and one to the right node starting from the root of the tree. To obtain Huffman code for a particular symbol, all zero and one collected from the root to that particular node in the same order [8].

IV. Need For Compression

Research indicates that the size of the largest data warehouses doubles every three years. According to Wintercorp's 2005 TopTen Program Summary, during the five year period between 1998 and 2003, the size of the largest data warehouse grew at an exponential rate, from 5TB to 30 TB. But in four year period between 2001 and 2005, that exponential rate increased, with the largest data warehouse growing from 10 TB to 100 TB [9].

To store these data including images, audio files, videos etc, and make them available over network (e.g. the internet), compression techniques are needed. Image compression addresses the problem of reducing the amount of data required to represent digital image. The underlying basis of the reduction process is the removal of redundant data. According to mathematical point of view, this amounts to transforming a two-dimensional pixel array into a statistically uncorrelated data set. The transformation is applied prior to storage or transmission of the image. At receiver, the compressed image is decompressed to reconstruct the original image or an approximation to it. The example below clearly shows the importance of compression. An image, 1024 pixel×1024 pixel×24 bit, without compression, would require 3 MB of storage and 7 minutes for transmission, utilizing a high speed, 64Kbits/s, ISDN line. If the image is compressed at a 10:1 compression ratio, the storage requirement is reduced to 300 KB and the transmission time drop to less than 6 seconds.

4.1 Principle behind Compression

A common characteristic of most images is that the neighboring pixels are correlated and therefore contain redundant information. The foremost task then is to find less correlated representation of the image. Two fundamental components of compression are redundancy and irrelevancy reduction.
a) Redundancies reduction aims at removing duplication from the signal source (image/video).
b) Irrelevancy reduction omits parts of the signal that will not be noticed by the signal receiver, namely the Human Visual System.
In an image, which consists of a sequence of images, there are three types of redundancies in order to compress file size. They are:
a) Coding redundancy: Fewer bits to represent frequently occurring symbols.
b) Inter-pixel redundancy: Neighboring pixels have almost same value.
c) Psycho visual redundancy: Human visual system cannot simultaneously distinguish all colors.

V. Various Types Of Redundancy

In digital image compression, three basic data redundancies can be identified and exploited:
a. Coding redundancy
b. Inter pixel redundancy
c. Psycho visual redundancy
Data compression is achieved when one or more of these redundancies are reduced or eliminated.

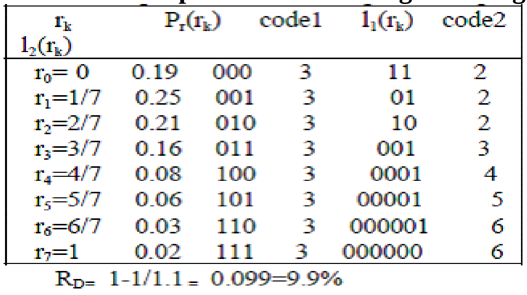
5.1 Coding Redundancy

A gray level image having n pixels is considered. Let us assume, that a discrete random variable r_k in the interval (0,1) represents the grey levels of an image and that each r_k occurs with probability $Pr(r_k)$. Probability can be estimated from the histogram of an image using
$Pr(r_k) = h_k/n$ for $k = 0,1……L-1$
Where L is the number of grey levels and h_k is the frequency of occurrence of grey level k (the number of times that the kth grey level appears in the image) and n is the total number of the pixels in the image. If the number of the bits used to represent each value of rk is $l(rk)$, the average number of bits required to represent each pixel is :

