Asian Journal of Information Technology 4 (12): 1180-1186, 2005

Efective Compression Algorithm for Medical Images as an Aid to Telemedicine

Janet, J. and T.R. Natesan
Sathyabama Institute of Science and Technology, Chennai 119, India

Abstract: This study provides an improved compression ratio over the existing compression algorithm of Huffman. The algorithm is tested on many kinds of files and is best suited for the Medical images like MRI, CTSCAN, and ULTRASOUND Medical images etc., which are mainly BMP images. Huffman Compression, also known as Huffman encoding, an algorithm for the lossless compression of files is based on the frequency of occurrence of a symbol in the file that is being compressed. It is a type of statistical coding, in which frequently occurring characters are given a short sequence while others that are used seldom get a longer bit sequence. The present system automates the whole process with the added advantage in the form of pattern recognition and footer information concept as a result of which an improved compression ratio over Huffman is achieved. The medical enterprise depends on a system that makes diagnostic images available for interpretation, that transmits images to physicians throughout the system, and that efficiently stores images pending retrieval for future medical or legal purposes. medical. Thus this study attempts to evaluate the performance of efficient and state of the art compression technique as applied to different types of medical images like CTScans, MRI, PET, Ultrasound, x-ray, Angiography Images etc We suggest a novel approach for compressing images or text documents based on building up a collection of n-length patterns in the image, and present the results of a network transfer implementation of this algorithm. It achieves better compression than existing alternate system (Huffman), and the decompression time appears substantially same as the previous method with almost the same compression rate and with no added complexity or resources. Thus, our Effective compression algorithm increases the compression ratio over Huffmann as well as maintains the quality of the Original image like Huffman.

Key words: Huffman compression, encoder, pattern recognition, footer information

INTRODUCTION

Data Compression can be defined as reducing of the amount of storage space required to store a given amount of data. Data compression comes with a lot of advantages, it saves storage space, and bandwidth, and cost and TIME required transmitting data from one place to another[1].

Our area of interest is compression of text files and Images. Here we deal with the concepts of developing a compression algorithm better than Huffman compression, which is the best-known Lossless compression technique.

Compression for medical images: Medical images give information of shape and function of organs of human body, being one of the most important means for establishing the diagnosis. They are a special means for controlling the therapeutic action[1].

Medical field is undergoing a profound transition from interpretation of images displayed on film to reading images on high-resolution computer monitor screens. The medical enterprise depends on a system that makes diagnostic images available for interpretation, that transmits images to physicians throughout the system, and that efficiently stores images pending retrieval for future medical or legal purposes. Computerized medical imaging generates large, data-rich electronic files. Hence there remains a strong demand for medical image compression.Thus the object of this study is evaluate the performance of efficient and state of the art compression technique as applied to different types of medical images like CTScan, MRI, PET, Ultrasound, x-ray, Angiography Images etc.

Huffman -a basic compression technique: Huffman compression is a Lossless compression technique.

Huffman coding is a statistical coding technique that is the basis for many compression techniques. Huffman can be relatively easy to implement and very economical for both coding and decoding purposes[1]. Arithmetic algorithm is difficult to implement and its complexity increases with file size. While Arithmetic coding needs more CPU power, Run Length Algorithm doesn't have many compression ratios and works well only for redundant characters. Dictionary wise compression has the need of providing the dictionary at both ends before processing though encoding of dictionary requires Huffman at later stages. JPEG, MPEG, Fractal Image

compression, LZ78, PKZIP, ARJ uses Huffman at some point in their algorithms. Modems come with built in techniques using (e.g.) MNP-5 algorithm. In fractal compression algorithm transformation step uses Huffman on their parameters.

Huffman algorithm can be used in the case of medical image compression where there should not be any loss of information during compression that will affect proper diagnosis.

Huffman compression (an observation to the existing work): One important method of transmitting messages is to transmit in their place sequences of symbols. If there are more messages, which might be sent than there are kinds of symbols available, then some of the messages must use more than one symbol. If it is assumed that each symbol requires the same time for transmission, then the time for transmission (length) of a message is directly proportional to the number of symbols associated with it.

