An Adaptive Multiple Order Context Huffman Compression Algorithm Based on Markov Model

Yonghua Huo[1(✉)], Zhihao Wang[1], Junfang Wang[1], Kaiyang Qu[2], and Yang Yang[2]

[1] Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory, 54th Research Institute of China Electronics Technology Group Corporation, Shijiazhuang, China
tsdhyh2005@163.com, {cetc540016,jfwang2015}@sina.com
[2] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China
1445154975@qq.com, echo_lzjf@163.com

Abstract. In this paper, an adaptive multiple order context Huffman compression algorithm based on Markov chain is proposed. Firstly, the data to be compressed is traversed, and the character space of the data and the times that one character transfers to its neighboring character are figured out. According to the statistical results, we can calculate the one-step transition probability matrix and the multi-step transition probability matrix. When the conditional probability between two adjacent characters is greater than the set threshold value, the adjacent characters are merged and compressed as an independent encoding unit. Improve the compression efficiency by increasing the length of the compression characters. The experimental results show that the algorithm achieves good compression efficiency.

Keywords: Data compression · Multiple order contexts · Markov chain · Huffman compression

1 Introduction

With the rapid development of information technology, the increasing amount of data in time and space brings great challenges to the storage of information. In this case, the data compression is particularly important, and is also a long-term concern in the field of data management.

Huffman algorithm [1] is a kind of compression algorithm with a fine effect. Huffman algorithm calculates the probability of each character to give the corresponding weight to each character, and constructs a Huffman tree, in which shorter higher probability

This work was supported by Open Subject Funds of Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory (ITD-U15002/KX152600011). NSFC (61401033, 61372108, 61272515). National Science and Technology Pillar Program Project (2015BAI11B01).

S. Wang and A. Zhou (Eds.): CollaborateCom 2016, LNICST 201, pp. 83–93, 2017.
DOI: 10.1007/978-3-319-59288-6_8

distribution of characters with high probability occurring are assigned short code, and encoding, of characters with low probability occurring are assigned longer code. The Huffman algorithm achieves good compression effect by reducing average code length of single character.

There is a certain relation between characters in the actual text environment. For example, a character sequence as 'abcabdefgabg', it is observed that the 'b' character is always behind the 'a' character, just like the 'man' always behind the 'Huff'. Therefore, this paper proposes a compression algorithm based on Markov chain and Huffman algorithm. Each character is regarded as a state in the Markov chain, and according to the jump numbers between characters calculates one step transition probability matrix and multi-step transition probability matrix.

Different threshold for different characters, when the transition probability between characters is greater than the threshold value, characters should be combined, otherwise, they are compressed separately as independent characters. When the *N* order context does not meet the conditions of the combined compression, it is automatically adjusted to the *N-1* order context compression, and the worst case is the 0-order context compression, that is, Huffman compression algorithm.

In the paper, we briefly discuss related work in Sect. 2. An adaptive Huffman compression model is proposed in Sect. 3, and Sect. 4 is the experimental results and comparative analysis. Section 5 concludes the paper.

2 Related Works

There are related works including Huffman compression algorithm [1], arithmetic coding [2], dictionary coding [3] and other compression algorithms popular at present. Arithmetic coding is a compression scheme which recodes based on the statistical results of the occurrence probability of characters. Arithmetic coding bypasses the idea of replacing an input symbol with specific code. It replaces the sequences of symbols in the input file with single arithmetic (float) number. More longer input messages, more bits are needed in the arithmetic number [4]. Output from an arithmetic coding process is a single number less than 1 and greater than or equal to 0. This single number can be uniquely decoded to create the exact stream of symbols. To construct the output number, the symbols are assigned a set of probabilities.

Dictionary coding is similar to the way of looking up the dictionary. Its basic principle is considering long strings or combination of characters with high probability occurring as notes in the dictionary, and using short number or symbols to represent these notes [3]. Dictionary-based code compression is commonly used in embedded systems [5], because it can achieve an efficient CR, possess a relatively simple decoding hardware, and provide a higher decompression bandwidth than the code compression by applying lossless data compression methods. The compression of dictionary coding is determined on the reoccurring data.