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) P_r(r_k)$$

Hence the number of bits required to represent the whole image is n x L_{avg}. Maximal compression ratio is achieved when L_{avg} is minimized. Coding the gray levels in such a way that the L_{avg} is not minimized results in an image containing coding redundancy. Generally coding redundancy is presented when the codes (whose lengths are represented here by $l(r_k)$ function) assigned to a gray levels don't take full advantage of gray level's probability ($Pr(r_k)$function). Therefore it almost always presents when an image's gray levels are represented with a straight or natural binary code. A natural binary coding of their gray levels assigns the same number of bits to both the most and least probable values, thus failing to minimize equation and resulting in coding redundancy**.**
Example of Coding Redundancy: An 8-level image has the gray level distribution shown in table I. If a natural 3-bit binary code is used to represent 8 possible gray levels, L_{avg} is 3- bits, because $l\ r_k$= 3 bits for all r_k. If code 2 in table I is used, however the average number of bits required to code the image is reduced to:
L_{avg} = (0.19) + 2(0.25) +2(0.21) + 3(0.16) + 4(0.08) + 5(0.06) + 6(0.03) + 6(0.02) =2.7 bits.
From equation of compression ratio (n2/n1) the resulting compression ratio *CR* is 3/2.7 = 1.11. Thus approximately 10% of the data resulting from the use of code 1 is redundant. The exact level of redundancy can be determined from equation R_D = 1 – 1/1.11 =0.099.

Table I: Example of Variable Length Coding



r_k	$P_r(r_k)$	code1	$l_1(r_k)$	code2	$l_2(r_k)$
r_0= 0	0.19	000	3	11	2
r_1=1/7	0.25	001	3	01	2
r_2=2/7	0.21	010	3	10	2
r_3=3/7	0.16	011	3	001	3
r_4=4/7	0.08	100	3	0001	4
r_5=5/7	0.06	101	3	00001	5
r_6=6/7	0.03	110	3	000001	6
r_7=1	0.02	111	3	000000	6

R_D= 1-1/1.1 = 0.099=9.9%

It is clear that 9.9% data in first data set is redundant which is to be removed to achieve compression.

5.1.1 Reduction of Coding Redundancy

To reduce this redundancy from an image we go for the Huffman technique where we are assigning fewer bits to the more probable gray levels than to the less probable ones achieves data compression. This process commonly is referred to as variable length coding. There are several optimal and near optimal techniques for constructs such a code i.e. Huffman coding, Arithmetic coding etc.

5.2 Inter pixel Redundancy

Another important form of data redundancy is inter-pixel redundancy, which is directly related to the inter-pixel correlations within an image. Because the value of any given pixel can be reasonable predicted from the value of its neighbors, the information carried by individual pixels is relatively small. Much of the visual contribution of a single pixel to an image is redundant; it could have been guessed on the basis of its neighbor's values. A variety of names, including spatial redundancy, geometric redundancy, and inter frame redundancies have been coined to refer to these inter-pixel dependencies. In order to reduce the inter-pixel redundancies in an image, the 2-D pixel array normally used for human viewing and interpretation must be transformed into a more efficient but usually non-visual format. For example, the differences between adjacent pixels can be used to represent an image. Transformations of this type are referred to as mappings. They are called reversible if the original image elements can be reconstructed from the transformed data set.

5.2.1 Reduction of Inter-pixel Redundancy

To reduce the inter-pixel redundancy we use various techniques such as:
1. Run length coding.
2. Delta compression.
3. Predictive coding.

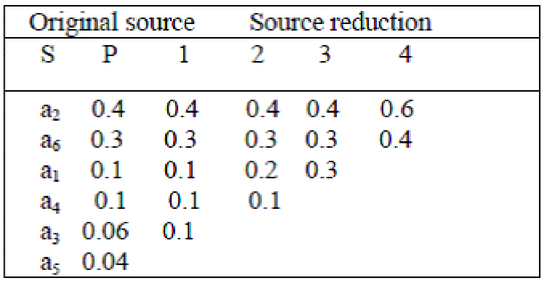
5.3 Psycho visual Redundancy

Human perception of the information in an image normally does not involve quantitative analysis of every pixel or luminance value in the image. In general, an observer searches for distinguishing features such as edges or textural regions and mentally combines them into recognizable groupings. The brain then correlates these groupings with prior knowledge in order to complete the image interpretation process. Thus eye does not respond with equal sensitivity to all visual Information. Certain information simply has less relative importance than other information in normal visual processing. This information is said to be psycho visually redundant. It can be eliminated without significantly impairing the quality of image perception. Psycho visual redundancy is fundamentally different from the coding Redundancy and inter-pixel redundancy. Unlike coding redundancy and inter-pixel redundancy, psycho-visual redundancy is associated with real or quantifiable visual information. Its elimination is possible only because the information itself is not essential for normal visual processing. Since the elimination of psycho-visual redundant data results in a loss of quantitative information. Thus it is an irreversible process.

