Data compression through adaptive Huffman coding schemes

Conference Paper · February 2000

DOI: 10.1109/TENCON.2000.888730 · Source: IEEE Xplore

2 authors, including:

Muhammad Younus Javed
HITEC University, Taxila, Pakistan
298 PUBLICATIONS **2,026** CITATIONS

SEE PROFILE

DATA COMPRESSION THROUGH ADAPTIVE HUFFMAN CODING SCHEME

Dr. Muhammad Younus Javed and Mr. Abid Nadeem

Computer Engineering Department
College of Electrical and Mechanical Engineering
Peshawar Road, Rawalpindi – 46000
(National University of Sciences and Technology - PAKISTAN)
E-mail : myjaved@yahoo.com
Fax : 051-476190, 474306
Telephone : 051- 478783, 051-561-32966

ABSTRACT

A number of data compression techniques have been introduced to reduce the text/data storage and transmission costs. This paper describes the development of a data compression system that employs adaptive Huffman method for generating variable-length codes. Construction of the tree is discussed for gathering latest information about the entered message. The encoder process of the system encodes frequently occurring characters with shorter bit codes and infrequently appearing characters with longer bit codes. Adaptive, sibling, swapping, escape code, and re-scaling mechanisms of the model are briefly explained as they are extremely useful in enhancing compression efficiency. The decoder process expands the encoded text back to the original text and works very much like the encoder process. Experimental results are tabulated which demonstrate that the developed system is very effective for compressing database files (provides compression ratio up to 52.51%) in a real-time environment.

1. DATA COMPRESSION

In the last twenty years, large-scale information transfer through intranet and internet applications, the development of massive information storage/retrieval systems, and the use of software packages for text/data preparation have witnessed a tremendous growth. Concurrent with this growth, several problem areas such as huge size of databases and data transmission through communication lines have resulted in major economic expenditures. One method that can be employed to reduce data storage and information transfer costs is data compression. An efficient data compression technique generates codes by exploiting redundancy in such a way that the average number of coding digits per message is minimised. Thus, it is possible to reduce database file sizes considerably without any loss of information. This paper describes development of a Data Compression System (DCS) in which data reduction is achieved by encoding frequently occurring characters into shorter bit codes while representing infrequently occurring characters by longer bit codes. Adaptive Huffman coding has been used for dynamic data compression in order to decrease the average code length used to represent the symbols of character sets [1-3]. Pioneering work in the construction of Huffman tree [1] lead to the development of a number of compression systems due to its simplicity for producing minimum-redundancy codes. Huffman codes have a prefix property [4,5] which means that no short code group is duplicated, in the beginning of the longer group.

2. CONSTRUCTION OF THE ADAPTIVE HUFFMAN TREE

The Huffman coding creates variable-length codes that are an integral number of bits. Symbols with higher probabilities get shorter codes. In the DCS, Huffman codes are built by using bottom up approach, starting with the leaves of the tree and working progressively closer to the root. The individual symbols are laid out as a string of leaf nodes that are going to be connected by a binary tree. Each node has a weight which is simply the frequency or probability of the symbol's appearance. Five steps involved for the construction of the tree are: (1) the two free nodes with the lowest weights are located, (2) a parent node

0-7803-6355-8/00/ $10.00 © 2000 IEEE

is created which is assigned a weight equal to the sum of the two child nodes, (3) the parent node is added to the list of free nodes, and the two child nodes are removed from the list, (4) one of the child node is designated as the path taken from the parent node when decoding a 0 bit and the other is arbitrarily set to the 1 bit, and (5) steps 1 to 4 are repeated until only one free node, designated as the root of the tree, is left. More statistics are required to get better compression. Adaptive coding continuously updates the tree to gather latest information (i.e. frequencies of distribution) about the data/text entered so far. With this approach, both compressor and decompressor start off with identical models to encode and decode symbols. So when the compressor puts out its first encoded symbol the decompressor will be able to interpret it. After the compressor emits the first symbol, it proceeds to update the model so that it adapts to the changing behavior of the inputted data. Frequency of every existing character is incremented by one in order to generate shortest codes for the most frequently used characters. The Huffman tree exhibits sibling properly as its nodes can be listed in order of increasing weight and every node appears to its sibling in the list. This property helps in adaptive coding for updating the model. Thus, tree is always maintained in a proper Huffman format before and after the frequency counts are adjusted. The average number of increment operations required will correspond to the average number of bits needed to encode a character. Whenever, sibling property is violated due to frequency increments the concerned node is detached from its present position in the tree and is swapped with a node farther up the list. New characters are placed on the model as and when they are seen in the incoming message. Escape code is generated when a character appears in the first time. All the counts are rescaled as soon as the counter reaches to a maximum value to avoid overflow.

3. THE ENCODER PROCESS

After initializing the tree, compress routine of the DCS first identifies the leaf node for the symbol to be encoded. If the leaf node return a –1 value, it means that this symbol is not present on the tree. In this case the escape code is generated. In case the symbol is present on the tree, the encoder works by starting at the leaf node and moving up through the parent nodes one at a time until the root is reached. The data structure always groups the two children of a parent node next to one another in the node list. The odd child is assigned 1 and the even is always the 0. Given this information, each bit is added to the cummulative Huffman code starting from the leaf node. As each bit is encoded, the current bit mask is shifted by one so that the next bit encoded will appear in the next most significant position. A counter is also incremented that keeps tracks of how many bits have been accumulated in the output word so far. The code is sent out immediately after it is complete for the concerned symbol. For a new character, escape code is generated and the character is inserted in the tree with a weight value of 1. The most difficult part is to update the tree for keeping sibling property in tact. If the tree reaches to its maximum frequency count it is divided by 2.