Huffmancoding shares most characteristi of Shannon-Fano coding. It creates variable length codes that are an integral number of bits. Symbols with higher probabilities get shorter codes. Huffman codes have the unique prefix attribute, which means they can be correctly decoded despite being variable length. Decoding a stream of Huffman codes is generally done by following a binary decoder tree. Minimum redundancy code generation was the basic principle of Huffman coding, which is also called as optimum coding. This means that individual symbols are replaced by bit sequences that have a distinct length[2].

In general, data compression consists of taking a stream of symbols and transforming them into codes. If the compression is effective, the resulting stream of codes will be smaller than the original symbols. The decision to output a certain code for a certain symbol or set of symbols is based on the model. The model is simply a collection of data and rules used to process input symbols and determine which code[s] to output. A program uses the model to accurately define the probabilities for each symbol and the coder to produce an appropriate code based on those probabilities.

Modeling and coding: are two distinctly different things. We frequently use the term coding to refer to the entire data-compression process instead of just a single component of that process. Using the example of Huffman coding, a breakdown of the compression process looks like Fig. 2.3.2.1.In the case of Huffman coding, the actual output of the encoder is determined by a set of probabilities. Model and the program's coding process are different because of the countless ways to model data, all of which can use the same coding process to produce their outputs. A simple program using Huffman coding, for example, would use a model that gave the raw probability of each symbol occurring anywhere in the input stream. A more sophisticated program might calculate the probability based on the last 10 symbols in the input stream. Even though both programs use Huffman coding to produce their output, their compression ratios would probably be radically different.

General steps for Huffman compression:

Step 1: Lay out the code words in decreasing frequency.
Step 2: Take the two items with lowest frequency make them a constituent. Its frequency, of course is the sum of the frequencies that compromise. Repeat Step 2 as necessary, treating constituents the same as original items.
Step 3: Use the tree thereby constructed to assign 0's and 1's to the leaves of the tree based on direction of branching.

Justification of the present work:

- Compression ratio achieved is not very satisfactory in Huffman. Effective compression algorithm's main objective is to increase the compression ratio
- Compression ratio and storage space is *inversely proportional*. Hence increasing the compression ratio *decreases the storage space*.
- Effective compression algorithm maintains the quality of the Original image like Huffman.

We suggest a novel approach for compressing images or text documents based on building up a collection of n-length patterns in the image, and present the results of a network transfer implementation of our new work. Compression does have its trade-offs. The more efficient the compression technique, the more complicated the algorithm will be and thus, requires more computational resources or more time to decompress. But our work achieves better compression than existing alternate system (Huffman), and the decompression time appears substantially same as the previous method with almost the same compression rate and with no added complexity or resources. It's a straightforward extension of existing Huffman.

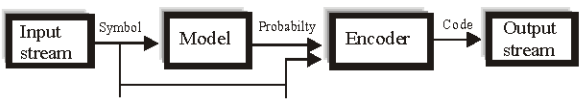


Fig. 1: A statistical model of huffman encoder

SYSTEM DESIGN

Overview of new compression algorithm: Basic model of new compression algorithm is similar to that of the Huffman except for the pattern finder. The operation of the pattern finder is to find the best pattern, which is the most frequent occurring pattern. Therefore the best pattern will also be an input to the encoder. The output of the encoder will be the code along with the footer information.

The impact of this research on the future of information technology is to develop data delivery systems where communication bandwidth is at a premium and storage is an exponentially costly endeavor. It is expected that this new lossless text and image compression algorithm will have 4 to 5% improved compression ratios over the best known pre-existing Huffman compression algorithm which might translate into a reduction of the traffic on the Internet.

Working steps for new algorithm: There are 4 basic steps to implement this new compression method to reach an effective compression ratio. On the other hand they does not impose an added complexity to the existing system. They include:

- The pattern is restricted to length 3 for our work.
- The first and last characters in the pattern must not be same as the second (middle) character to be replaced by Footer information.
- The pattern once traced for its positions must not be traced again. (I.e.) Each pattern must be traced only once.
- Positions of the patterns and subpatterns traced must be accurate enough to carry out the work.

These are the steps that are to be performed for replacing the first best pattern or 1-pattern replacement of length 3.They can be extended when we replace 2-best patterns and so on.