Huffman algorithm [4] has been proven the best fixed-length coding method available [6], which reduces average code length for a single character. If the probability of a symbol's appearance in a message is known, Huffman techniques can encode that

symbol using a minimal number of bits. A Huffman code is an optimal prefix code that guarantees unique decidability of a file compressed [7]. The code was devised by Huffman as part of a course assignment. It uses a code tree and has uniquely decodable code that is a proper prefix of any other code word.

But Huffman code does not consider the relationship between the characters [7]. In [8], a multiple subgroup data compression technique based on Huffman coding is proposed. This technique extends the work on Entropy calculation by reducing the codeword length of the characters. In [9], a quasi-lossless compression algorithm was proposed based on the analysis and comparison of performance between Huffman coding and arithmetic coding. Predictive coding and Huffman coding were organically combined. Furthermore, this algorithm removed redundant information through compression preprocessing and secondary quantization.

3 Adaptive Multiple Order Context Huffman Compression Algorithm Based on Markov Model

3.1 Definition and Symbol

In a sequence of characters, for example "abcdefg……z", the distance between character 'a' and 'b' is one step, so we call "ab" one-order context. Similarly, the distance between 'a' and 'c' 'a' and 'd' and 'a' and 'z' are two steps, three steps and n steps, thus the relationship between them are two-order context, three-order context and n-order context respectively. If the number of steps is greater than two, we call it multiple order.

Suppose that the text to be compressed is T, and V is the character space of T. $V = \{v_1, v_2, v_3, \ldots\ldots v_n\}$, v_i is i-th character of V. C, a two-order matrix, C_i represents the appearing times of v_i and C_{ij} represents the appearing times of v_j behind v_i. According to the C_{ij}, the one-step transition probability $P^{(1)}$ can be calculated. Further, we can get multiple transition probability matrix $P^{(n)}$.

Based on the n-step transition probability matrix, we define a set of Statistics $p = \{p_1, p_2, \ldots\ldots p_{2n-1}\}$ to stand for the relationship between $n + 1$ states. For example, if the n is three, the $p = \{p_1, p2, p3, p4, p5\}$. For four characters (states), $T_0, T1, T2$ and T_3, $p_1(p_1 = p(X_{n+1} = T_1 \mid X_n = T_0))$ represents the one-step transition probability from T_0 to T_1, that is the conditional probability of T_1 appearing when the T_0 appears. p_2 represents the one-step transition probability from T_1 to T_2, p_3 represents the one-step transition probability from T_2 to T_3; $p_4(p_4 = p(X_{n+2} = T_2 \mid X_n = T_0))$ represents the two-step transition probability from T_0 to T_2, that is the probability of the situation that T_2 is the second character after T_0; $p_5(p_5 = p(X_{n+3} = T_3 \mid X_n = T_0))$ represents the three-step transition probability from T_0 to T_3. The state transition is as following (Fig. 1):

If p_1 is greater than the threshold *TEMP*, we consider the T_0 and T_1 are relevant. If p_2 is greater than *TEMP*, then the T_1 and T_2 are also relevant. T_0 and T_1 are relevant, T_1 and T_2 are relevant don't stand for the relation between T_0 and T_2. Only when p_3 is greater than the *TEMP*, the T_0, T_1 and T_2 are relevant. The value of *TEMP* will be discussed

later. Equally, the T_0, T_1, T_2 and T_3 are relevant if all the p_1, p_2, p_3, p_4 and p_5 meet the condition.

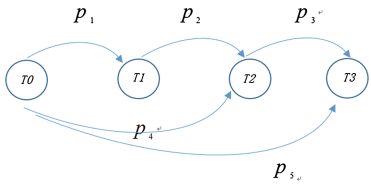

Fig. 1. State transition

3.2 Steps of Algorithm

In this paper, we set the number of context order to three. The method of n-order context is also same. How the number of context order influences the compression results will be discussed.