5.3.1 Reduction of Psycho visual Redundancy

To reduce psycho visual redundancy we use Quantizer. Since the elimination of psycho-visually redundant data results in a loss of quantitative information. It is commonly referred to as quantization. As it is an irreversible operation quantization results in lossy data compression [].

Table II: Huffman Source Reductions



Original source		Source reduction			
S	P	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

S-source, P-probability

VI. Implementation Of Lossless Compression And Decompression Techniques

6.1 Huffman coding

Huffman code procedure is based on the two observations.

a. More frequently occurred symbols will have shorter code words than symbol that occur less frequently.

b. The two symbols that occur least frequently will have the same length. The Huffman code is designed by merging the lowest probable symbols and this process is repeated until only two probabilities of two compound symbols are left and thus a code tree is generated and Huffman codes are obtained from labeling of the code tree. This is illustrated with an example shown in table II:

Table III: Huffman Code Assignment Procedure

Original source reduction			Source reduction			
S	P	code	1	2	3	4
a_2	0.4	1	0.4 1	0.4 1	0.4 1	0.6 0
a_6	0.3	00	0.3 00	0.3 00	0.3 00	0.4 1
a_1	0.1	011	0.1 011	0.2 010	0.3 01	
a_4	0.1	0100	0.1 0100	0.1 011		
a_3	0.06	01010	0.1 0101			
a_5	0.04	01011				

S-source, P-probability

At the far left of the table I the symbols are listed and corresponding symbol probabilities are arranged in decreasing order and now the least t probabilities are merged as here 0.06 and 0.04 are merged, this gives a compound symbol with probability 0.1, and the compound symbol probability is placed in source reduction column1 such that again the probabilities should be in decreasing order. So, this process is continued until only two probabilities are left at the far right shown in the above table as 0.6 and 0.4. The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to its original source [3]. The minimal length binary code for a two-symbol source, of course, is the symbols 0 and 1. As shown in table III these symbols are assigned to the two symbols on the right (the assignment is arbitrary; reversing the order of the 0 and would work just and well). As the reduced source symbol with probabilities 0.6 was generated by combining two symbols in the reduced source to its left, the 0 used to code it is now assigned to both of these symbols, and a 0and 1 are arbitrary appended to each to distinguish them from each other. This operation is then repeated for each reduced source until the original course is reached. The final code appears at the far-left in table 1.8. The average length of the code is given by the average of the product of probability of the symbol and number of bits used to encode it. This is calculated below:

Lavg = (0.4)(1) +(0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) = 2.2bits/ symbol and the entropy of the source is 2.14bits/symbol, the resulting Huffman code efficiency is 2.14/2.2 = 0.973.

Huffman's procedure creates the optimal code for a set of symbols and probabilities subject to the constraint that the symbols be coded one at a time.

6.2 Huffman decoding

After the code has been created, coding and/or decoding is accomplished in a simple look-up table manner. The code itself is an instantaneous uniquely decodable block code. It is called a block code, because each source symbol is mapped into a fixed sequence of code symbols. It is instantaneous, because each codeword in a string of code symbols can be decoded without referencing succeeding symbols. It is uniquely decodable, because any string of code symbols can be decoded in only one way. Thus, any string of Huffman encoded symbols can be decoded by examining the individual symbols of the string in a left to right manner. For the binary code of table III, a left-to-right scans of the encoded string 010100111100 reveals that the first valid code word is 01010, which is the code for symbol a3. The next valid code is 011, which corresponds to symbola1. Valid code for the symbol a2 is 1, valid code for the symbols a6 is 00, valid code for the symbol a6 is continuing in this manner reveals the completely decoded message a5 a2 a6 a4 a3 a1, so in this manner the original image or data can be decompressed using Huffman decoding as explained above.