Idea of pattern recognition: The idea is based on the redundant nature of character or signals in the data we encounter. Consider for example a 3-length pattern commonly occurring in text files say ABC. We can also encounter its sub pattern of length 2 (i.e.)AC. So we encode all the ABC patterns as AC in the corresponding compressed version. To differentiate between ABCs and ACs in order to bring out the exact original file on decoding, the concept of a new feature called Footer Information is introduced. Through this footer information bits we add 1s to indicate the presence of Bin ABCs we encounter and 0s to denote the absence of B in the sub

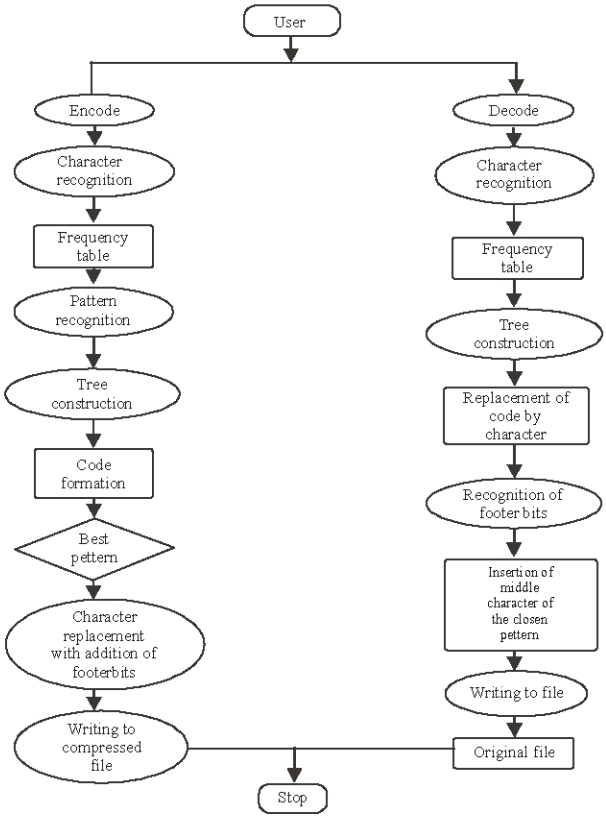


Fig. 2: DFE for new compression technique

patterns ACs. These bits are added at the last of compressed file after coding other characters in the file. These extra sets of bits are called footer bits, as they resemble the footer note of a word document that is added at the end of a page. Generally to identify the start of Footer bits from the normal compressed bits we add the codes for first and third characters found in the pattern being selected (i.e.) codes of A and C are added in footer information. They also help in the identification of the subpatterns between which the reduced character namely B(here) have to be inserted during decompression.

Focusing its functionality on medical images and search of patterns: The original medical image considered for compression can be of any type namely scan images, or x-ray or MRI etc. As the first step all the characters are read with an initial note of forming the frequency table. The count for each character is entered in their corresponding ASCII position in the output file. Now the file is searched for 3-length patterns with the initial conditions stated above. This aim of finding the patterns forms the basic differences from original method. The positions of occurrence of each and every pattern throughout the file are stored with their count of occurrence. Each and every pattern has its corresponding

subpattern formed by their first and last character (i.e.) omitting their middle character. These subpatterns are also searched for its occurrence throughout the file. Their positions and their count are stored along with their corresponding pattern's positions and count.

After tracing the patterns selection is made for the best pattern that can yield the maximum amount of bits, that can be reduced.This is done by forming an array of total bits that can be reduced for each and every pattern. The total bits that can be reduced are calculated by forming the product of the code length of the middle character of each and every pattern of length 3 that are being stored and their count. Hence a tree data structure is constructed. Based on the codes formed for middle character for the patterns, the maximum bits that can be reduced are noted. Then this large sum is subtracted from the number of footer bits being added to the file. This is obtained by forming the sum of counts of occurrences of the pattern and its corresponding subpattern. The total bits that can be reduced for each pattern are then stored in a new array for each pattern. Then we sort the patterns in the descending order to decide for the best pattern. This best pattern is our main concern to work out with.

After deciding for the best pattern we go for forming the footer bits. Any footer information starts with the codes for the first and the third character of the chosen pattern. The remaining bits are formed based on the position of occurrences of the selected pattern and its corresponding subpattern. This is done by getting the index of the best pattern from the stored pattern array. Based on this index 1 is added to the footer information if the first encountered position is from patterns list ,else a zero is added if the first encountered position belongs to that of subpatterns.

