Analysis of Common Lossless Coding Algorithm

Xue-feng Cao
College of Mathematics and Information Science, Huanggang Normal University, Huanggang, Hubei 438000
e-mail caoxuefeng852@hgnu.edu.cn

Abstract: The paper gives out the basic principle and algorithm for lossless coding, conducts mathematical analysis for its time complexity, code length, and code rate, etc, and anticipates existing optimum coding principle possible for coding function image and variable length coding. Meanwhile, it proposed coding suggestions for basic lossless coding to make it play maximal role in the aspect of basic teaching research, and so on.

Keywords: coding algorithm; time complexity; code length; code utilization ratio; code image; optimal coding principle

1 Introduction

The fundamental issue of communication is how to replicate the output information of information sources in the information sink of the receiving terminal exactly or approximately; how to transmit information of information sources with the least signs under lossless condition or allow certain distortion condition to improve information transmission rate in the process of the communication; and how to increase anti-interference capability of the signs and get the maximum information transmission rate under the condition that the information source is interrupted. If the information sink of the receiving terminal requires replicating the output information of the information sources exactly without distortion, this representation is one-to-one correspondence, that is, ensure the entire information produced by the information source transmitted to the information sink without distortion, the code of information source at this time is lossless coding. If the information sink of the receiving terminal doesn’t require that the output information of the information sources to be recovered exactly, and allows the occurrence of certain distortion which conforms to most realization situation, the code of information source at this time lossy coding. As to certain information sources and information sink, the acceptable standard with information sink shall be confirmed firstly, which is called as fidelity criterion, and will be taken as the basis for incomplete accurate replication.

Coding is a kind of transformation conducted for original signs of information source in accordance with certain mathematical rules in essence. For the convenience of analysis and highlight emphasis of the problem, consider the coding and highlight emphasis of the problem, consider the coding and decoding of information sources as a part of the information sources, and highlight the code of information sources while study the code of information sources. Likewise, consider the coding and decoding of information sources as a part of information sources and information sink, and highlight the code of information sources while study the coding of information sources. The coding algorithm for lossless information sources, which is an important aspect of coding, has important function for basic research and engineer application. Simple analysis is made in the following for some common lossless coding algorithms:

2 Simple Analysis for Some Common Lossless Coding Algorithms

2.1 Shannon Coding

Coding principle: As for the summation of sequence of number, the code length of any probability can be confirmed by probability and unicity. As q(k)=p(i), the code length can be confirmed as l(k)=[-logq(k)], and the truncation for probability and transformed codeword can obtain unique code.

Algorithm realization:

```c
int biggerfun( double x)
{
    return (int)(x+1)?(int)(x):(int)(x)<x; /* return to minimum integer which is not smaller than x*/
}
void Shannon Coding(double  p[N])
{
    int i;
    double sum1[N];
```
Coding characteristics of Shannon coding:
Under the condition that the difference of probability value is significant, the efficiency of Shannon coding is not very good.

Time complexity $\sum_i (1 + 4 \sum_j + 1)$ conducts homogenizing treatment for $j$ in $[1, l[i]]$ state, then the time complexity is $O(n \log n)$. The average code length can then be derived through mathematical knowledge: $L < H(S) / \log r + 1$.

### 2.2 Fano Recursive Coding

Coding principle: classify the information into two groups of probability and similar group, and assign “0” and “1” for them respectively, then the information code can be confirmed uniquely.

Algorithm realization:

/*Construct fano code*/
void fanofun(double p[N], int low, int high) {
    double sumf[N], sume[N], t;
    int i0 = low, j0 = high;
    int v1 = 0, v2 = 0, k;
    char str[N][MaxSize];
    t = fabs(sumf[i0] - sume[j0]);
    if(t < p[i0]) {
        /*classify the information into two groups of probability and similar group*/
        if(sumf[i0] < sume[j0]) /*set 0 for former group*/ {
            sumf[i0] += p[i0];
            str[i0][v1] = '0';
            i0++;
        }
        else {
            sume[j0] += p[j0]; /*set 1 for later group*/
            str[j0][v2] = '1';
            j0--;
        }
        k = i0;
    }
    else {
        j0--;
        sume[j0] += p[j0];
        v1++; v2++;
        i0 = low, j0 = high;
        fanofun(p[k], i0, k); /*conduct fano code for the remaining two recursive components*/
        fanofun(p[N-k], k, j0);
    }
    /*Output fano coding*/
    void fanoPrint(char str[N][MAX]) {
        for(i0 = 0; i0 < N; i0++)
            for(j0 = 0; j0 < length(str[i0]); j0++)
                printf("%c", str[i0][j0]);
        printf("n");
    }
}

Coding features of fano code:
Conduct amplifying treatment $f(n) = 2 * f(n/2) + n$ for recognizable time complexity $f(n) = f(k) + f(n-k) + n + 2$, the time complexity is known as $O(2^n)$, the average code length: $L < H(S) / \log r + 2$.

