動態霍夫曼編碼之計算時間與空間關係之探討

On the time v.s. space complexity of Adaptive Huffman coding

計畫編號：NSC 882815C260002E
執行期限：八十七年十一月一日至八十八年六月三十日

游逸平
Yi-Ping You

國立暨南國際大學資訊工程學系
Department of Computer Science and Information Engineering
National Chi-Nan University
E-mail: u5321009 @ncnu.edu.tw

蔡錫鈞
Shi-Chun Tsai

國立暨南國際大學資訊工程學系
Department of Computer Science and Information Engineering
National Chi-Nan University
E-mail: tsai@csie.ncnu.edu.tw

摘要

霍夫曼編碼法是一種有效且廣受使用的資料壓縮技術。在本文中，我們將重點著眼在其編碼時所需計算時間與空間的關係，並且提出緩衝式動態霍夫曼編碼改良傳統的編碼方式，其運作方式是在編碼與解碼的同時動態的改變霍夫曼樹的結構，不同於傳統的動態霍夫曼編碼方式，我們並不是每讀取一個字元就更新樹狀結構，只有在更新點時才做修正的動作，這可使得即時編碼更快速、有用。另外，我們也提出動態多元樹霍夫曼碼法。

關鍵詞：霍夫曼編碼，緩衝式動態編碼法 動態多元樹編碼法

Abstract

Huffman codes are a widely used and very effective technique for compressing data. In this paper, we focus on the relationship between the computing time and space that is needed when compressing data with Huffman codes. We propose a further improvement to the Huffman method, called *buffered adaptive Huffman coding*. This approach dynamically changes the structure of Huffman code trees when encoding and decoding. Unlike original adaptive Huffman coding, the time when to update the tree is adjusted to not change the tree every time we read a symbol. It is changed only when the updating point is reached and it will save the number of updating the adaptive Huffman tree. The scheme is fast and useful to on-line encoding. We also propose a method of nonbinary Huffman coding based on adaptive encoding, called *m-ary adaptive Huffman coding*.

Keywords: Huffman codes, buffered adaptive Huffman coding, updating point, *m*-ary adaptive Huffman coding

1. Introduction

As the amount of information that is needed, desired, and available increases, the need for more efficient ways of representing formation increases as well. The goal of data compression is to provide the most efficient way to represent information. Huffman coding [4] is one of the lossless compression techniques. Since David Huffman invented

the scheme in 1952, Huffman code has been widely used in a variety of forms, such as text, images, speech, video, and so on.

The basic idea of Huffman coding is to produce a binary tree and encode a symbol, which is a leaf of the tree, to a binary string with the sequence of edge labels on the path from the root to the symbol. One disadvantage of Huffman's algorithm is that it makes a two-pass procedure over the data: the frequency counts are collected in the first pass, and the data is encoded in the second pass. Moreover, the method cannot encode data on-line to cater the network communication with massive data of multimedia today.

In order to convert this algorithm into a one-pass procedure, Faller [2] and Gallagher [3] independently developed algorithms for adaptively developing the Huffman code based on the statistics of the symbols already encountered, latter improved by Knuth [5] and Vitter [8]. In the present paper we propose a new approach on adaptive Huffman coding by adding a buffer. By using the buffer, the time of encoding and decoding is faster than the scheme proposed by Vitter. It is described in the following section. In section 3, we propose a method of nonbinary and adaptive Huffman coding. The algorithm is similar to that of adaptive Huffman coding.

2. Buffered adaptive Huffman coding

Adaptive Huffman coding [6][7] is a technique for on-line compressing data that requires only one-pass over the data. That is, adaptive Huffman codes encode the next character with the current tree and then rebuild the tree to be optimal for all characters seen thus far. The scheme follows the original Huffman's algorithm and modifies the structure of binary tree adaptively to maintain the property of Huffman tree.

To describe how the adaptive Huffman code works, we add two other parameters to the binary tree: the weight of each node and a node number. The weight of each external node is simply the number of times the symbol has been encountered. And that of each internal node is the sum of the weights of its children. The node number is a unique number assigned as the order, $2n-1, 2n-2, 2n-3, \ldots$, where n is the size of alphabet. At the start of transmission, the tree at both the transmitter and the receiver consists of a single node that corresponds to all symbols not yet transmitted (NYT) which has a weight of zero. Before the beginning of transmission, a short fixed code for each symbol is agreed upon between transmitter and receiver. Once a symbol is encountered for the first time, the code for the NYT node is transmitted followed by the fixed code for the symbol, and then taken out of the NYT list. Suppose that the source has an alphabet $(a_1, a_2, \ldots, a_m)$ of size m. We pick e and r such that $m = 2^e + r$ and $0 \leqq r < 2^e$. Then a letter a_k is encoded to $a_k.enc$ as follows:

If $1 \leqq k \leqq 2r$, **then**

$a_k.enc$ = $(e + 1)$-bit binary representation of $k-1$

Else

$a_k.enc$ = e-bit binary representation of $k - r - 1$.