In case we are unable to find any best pattern, we follow the normal Huffman method. The compressed file is formed by replacing each character with their codes except the middle characters in the chosen pattern, which are checked by their positions already traced. The footer bits are added at the last after adding compressed codes.The set of bytes so formed make up a compressed file. The compressed file includes an extension
. hff1 to the original bmp file.

$$\text{\% of compression ratio} = \frac{\text{(Original filesize)-(Compressed filesize)}}{\text{(Original file size)}} \times 100$$

As the compressed file size gets reduced further the amount of compression done here increases than the Huffman model.

Formulation: The formula present demonstrates the efficiency of our work. By bringing out the basic assumptions and cases, we finally justify that the new compression ratio obtained by the above working method will always be grater than or equal to compression ratio obtained by Huffman.

Assume:

Maximum code length (i.e.) size of compressed file by Huffman = x bytes
Huffman Compression Ratio = n1
Code length of the middle character of the best pattern = m
Total number of occurrence of the best pattern = rp
Let number of bits that have maximum possibility of getting reduced = Rp
(I.e.) Rp=rp * m
Total no of occurrence of subpattern = sp
Footer information size (Fp) = φ(c1, c2, c3) + φ(c1, c3)
(I.e.) Fp = rp + sp
Maximum bits that can be reduced finally (Bp) =Rp- Fp
Maximum Bytes that can be reduced is = (Rp-Fp) / 8
New Compression Ratio (n2) = n1+Bp
Where Bp>=0 (i.e.) the number of bits gets reduced may be null or greater than zero.
Assuming two basic cases,

Case 1:

When (Rp-Fp)=0 (i.e.) no bits gets reduced
Bp=0
n2=n1+0
n2=n1

Case 2:

When (Rp-Fp)>0 (i.e.) some no of bits gets reduced
Bp>0
n2=n1+Bp
n2>n1

By these two cases we conclude that new compression ratio is always equal to or greater than the compression ratio obtained by Huffman.
n2>=n1

This shows that as the compression ratio of new compression method increases than the original Huffman

Maximum code length (i.e.) size of compressed file by new compression method <= x bytes

This shows that the new compression ratio is always dependent on Huffman's compression ratio as well as on the number of bits gets reduced on replacing the patterns. As more number of bits gets reduced the compression ratio increases largely. Thus we are able to prove that n2 is directly proportional to Bp.
n2 α Bp

This formulation is generalized for n-best patterns replacements as well as for n-length pattern replacements assisted via a single set of footer information bits.

Working of decoder: Our decoder meets all the basic requirements of an efficient decoder. It takes the compressed file with . hff1 as input. It includes no other extra information regarding name of original file, its size etc. This input file is made up of set of bytes, which encloses in them the frequency table, the compressed codes for the characters found in original file and also the footer information. Thus these help the decoder to built the original image back to form with exact color combinations. We also focus on the differences in working with Huffman's decoder.

Frequency table and tree construction: The frequency table denotes the exact count of occurrence of the 256 ASCII values in the original file. These values are in turn used to construct the tree as in Huffman. On the other hand they help in providing a check over the count of the characters formed while decoding which would help in exact tracing of footer information. Here we aim to do the reverse process of using those codes to bring down the exact character. The following bits can be used to trace from the root of the tree till we encounter a leaf which substitutes the exact character for the code. The characters once formed are written to a file but instead they are stored in array structure to go for insertion at later period. These character tracking is same as in Huffman. But there is an extra step is performed (i.e.) each time a character is formed, its count is incremented. This count is checked with the original count from the frequency table noted above. This mainly helps in identifying the start of footer bits. The count formed must be always less than or equal to its frequency value. This is the point at which he decoder differs from that of Huffman.

Tracking the file content and insertion: If we don't find any character with their count greater than the frequency value then we can write those characters to the file. This doesn't require any insertions to be followed as this situation indicates that the file is being encoded by normal Huffman compression method (due to absence of any best patterns). On the other hand at one stage of decoding the bits, we encounter the count value of the formed character is more than its original frequency count. This indicates the start of footer bits. The code is decoded to form the next character whose count also increases than the original frequency count. The following bits are not further decoded. They are rather stored as footer bits based on which the missed character gets inserted at appropriate positions.