(1) First go through the text T, get character collection of T, $V = \{v_1, v_2, v_3 \cdots v_n\}$. Compute the occurring times of $v_i (1 \leq i \leq n)$ as C_i, and C_{ij} is the times that v_j is behind v_i. A two-order matrix C made up with C_{ij} is as follow:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & c_{ij} & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} \tag{1}$$

(2) According to the model of Markov chain, we consider $v_i (1 \leq i \leq n)$ as a state, and the one-step transition probability matrix $P^{(1)}$ can be got from (1):

$$P^{(1)} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & p_{ij} & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix} \tag{2}$$

and

$$P_{ij} = P(X_{n+1} = v_j \mid X_n = v_i) = c_{ij} / \sum_{k=1}^{n} c_{ik} \tag{3}$$

P_{ij} is the appearing probability of j, when i occurs. According the one-step transition probability matrix, two-step transition probability matrix:

$$P^{(2)} = P^{(1)} * P^{(1)} \tag{4}$$

Three-step transition probability matrix:

$$P^{(3)} = P^{(2)} * P^{(1)} \tag{5}$$

Four-step transition probability matrix:

$$P^{(4)} = P^{(3)} * P^{(1)} \tag{6}$$

(3) Read the four character T_0, $T1$, $T2$ and T_3 in T, according to the step (2), we can get a five tuples $p = \{p_1, p_2, p_3, p_4, p_5\}$ about T_0, $T1$, $T2$ and T_3 to show the relationship among them. The detail of judging the relevance of these four characters is described by the following Algorithm 1.

Algorithm 1: **The Algorithm for relation**

Input: The sequence of four characters, T_0, T_1, T_2 and T_3 and the $p = \{p_1, p2, p3, p4, p5\}$

Output: The Independent coding unit that to be compressed.

```
if(p1>=TEMP) {
    there is a relationship between T0 and T1;
    if(p2>=TEMP){
        there is a relationship between T1 and T2;
        if(p3>=TEMP){
            there is a relationship between T2 and T3;
            if(p4>=TEMP && p5>=TEMP){
                T0、T1、T2 and T3 can be merged and compressed.
            else if(p4>=TEMP) {
                T0、T1、T2 can be merged and compressed.
            else{
                (T0 and T1) or (T1 and T2) or (T2、T3) are merged and
                compressed.
            } else{
                (T0 and T1) or (T1 and T2) are merged and compressed. }
            } else{
                T0 and T1 are merged and compressed. }
            }else{
                The T0 compressed separately, and read the following four
                characters in T, repeat the process above.
}
```

TEMP is the threshold which will influence the compression results. When one-step transition probability from character T0 to T1 $p_1 >= TEMP$, we think that there is relationship between T_0 and T_1, which should be merged and compressed. What's more, if $p_2 >= TEMP$ and $p_3 >= TEMP$, T_0, T_1 and T_2 are relevant and should be merged. If all the probabilities p_1, p_2, p_3, p_4 and p_5 satisfy the condition, then the T_0, T_1, T_2 and T_3 are relevant. Otherwise, the max related string is "$T_0T_1T_2$", which doesn't include "$T3$". If

the *n*-order context doesn't contain the max related string, we will try to find the max related string in the *(n-1)*-order until we find the max related string. At last, the max related string is output as an independent coding unit.

(4) The occurring frequency of each individual coding unit is calculated as their weight, e.g. $\{W_1, W_2, \ldots W_i, \ldots W_n\}$ create n binary trees, and the set of these binary trees is $F = \{T_1, T_2, .. T_i, \ldots T_n\}$. each binary tree T_i in the set only has a root node with weight value w_i and its left and right subtrees are empty (In order to facilitate the implementation of algorithm in computer, T_i are also required to be in ascending order as the value of W_i).
(5) select two trees with the least weight of root node as the left and right subtree of a new binary tree, whose weight of root node is the sum of the subtrees' weights.
(6) Delete the two trees with the least weight, and the new tree should be inserted in the set F in ascending order.
(7) Repeat the two steps (5) and (6), until there is only one tree in the set F.