The original adaptive Huffman coding says that we should update the Huffman tree once a symbol is read. It is very inefficient since calling a procedure is expensive and the result of update may be the same before changing. To make the adaptive Huffman coding more useful and effective, we propose the buffered adaptive Huffman coding by adding a new parameter, called buffer. The buffer is added to record the frequency of each symbol that has been encountered. Every time a symbol is read, the frequency of the symbol is increased by 1. We update the tree only when the number of frequency of a symbol reaches the number B, which is defined as *updating point*. In this way, the times of calling procedure of update are reduced. Moreover, the period of encoding and decoding are both shortened. This approach speedups the coding time within only a small buffer, i.e. we trade some space for time. The algorithm of buffered adaptive

Huffman coding is as follows:

Encoding( )
Set the frequency of each symbol to zero.
While the symbol read is not the last one **do**
  Increase the frequency of the symbol by 1.
  If the frequency of the symbol is 1 **then**
    Send code for NYT node followed by index in the NYT list.
  Else
    Code is the path from the root node to the corresponding node.
    If the frequency of the symbol **mod** *B* is not zero **then**
      Continue the loop.
  Call *update* procedure.
End do

Decoding( )
Set the frequency of each symbol to zero.
While the bit read is not the last one **do**
  Go to root of the tree.
  While the node is not an external node **do**
    Read bit and go to corresponding node.
  End do
  If the node is the NYT node **then**
    Read *e* bits.
    If the *e*-bit number *p* less than *r* **then**
      Read one more bit.
    Else
      Add one to *p*.
    Decode the (*p* + 1) element in NYT list.
    Increase the frequency of the symbol that is decoded by 1.
  Else
    Decode element corresponding to node.
    Increase the frequency of the symbol that is decoded by 1.
    If the frequency of the symbol **mod** *B* is not zero **then**
      Continue the loop.
  Call *update* procedure.
End do

Update( )
If symbol appear at first time **then**
  NYT gives birth to a new NYT and an external node.
  Increment weight of external node and old NYT node.
  Go to old NYT node.
  While the node is not the root node **do**
    Go to parent node.
    If the node number is not the maximum in block **then**
      Switch node with highest numbered node in block.
    Increment node weight.
  End do
Else
  Go to symbol external node.
  While true **do**
    If the node number is not the maximum in block **then**
      Switch node with highest numbered node in *block*.
    Increment node weight.
    If the node is not the root node **then**
      Go to parent node.
    Else
      Break the loop.
  End do

---

* The set of nodes with the same weight makes up a *block*.

3. *M*-ary adaptive Huffman coding

The binary adaptive Huffman coding can easily be extended to the nonbinary case where the code elements come from an *m*-ary alphabet, and *m* is not equal to two. The approach of the scheme is almost in the same way. The major difference is in the encoding and decoding procedures. In both procedures, we check if the number of symbol first appeared reach the value of $m - 1$ before calling the update procedure. The check ensures the growth of tree rise *m* nodes every time. The algorithms of encoding and decoding procedures are as follows:

Encoding( )
Set *N* to zero.
While the symbol read is not the last one **do**
  While *N* is not equal to $m - 1$ **do**
    If this is the first appearance of the symbol **then**
      Send code for NYT node followed by index in the NYT list.
      Increase *N*.
      Continue the loop.
    Else

```
        Code is the path from the root node to
        the corresponding node.
        Break the loop.
    End do
    Call update procedure.
    If N is equal to m – 1 then
      Set N to zero.
End do

Decoding( )
While the bit read is not the last one do
    Go to root of the tree.
    While the node is not an external node do
      Read bit and go to corresponding node.
    End do
    Set N to zero.
    While N is not equal to m – 1 do
      If the node is the NYT node then
        Read e bits.
        If the e-bit number p less than r then
          Read one more bit.
        Else
          Add one to p.
        Decode the (p + 1) element in NYT list.
        Increase N.
        Continue the loop.
      Else
        Decode element corresponding to node.
        Break the loop.
    End do
    Call update procedure.
    If N is equal to m – 1 then
      Set N to zero.
End do
```

In this way, the depth of *m*-ary adaptive Huffman tree will be reduced, since the growth of tree rise on width of it partly. This means that codewords of symbols are reduced. Hence, the period of tracing path is shorter than binary scheme when decoding.

4. Conclusion

The efficiency of the new method of buffered adaptive Huffman coding depends on the judicious choice of the size of buffer. Too large or too small size would cause needless waste on transmission. Experiments are needed to decide the size for different environment. In the m-ary adaptive Huffman coding, the efficiency depends on the value of *m*. The more the value is, the faster it will be when decoding.

Reference:

[1] K.L. Chung. *Efficient Huffman decoding*, Information Processing Letters 61 (1997) 97-99

[2] N. Faller. An Adaptive System for Data Compression. In *Record of the 7th Asilomar Conference on Circuits, Systems, and Computers*, pages 593-597. Piscataway, NJ: IEEE Press, 1973.

[3] R.G. Gallagher. Variations on a Theme by Huffman. *IEEE Transactions on Information Theory*, IT-24(6):668-674, November 1978.

[4] D.A. Huffman. *A method for the construction of minimum redundancy codes.* In Proc. IRE 40 (1951), 1098-1101.

[5] D.E. Knuth. Dynamic Huffman Coding. *Journal of Algorithms*, 6:163-180, 1985.

[6] X. Lin. *Dynamic Huffman Code for Image Compression*. MS thesis, University of Nebraska, 1991.

[7] K. Sayood. *Introduction to data compression*, pages 43-50.

[8] J.S. Vitter. Design and Analysis of Dynamic Huffman Codes. *Jounal of ACM*, 34(4):835-845, October 1987.