4. THE DECODER PROCESS

This routine of the DCS decodes an incoming symbol. It starts at the root node of the tree and reads in a single bit at a time. As each bit is read in, one of the two children of the node is selected based on whether the input bit is a 1 or 0. Eventually, it reaches to the leaf node to decode a symbol. The only possible complication at this point is if the decoded symbol is an escape code. If so, the symbol encoded by the encoder did not yet appear in the Huffman tree. This means that the next eight bits in the input stream will contain an unencoded version of the symbol. In this case, plain-text version of the symbol is obtained. Either the decoded symbol or the escaped plain version of the symbol is placed in the output file. After decoding the symbol, the tree is updated to reflect the newly arrived symbol. The decoder process works very much like the encoder process. It also initializes the Huffman tree. Afterwards it continues decoding characters and updating the model until it reads an end of stream symbol. After incrementing the counter for the given character, the decoder has to work its way up through the parent nodes. Nodes have to be moved in their proper order for adhering to the sibling property. Rebuilding of tree and swapping of nodes works like the encoder. The decoder expands the encoded text back to the original text.

5. EVALUATION RESULTS

Performance of the DCS has been checked by measuring compression percentage. A file that remains unchanged when compressed will have zero percent compression whereas another file compressed down to one-third of its original size will have compression value of sixty-seven percent. Ten text samples of various forms and sizes were selected to obtain compression percentage. Each file was handed over to the encoder process, which was read character-by-character. The encoder constructed tree in accordance with the input text. The adaptive mechanism changed the tree to accommodate changing behavior of the text. Complete text/data of the file was compressed accordingly. The information

Table 1 Experimental Results

Text Samples	Nature of Text	Size Before Compression (KB)	Size After Compression (KB)	Compression Percentage
1	Pure text from book on "Data Processing Network"	1940	1120	42.27
2	Pure text from a book on "Digital Systems"	647	385	40.50
3	A complete chapter on "Data Compression Techniques" taken from an M.Sc. thesis	13043	7173	45.01
4	Source code of the DCS	15136	9419	37.78
5	Pay roll of employees	184567	87652	52.51
6	Petrol, Oil and Lubricant state of an organization	19904	10250	48.50
7	Weekly training programs of an engineering college	69266	33624	5146
8	Standing orders of an establishment	16256	10539	35.17
9	Program of courses for the year 1993/94 of an engineering college.	9625	5687	40.71
10	Technical evaluation report of a helicopter	33884	21878	35.43

displayed on the screen gave original size and compressed size of the file in kilobytes, and the value of compression percentage. Table 1 shows experimental results of all text samples. Text samples were practical in nature so they depicted routine documentation being used in the books/organizations. Text samples 4, 8 and 10 obtained compression ratio from 35.17% to 37.78% as they had minimum redundancy due to specific nature of text/data. Compression results ranging from 40.50% to 42.27% were achieved for text samples 1, 2 and 9. These samples contained real text and had natural redundancy. Sample 3 contains information about various text compression strategies and it provided 45.01 % compression. This is an improvement because information on the same topic had many common words and terminologies. Therefore, the Huffman tree generated the shortest codes for the most frequently used text. Sample 6 was a mixture of text and numerical data pertaining to stock of petrol, diesel and lubricants held in an establishment. State contained information about allotted, drawn, consumed and balance of each item. It gave compression of 48.50% due to shorter average code length as only digits were changing for the limited entries. Compression ratios of 52.51% and 51.46% were yielded for text samples 5 and 7 respectively. These two samples contained repetition of words like rank, basic pay and allowances, credits, debits, days of the week, names of the subjects, names of tools. So a lot of redundancy was present in the text/data which was effectively exploited through the adaptive Huffman tree by producing minimum-redundancy codes. This compression was very encouraging and it proves the fact that more redundant information gives better compression percentage.

6. CONCLUSIONS

Data compression is of great interest in data processing because it offers cost savings and the potential for increased capacity in mass storage devices, channels and communication lines. For this purpose, a Data Compression System (DCS) has been developed which employs adaptive Huffman coding for generating minimum-redundancy codes. The encoder process compresses the source text character by character. The adaptive mechanism automatically updates the tree in accordance with changing behavior of the text/data. It encodes characters using variable-length codes. Shorter bit codes are assigned to the frequently used characters whereas infrequently occurring characters are represented by longer bit

codes. DCS properly decodes the coded text because of prefix property. Evaluation results were obtained by running a number of text samples of varying size and redundancy. DCS has provided compression ratio upto 52.51% due to effective features of the model such as sibling, swapping and re-scaling. These results are very promising and indicate that DCS is well suited for on-line encoding/decoding in data networks and compression for larger files in order to reduce storage, transmission, rental and maintenance costs.

7. REFERENCES

[1] Huffman, D.A., A Method for the construction of Minimum Redundancy Codes, Proceedings of the IRE, 1952, pp. 1098-1101.

[2] Ferguson, T.J. and Rabinowitz J.H., Self–Synchronizing Huffman codes, IEEE Transactions on Information theory, Vol. IT-30, No. 4, 1984, pp. 687-693.

[3] Vitter, S.V., Design and Analysis of Dynamic Huffman Codes, Journal of the Association for Computing Machinery, Vol. 34, No. 4, 1987, pp. 825-845.

[4] Hirschberg, D.S. and Lelewer, D.A., Efficient Decoding of Prefix codes, Communication of the ACM, 1990, pp. 449-458.

[5] Nadeem, A., Design and Implementation of Data Compression System, College of Electrical and Mechanical Engineering, Rawalpindi, MSc. Thesis, 1995.