At first we have as much as the compressor does a probability distribution. The compressor made a code table. The decompressor doesn't use this method though. It instead keeps the whole Huffman binary tree, and of course a pointer to the root to do the recursion process. In our implementation we'll make the tree as usual and then you'll store a pointer to last node in the list, which is the root. Then the process can start. We'll navigate the tree by using the pointers to the children that each node has. This process is done by a recursive function which accepts as a parameter a pointer to the current node, and returns the symbol.

VII. Quality Measures:

7.1 Peak Signal To Noise Ratio:

The Peak Signal to Noise Ratio (PSNR) is the ratio between maximum possible power and corrupting noise that affect representation of image. PSNR is usually expressed as decibel scale. The PSNR is commonly used as measure of quality reconstruction of image. The signal in this case is original data and the noise is the error introduced. High value of PSNR indicates the high quality of image.
It is defined via the Mean Square Error (MSE) and corresponding distortion matric, the Peak Signal to Noise[10].

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

7.2 Mean Square Error

Mean Square Error can be estimated in one of many ways to quantify the difference between values implied by an estimate and the true quality being certificated. MSE is a risk function corresponding to the expected value of squared error. The MSE is the second moment of error and thus incorporates both the variance of the estimate and its bias[10].

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

VIII. Development Steps of Column-Oriented Huffman Coding and Decoding Algorithm

Step1- Plot the interested Columns of column -oriented database in workplace of MATLAB.
Step2- Convert the given figure into grey level image.
Step3- Read the image on to the workspace of the MATLAB.
Step4- Call a Column-Oriented Huffman Coding Algorithm.
Step5- Following five figures are generated as results.
Figure 1: Construction of Image from Column-Oriented Database.
Figure 2: Image Encoding Steps from 1-6
Figure 3: Final Image Encoding Steps.
Figure 4: Image Decoding Steps from 1-6
Figure 5: Final Image Decoding Steps.
Step 6- Calculate values of MSE, PSNR and Elapsed Time.

IX. Results:

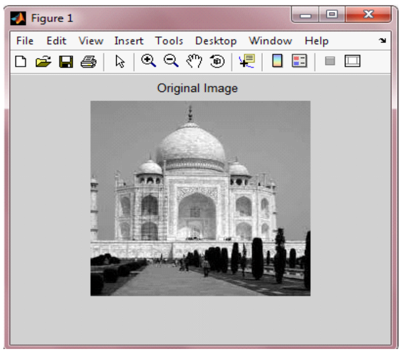


Fig1: Construction of Image from Column-Oriented Database.

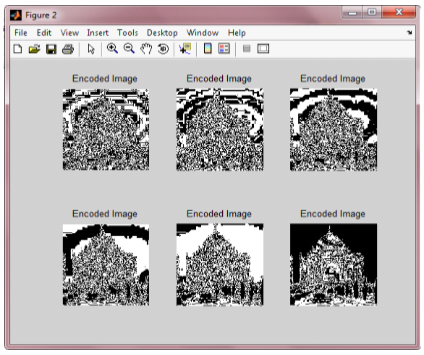


Fig2: Image Encoding Steps from 1-6

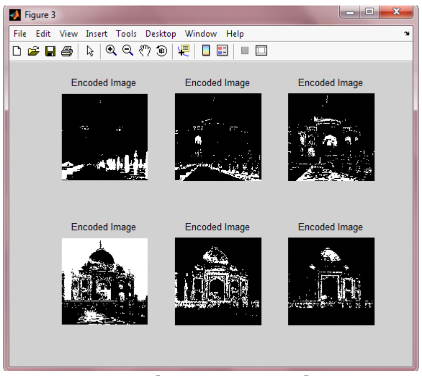


Figure 3: Final Image Encoding Steps.