Construct the Markov chain model based on the statistical results of text T in step (1) and (2), and then get the n-step transition probability matrix; The step (3) is the main part of this algorithm, which search for independent compression units according to the transition probability matrix, in order to increase the length of independent encoding units and realize the automatic adjustment between the multiple order context; calculate the occurrence probability of independent encoding units according to their occurrence times and give them corresponding weights. Construct the Huffman tree and output compression code according to the Huffman algorithm.

3.3 The Confirmation of Threshold

The value of *TEMP* will influence the results of the compression. So, a proper value is very important.

According to the statistics of the text, the transfer one-step transfer distribution of the text (based on letter a, b, c, d) is as following.

Two-step and three-step transfer distribution are same as above. Ordinate represents the occurrence times of one-step transition, horizontal ordinate represents the adjacent characters, and each number is a corresponding character. Corresponding relationship is shown in Table 1 as following.

Table 1. Corresponding relationship

Integer value	Corresponding character
0–25	a–z
26–51	A–Z
52–61	0–9
62	space
63	.

As Fig. 2-1, the circumstance that letter 'a' behind 'a' occurs 0 times, the circumstance that 'b' behind 'a' occurs 6 times, and the letter 'c' is 5 times, 'd' is 14 times. As shown in the Fig. 2-1, the first 25 characters are more active, occurring times of the rest are almost 1. In order to increase the length of the compression coding and also improve the compression speed, we define the *n*-order *TEMP* of character 'a' as following:

$$TEMP = \frac{p_{i0}^{(n)} + p_{i1}^{(n)} + p_{i2}^{(n)} + \cdots + p_{i25}^{(n)}}{25} \tag{7}$$

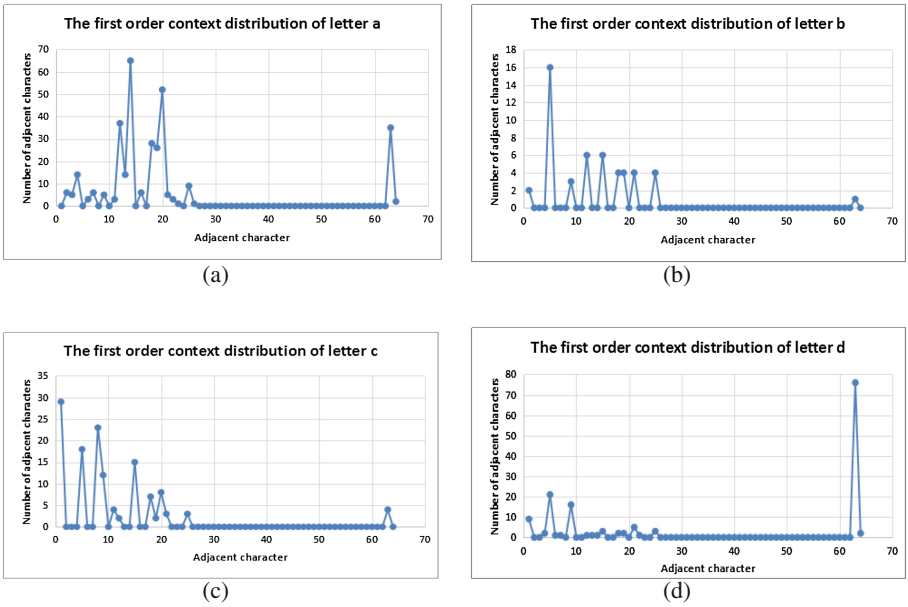

Fig. 2. 1. The first order context distribution of the letter a. 2. The first order context distribution of the letter b. 3. The first order context distribution of the letter c. 4. The first order context distribution of the letter d

4 Experimental Results and Analysis

4.1 The Experimental Environment

Intel(R) Core(TM) i3-4010U 4 CPU 1.70 GHz, memory 4 GB,cache L1:128 KB,cache L2:512 KB, cache L3:3.0 MB. Operating system is Windows 8,the development environment is DEV-C++. We realize the multiple order context Huffman compression algorithm based on Markov chain model by C language, and compare this algorithm with Huffman compression algorithm [12] and adaptive Huffman [13] compression algorithm.