The non-recursion of fano code is similar; no analysis will be made temporarily here.

### 2.3 Huffman Coding

#### 2.3.1 Huffman Recursive Coding

Coding principle: make scheduling for sequence each time, then encoding for two minimum codes, recursion will be conducted till all the codes are encoded com-
void HuffmanCode(float p[N])
{
    /*Reverse Bubble sort*/
    int i,j,v=1;
    float temp;
    char b[N][MAX];
    for(i=0;i< N-1;i++)
    {
        for(j= N-1;j> i;j--)
            if(p[j]> p[j-1])
            { temp=p[j];
              p[j]=p[j-1];
              p[j-1]=temp;
            }
    }
    if(N==2)
    {
        b[0][v]='0';
        b[1][v]='1';
    }
    else
    {
        b[N-2][v]='0';
        b[N-1][v]='1';
        v++;
        HuffmanCoding(p[N-1]);
    }
}

void HuffmanPrint(char b[N][M])
{
    int i,j;
    for(i=0;i< N;i++)
    {
        for(j=length(b[i]);j>0;j--)
            printf(" %c ", b[i][j]);
        printf("n");
    }
}

Coding characteristics of Huffman recursive code:
1. From f(n)=f(n-1)+n^2, we know the cycle time is O(n^3), in fact, the sequence has order, and Bubble sort cycle time is approximate to 0, and for many cases the cycle time is about O(n^2).

2.3.2 Huffman Non-recursive Code
Coding principle: create a Huffman tree according to the sequential queue, then the cycle times can be reduced greatly for the traversal.
Steps are conducted as follows:

1) Sequence the signs of information sources in accordance with decreasing order of the probability occurred.

2) Merge and plus two minimum probability occurred, the result obtained will be taken as the occurrence probability of the new sign.

3) The step 1 and step 2 will be conducted repeatedly till the result of probability summation equals to 1.

4) During the joint operation, sign with large probability will be represented with code 0, and sign with small probability will be represented with code 1.

5) Record the 0, 1 sequence of probability from 1 place to the place between present signal sources signs, so as to obtain the code of each signs.

Algorithm realization
1. Create Huffman tree, and traversal
typedef struct
    {char data[5];
    int weight;
    int parent,lchild,rchild;
    }HTNode;
typedef struct
    {char cd[N];
    int start;
    }HTCode;
void CreatHuffmanTree(HTNode ht[],int n)
{
    /* Create Huffman tree*/
    int i,k,lnode,rnode,min1,min2;
    for (i=0;i<2*n;i++)
    {
        ht[i].parent=-1;ht[i].lchild=-1;ht[i].rchild=-1; }
    for(i=n;i<2*n-1;i++)
    {
        min1=M,min2=M;
        lnode=-1,rnode=-1;
        for(k=0;k< i-1;k++)
            if(ht[k].weight< min1)
                rnode=lnode;
            else
                lnode=k;
        min1=ht[i].weight;
        min2=ht[i].weight;
        for(k=0;k< i-1;k++)
        {
            if(ht[i].parent==lnode)
                if(ht[k].weight<min1)
                    { mnode=lnode;
                      lnode=k;
                      min2=min1;
                      min=ht[k].weight;
                    }
                else
                    if(ht[k].weight<min2)
        

2 Create Huffman Coding

```c
void CreatHuffmanCode(HTNode ht[], HCode hcd[], int n)
{
    /* Create Huffman tree*/
    int i, f, c;
    HCode hc;
    for(i=0; i<n; i++)
    {
        hc.start = n;
        c = i;
        f = ht[i].parent;
        while(f != -1)
        {
            if(ht[f].lchild == c)
            {
                hc.cd[hc.start-1] = '0'; /* assign 0 for the left sub-tree*/
            }
            else
            {
                hc.cd[hc.start-1] = '1'; /* assign 1 for the right sub-tree*/
            }
            c = f;
            f = he[f].parent;
        }
        hc.start++;
        hcd[i] = hc;
    }
}
/*Output Huffman code*/
void DispHuffmanCode(HTNode ht[], HCode hcd[], int n)
{ int i, k;
    for(i=0; i<n; i++)
    {
        for(k=hc[i].start; k<n; k++)
        {
            printf("%c", hcd[i].cd[k]);
        }
        printf("\n");
    }
}
```

Characteristics of Huffman Code:
Compared with non-recursive, the cycle times is reduced by one order \(O(n^2)\). Average code length:
\(L \in (H(S)/\log r, H(S)/\log r+1)\)

1) The coding efficiency of different signal source is different, when the code probability of signal source is the negative power of 2, it reaches 100% encoding efficiency; if the probability of signs of the signal sources is the same, encoding efficiency is the lowest.