The last 2 characters tracked are the surrounding characters of the pattern between which the insertion has to be made. Though this is an additional work done this doesn't affect the rate of decompression for the user. The decompression speed is always same as Huffman, which is an added advantage to our present work. The character that has to be inserted (i.e.) the middle character of the pattern is obtained by checking for the frequency counts of each and every character with their counts formed after tracking them. We can get just one character, which has the above condition satisfied, as it is the middle character (which has been avoided in the whole file of characters during encoding of file). Now the proper insertion begins by checking for the occurrence of subpattern in the formed character array. If the subpattern occurs then we check for the footer bits in sequence. If the footer bit is 1 then we insert the middle character between the 2 characters (subpattern) and write to the output file. If the footer bit is 0 then we don't perform the insertion which denotes the occurrence of the subpattern in that position. We repeat the process till end of footer.

The name of the output file can be traced from that of the compressed one by omitting the .hff1 which gives the original file's name. The file that has been formed after decompression is the original file. So we recover the original file back with same kind of image and with the same size as in Huffman with the same speed. This proper decoding ensures the successful application of our present work without any additional complexity or added mechanisms. This also denotes that more compression can be made to any kind of image or text file without any loss in their original quality or state of information.

OPTIMIZATION

Overview of the optimized work: In the optimization part we go for further improvising the new compression algorithm by selecting out two best patterns each of length 3 but assisted via a single set of footer information. In simple words it's the extension to our one pattern replacement. This part helps in increasing the compression percentage by 2-3% more than the single pattern and by 4-5% than the Huffman method. Optimization of our research is done to introduce the basic underlying fact that our new algorithm is not restricted for either 3-length pattern replacement or single pattern replacement.

$$\text{\% of compression ratio} = \frac{\text{(Original filesize)-(Compressed filesize)}}{\text{(Original file size)}} \times 100$$

Thus the amount of compression increases than Huffman and Single pattern method but without any loss of data or quality of the original image without affecting much the compression speed.

RESULTS AND DISCUSSION

Result of Huffman module: This image is first compressed and decompressed by Huffman method. On compressing by Huffman we provide a summary of result in a JOptionPane.This summary contains the details of the input image like the File name, Original file size in Bytes, Compressed file size in bytes and also the amount of compression being done. The original file size is noted to be 24762 bytes. On compression the size gets reduced to 15129 bytes with nearly 38.90235% compression done. On decompressing the file regain its original image with decompressed file size as 24762 bytes.

Results of single pattern replacement: This image is next compressed and decompressed by our present method. On compressing by single pattern replacement method we provide an indication of steps being done in order. The steps includes:

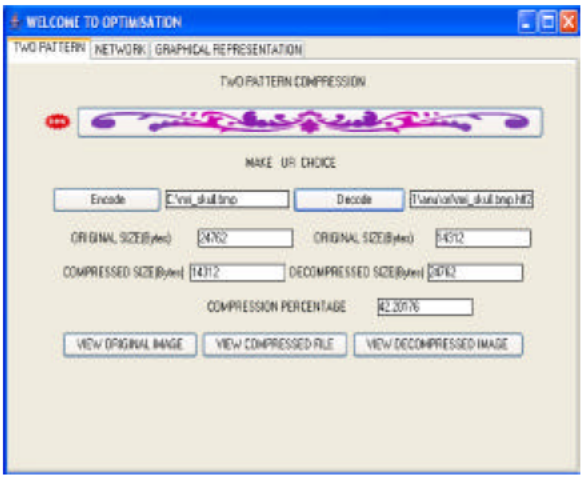
- Reading File
- Checking pattern
- Constructing Tree
- Footer calculation
- Adding Footer Bits
- Writing File

` This summary contains the details of the input image like the File name, Original file size in Bytes, Compressed file size in bytes and also the amount of compression being done. The original file size is noted to be 24762 bytes. On compression the size gets reduced to 14628 bytes with nearly 40.925613% compression done. On decompressing the file regain its original image with decompressed file size as 24762 bytes.