The compression test is carried out on (20 KB, 200 KB, 500 KB, 1000 KB) four kinds of text data of different sizes.

4.2 Experimental Results and Analysis

This paper compares the compression algorithm Huffman compression algorithm, adaptive Huffman compression algorithm and the multiple order context Huffman compression algorithm based on Markov model, from two aspects:

Comparison of Compression ratio: Comparison of different amount of data on the same type of data (the string type). (Note: compression ratio = amount of data after compression/amount of data before compression)

Figures 3, 4, 5 and 6 are the comparison result of one-order, two-order, three-order and four-order Markcov chain compression algorithm with the other two compression algorithm respectively. Figure 7 is the comparison result of different order.

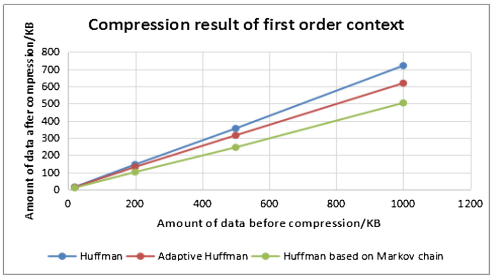

Fig. 3. Compression result of one-order context

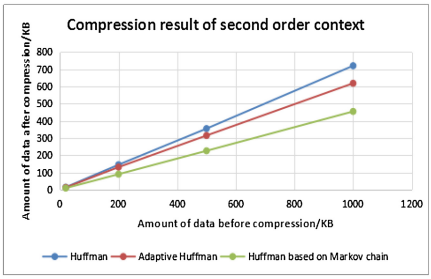

Fig. 4. Compression result of two-order context

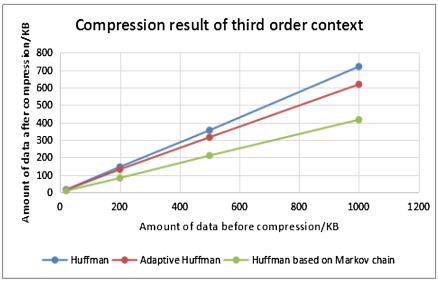

Fig. 5. Compression result of three-order context

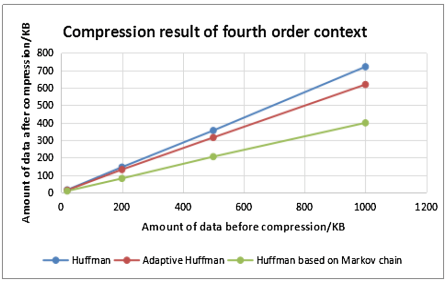

Fig. 6. Compression result of four-order context

As shown in Fig. 3, the compression effect of the Huffman algorithm based on Markov chain is better than the other two algorithms in the case of using one-order context compression. The compression effect using two-order context compression is shown in Fig. 4. We can see the advantage of Huffman algorithm based on Markov chain is more obvious. When the amount of data reaches 1000 KB, compression ratio of the Huffman algorithm based on Markov chain is 45%, the other algorithm are 72% and 62% respectively.

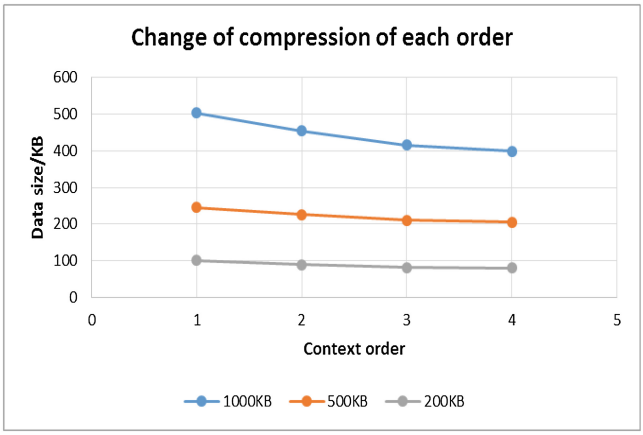

Fig. 7. Change of compression of each order

The three-order case is shown in Fig. 5, and the compression algorithm proposed in this paper has a good compression effect. The compression rate can reach 42%. We can see that compared to the other two algorithms, the advantage of Huffman algorithm based on Markov chain is more and more obvious in the four-order case.