2) The assignment of "0" and "1" is arbitrary, so the above optimum codes compiled are not unique; however, the coding efficiency will not be affected for the average code length is the same.

2.4 Arithmetic Coding

Coding principle: define cumulative probability for sequence, then adopt S-F-E method for the encoding, the efficiency of sequence with little information signs can be increased.

```c
void SuanshuCoding(float b[N], float p[M], char a[M])
{
    int i, j;
    for(i=0; i<N; i++)
    {
        for(j=0; j<M; j++)
        {
            for(k=0; k<=j; k++)
            {
                if(b[i] == a[k])
                {
                    Ps[k] *= p[k]; /* define cumulative probability for sequence*/
                }
                else
                {
                    break;
                }
            }
            for(j=1; j<=M; j++)
            {
                if((int)(acp[j]*2)==0)
                {
                    avp[j] = 2;
                    printf('0');
                }
            }
            for(i=1; i<=M; i++)
            {
                sum[i] += Ps[i-1];
                avp[i] = sum[i] + Ps[i]/2.0;
                l[i] = (int)(-log(avp[i]))+1;
                for(j=0; j<l[i]; j++)
                {
                    if((int)(acp[j]*2)==0)
                    {
                        avp[j] = 2;
                        printf('0');
                    }
                }
            }
        }
    }
}
```
Characteristics of arithmetic coding:

The same has time complexity: $O(n^3)$, average code length: $L \in [N?H(S),N?H(S)+1]$.

Arithmetic coding can be static or self-adaptive. In static arithmetic coding, the probability of source signals is fixed. In the self-adaptive arithmetic coding, the probability of source signals will be modified dynamically according to the occurrence frequency of codes; When compress information, the probability shall be estimated in the encoding process, and the key for confirming the compression efficiency of the encoder is dynamic modeling.

There is another way for arithmetic coding: represent the information of encoding as a space between the real number 0 and 1, the longer the information, the smaller the space of the code is, and the more the binary digit is required for the space. Two basic parameters are used: the probability of the signal and the encoding space of it.

Among them, the probability of information source signs determines the efficiency of compression coding, and these spaces are included between 0 and 1. The spaces in the encoding process determine the output after the signs are compressed.

The steps are as follows:

1) When starts, the encoder will set the “current space” as [0, 1];
2) Encoder conducts treatment for each event according to the steps (a) and (b);
   (a) The encoder divides “current space” into sub-spaces, one for each event;
   (b) The size of one sub-space is proportional to the probability of the next event, the selection of sub-space of the encoder is corresponding with the next event which is going to happen exactly, and make it the new “current space”.

Algorithm realization:

```c
void  SuanshuCoding(double p[N])
{
    int  i,j=0;
    for(i=0;i<count;i++)
    {
        if(code[0]==number[i])  break;
    }
    printf("n");
}
```

```
else
    printf('1');
}
printf("n");
```
```c
{count[j]++; /*preserve the run length temporarily*/
    ps[j]=p[0];
    j++;
}
else
    {count[j]++;  
    ps[j]=p[0];
    /*Multiply-Accumulate to 0, 1 sequence*/
    }
}  
else
    {if(b[i]!=b[i+1])
        {count[j]++;
        ps[j]*=p[1];
        j++;
        }
    else
        {count[j]++;
        ps[j]*=p[1];
        }
    }
}
HuffmanCoding(ps[j]);
}
```

Characteristics of Run Length Coding:
Recognizable time complexity is $O(n^2)$, the average code length will be confirmed by treatment method.

Run length Coding is applicable to binary sequences only, the compression ratio of it is related to the probability occurred in the text and the length of it. Under the condition that the occurrence times of word is the same in the text, the long the average length of the repeated word string, the higher the compression ratio is; Under the condition that the average length of word string is the same, the more the times of the word string occurred, the higher the compression ratio is.

Most of the following coding methods are the combinations of basic coding methods:
LZ code $L = H_{\infty}$
S-F-E code $L \in (H(S)+1,H(S)+2)$
MH code $L \in [H_{wb},H_{wb}+P_{w}/n_{w}+P_{b}/n_{b})$
For space constraints, other coding methods that are rarely used will be overlooked.

In the opinion of the author, for there is linear correlation between $L$ and $H(S)$, when the difference of the probability of information source sequence forms the disperse symmetry, and the geometric distribution of it is approximately loose ball, $H(S)$ reaches the minimum, and take the minimum $L$ for the time being. Although the sequence probability is not special, the sequence distribution can reach approximately the ideal state on large-scale event.

3 Conclusions
In summary, the universal lossless compression algorithm for high compression ratio in a real sense needs further studies for each lossless compression has its own applicable scope, and compression ratio is restricted by lossless requirements; thus, analysis for image data shall be conducted before the selection of algorithms, and flexible use of algorithms according to the characteristics of the data by algorithm idea is an efficient way for improve compression ratio.

References