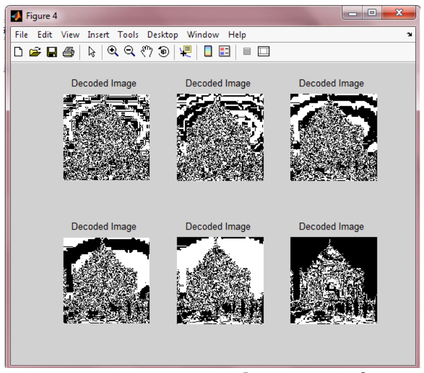


Figure 4: Image Decoding Steps from 1-6

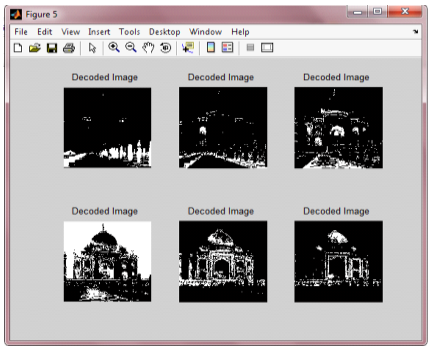


Figure 5: Final Image Decoding Steps

The input image shown in Fig.1 to which the above Huffman coding algorithm is applied for the generation of codes and then decompression algorithm (i.e. Huffman decoding) is applied to get the original image back from the generated codes, which is shown in the Fig.3. The number of saved bits is the difference between the number of bits required to represent the input image i.e. shown in the table II by considering each symbol can take a maximum code length of 8 bits and the number of bits taken by the Huffman code to represent the compressed image i.e. Saved bits = (8*(r*c)-(l1*l2))=3212, r and c represents size of the input matrix, l_1 and l_2 represents the size of Huffman code. The compression ratio is the ratio of number of bits required to represent the image using Huffman code to the number of bits used to represent the input image. i.e. Compression ratio = $(l_1 * l_2)/ (8*r*c) = 0.8456$, The output image is the decompressed image i.e. from the Fig.5 it is clear that the decompressed image is approximately equal to the input image.

X. Conclusion

The experiment shows that the higher data redundancy helps to achieve more compression. The above presented a new Column-Oriented compression and decompression technique based on Huffman coding and decoding for scan testing to reduce test data volume, test application time.

Assessment for image quality is a traditional need. The conventional method for measuring quality of image is MSE & PSNR. In this paper we compared the different image enhancement techniques by using their quality parameters (MSE & PSNR). Experimental results show that

- Both PSNR and MSE are inversely proportional to each other.
- Whose PSNR is High, the Image Compression is Better.

MSE=2.2710e+004 , PSNR= 4.5687 dB and Total Elapsed Time=133.3965

Therefore, better compression ratio for the above image is obtained. Hence we conclude that Column-Oriented Huffman coding is efficient technique for image compression and decompression. As the future work on compression of images for storing and transmitting images can be done by other lossless methods of image compression because as we have concluded above the result the decompressed image is almost same as that of the input image so that indicates that there is no loss of information during transmission. So other methods of image compression can be carried out as namely JPEG method, Entropy coding, etc.

References

[1] Miguel C. Ferreira, 'Compression and Query Execution within Column Oriented Databases'.

[2] Jagadish H. Pujar, Lohit M. Kadlaskar, 'A New Lossless Method Of Image Compression And Decompression Using Huffman Coding Techniques', Journal of Theoretical and Applied Information Technology, © 2005 - 2010 JATIT.

[3] Sushila Aghav, "Database compression techniques for performance optimization", 2010 IEEE, V6-714.

[4] Daniel J. Abadi, Query Execution in Column-Oriented Database Systems, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, June 2005 (c) Massachusetts Institute of Technology 2005.

[5] Infobright,"Analytic Applications With PHP and a Columnar Database", 403-47 Colborne St Toronto, Ontario M5E 1P8 Canada.

[6] C. Saravanan, M. Surender, 'Enhancing Efficiency of Huffman Coding using Lempel Ziv Coding for Image Compression', International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-6, January 2013

[7] Daniel J. Abadi, Peter A. Boncz, Stavros Harizopoulos,'Column-oriented Database Systems', *VLDB '09*, August 24-28, 2009, Lyon, France.

[8] C. Saravanan & R. Ponalagusamy "Lossless Grey-scale Image Compression using Source Symbols Reduction and Huffman Coding", International Journal of Image Processing (IJIP), Volume (3): Issue (5).

[9] SyBase, David Loshin (President, Knowledge Integrity Inc) , ' Gaining the Performance Edge Using a Column-Oriented Database Management System'. en.wikipedia.org/wiki/Peak_Signal-to-noise_ratio.