As shown in above, multiple order context compression algorithm based on Markov chain is better than the other two compression algorithms. This is because the Huffman compression algorithm and the adaptive Huffman compression algorithm only take each character as an independent coding unit without taking into account the link between characters. The algorithm proposed in this paper considers the link between the characters, and makes the letters merged and compressed. But with the increase of the compression order, the compression rate is stable. Comparison of the compression ratio in different order of context is shown in Fig. 8 for the algorithm proposed in this paper.

Fig. 8. Comparison of the compression ratio

As can be seen in Fig. 8, with the increase of the order of context, the compression ratio is gradually reduced, and the compression effect becomes better. But when the order is more than three, the compression rate becomes stable. This is because with the increase of the context order, the relation between characters becomes weak. So the optimal context compression order should be three or four.

5 Conclusion

Through the analysis of Sect. 4, we can see Huffman compression algorithm based on Markov chain is superior to the traditional Huffman compression algorithm and the adaptive Huffman compression algorithm. By constructing the Markov model and calculating the transfer probability matrix, characters are merged and compressed, which gets fine effect. The performance of the compression algorithm proposed in this paper is good, but we use threshold (e.g. *TEMP*), the value of which is also an important factor affecting the results of the experiment. These thresholds need to be further tested to achieve better compression results.

References

1. Kuruvila, M., Gopinath, D.P.: Entropy of Malayalam language and text compression using Huffman coding. In: First International Conference on Computational Systems and Communications. IEEE, pp. 150–155 (2014)
2. Wu, J., Dai, W., Xiong, H.: Regional context model and dynamic Huffman binarization for adaptive entropy coding of multimedia. In: IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, pp. 1–6. IEEE (2014)
3. Wang, Z., Le, J.J., Wang, M., et al.: The column storage district level data compression mode and compression strategy selection method. In: ndbc2010 National Database Conference of China, pp. 523–1530 (2010)
4. Darwiyanto, E., Pratama, H.A., Septiana, G.: Text data compression for mobile phone using burrows-wheeler transform, move-to-front code and arithmetic coding (Case Study: Sunan Ibnu Majah Bahasa Translation). In: International Conference on Information and Communication Technology, pp. 178–183. IEEE (2015)
5. Wang, W.J., Lin, C.H.: Code compression for embedded systems using separated dictionaries. IEEE Trans. Very Large Scale Integr. Syst. **24**, 1 (2015)
6. Yokoo, H.: An adaptive data compression method based on context sorting. In: Data Compression Conference, pp. 160–169. IEEE (1996)
7. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inform. Theory **23**(3), 337–343 (1977)
8. Ren, W., Wang, H., Xu, L., et al.: Research on a quasi-lossless compression algorithm based on huffman coding. In: 2011 International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE), pp. 1729–1732. IEEE (2011)
9. Ong, G.H., Ng, J.P.: Dynamic Markov compression using a crossbar-like tree initial structure for chinese texts. In: International Conference on Information Technology and Applications, pp. 407–410. IEEE (2005)

10. Wei, J., Wang, S., Zhang, L., et al.: Minimizing data transmission latency by bipartite graph in MapReduce. In: IEEE International Conference on CLUSTER Computing, pp. 521–522. IEEE (2015)
11. Papamichalis, P.E.: Markov-Huffman coding of LPC parameters. IEEE Trans. Acoust. Speech Signal Proc. **33**(2), 451–453 (1985)
12. Nandi, U., Mandal, J.K.: Adaptive region based huffman compression technique with selective code interchanging. In: Advances in Computing and Information Technology, pp. 739–748 (2012)
13. Singh, S., Singh, H.: Improved adaptive huffman compression algorithm. Int. J. Comput. Technol. **1**(1), 1–6 (2011)