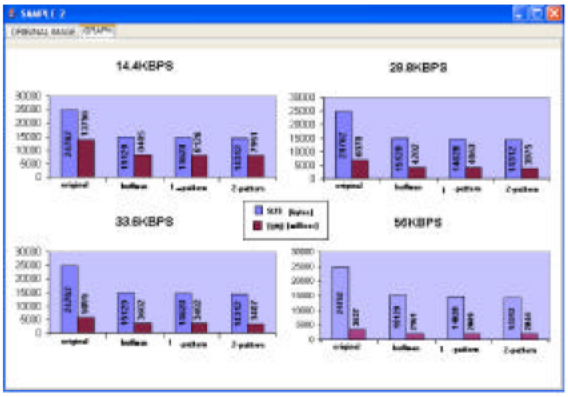
Proof of Efficiency: On providing the summary details on both Huffman and Single pattern model in a single screen for easy comparison we observe that there are nearly 2 % more compression done with a reduction of 501 bytes in the present work than Huffman. This clearly indicates the success of our research. The decompressed sizes indicate that original files are regained backed without any loss of data or quality of information maintaining the compression and decompression rate by both the methods of working.

Results of Two pattern replacement: This image is next compressed and decompressed by our optimized method. On compressing by two-pattern replacement method we provide an indication of steps being done in order. They include the same as single pattern replacement model but go with 2 best patterns. This selection of extra pattern takes no time and hence the speed of compression maintained as in single pattern working.

The summary contains the details of the input image like the File name, Original file size in Bytes, Compressed file size in bytes and also the amount of compression being done. The original file size is noted to be 24762 bytes. On compression the size gets reduced to 14312 bytes with nearly 42.20176% compression done. There is nearly 1.5 % more compression done with a reduction of 316 bytes in the optimized work than single pattern model. There are nearly 3.4 % more compression done with a reduction of 817 bytes in the optimized work than Huffman. On decompressing the file regain its original image with decompressed file size as 24762 bytes. This denotes the success of our optimized model, which is mainly done to bring the concept of extensibility nature to our present idea.



Results of Network module and graph: The sample-input image is then given to network module to denote the improvements over the transfer time along the 4 specific bandwidths. There is an average reduction of nearly 1-5 milliseconds by present algorithm



CONCLUSIONS

Here we present a new idea of achieving more compression percentage than the existing Huffman.Based on this idea of using the concept of redundancies in patterns we can witness the effectiveness in underlying method. This process of utilizing the redundant patterns for our work can attain more compression without any loss of information. This kind of losslessness is a basic need for any kind of medical image and normal texts. We restricted our work to search for patterns of length 3.The best patterns are replaced in the compressed file by a new set of footer bits instead of providing them the codes directly.

We aim for controlling the lossiness that is generally done by other methods. So we go for Huffman which forms the basis of any technique being a good example of a lossless compression method. This project is mainly done to make a small attempt of bringing more compression to help the compression world that mainly go for images and texts reduction.

We have added an optimization part to say that the algorithm can have its extension. The formulation we have described in implementation chapter is generalized one. The sample input is compressed and decompressed by Huffman, Single pattern replacement model, Two-pattern replacement model sequentially to note the difference in the compression done. We see an improved performance in both new works than Huffman.They are the implemented in network to prove our result in stronger way that they could inturn make a reduction of time in getting transferred to the destination. Thus we conclude the preferable use of our new compression method for medical images and for network transfer.

FUTURE SCOPE

There can be numerous improvements made in this new work. The present work is not restricted to either of length 3 or the replacement of 2 best patterns. When the work gets extended there will be an improved compression ratio than Huffman.The areas of interest where future enhancements can be include

- Motion video files
- Sound files

It can be extended as:

- The future scope lies in increasing the length of the traced pattern to 4 or 5 including more options of finding the subaptterns.
- The n best patterns can be replaced at a time with single footer information. Incase of no such n- best patterns we can replace just (n-1) patterns. If there are no patterns we can follow normal Huffman method.
- Replacing patterns of different lengths can also be done if possible.
- The compression speed can be increased or maintained as Huffman in case of longer files.

REFERENCES

1. Nelson, M. and J.L. Gailly, 1995. The Data Compression Book , 2nd Edn., M and T Books, New York.
2. Bell, T., J. Cleary and I. Witten, 1990. Text Compression, Prentice-Hall, Englewood, USA.