Data Compression by Replacement of Symbol Pairs and Its Implementation

Hirofumi NAKAMURA[1]

(Accepted October 2, 2017)

Abstract For lossless data compression, an off-line encoding method which has analysis and cutting of input symbol string is proposed. The analysis is based on replacement of frequent symbol pairs appearing repeatedly. By the operation, the encoder does not cut in the symbol string which emerges more frequently in the input symbol string. The encode does not send dictionary data to the encoder directly, but send information for construction of dictionary to the encoder. Its decoding can be done fast with one-path method without deep analysis of its input. A set of efficient data structures for the encoding and the decoding such as hash table, frequency table, and bidirectional list is used. Theoritical proof that the encoding and the decoding can be executed in time $O(N)$ is proved, where N is the size of input data and $O(\cdot)$ is Bachmann-Landau Big O notation.

Keywords [Data compression, Most frequent symbol pair, Replacement, Concatenation, Digram]

1 Introduction

A data compression method based on cutting of input symbol string was proposed by Ziv and Lempel (LZ78[1)]) and many of its revised methods(LZW[2)], compress[3)] and so on[4, 5, 6)]) are proposed and widely used.

The gzip(using another method proposed by Ziv and Lempel(LZ77[7)]) and Huffman Coding) is also widely used.

There are methods which do generation process of output by one input symbol: comp-2[8)](using Markov model and Arithmetic Coding) and Block-sorting method[9)](using run-through information of sorted data of all rotated whole data, Move To Front method and Huffman Coding).

Off-line methods based on replacement of symbol pairs which emerge repeatedly in input data are proposed by Nakamura and Murashima[10, 11, 12)], by Nevill-Manning et al.[13, 14, 15)](named SEQUITUR[14, 15)]) and by Gage[16, 17)] independently. Furthermore, an adaptive method is proposed by Nagayama et al.[18)].

LZ78 and its revised encoders have the following redundancies:
(1) even if a certain symbol string appears repeatedly, each appearance is cut in various positions and
(2) when input size N is finite, many enrolled symbol strings are not used later.
They are improved as follows[10, 11, 12)]:
(M1) encoder divides symbol strings between the symbols which contact with lower frequency and
(M2) encoder does not enroll unnecessary symbol strings by watching whole of input string.

The methods by Gage and the ones by us do similar analysis for input data in the point of that replacement is done for the most frequent symbol pairs. But how to stop the analysis and how to cut strings are different between the both methods.

This paper presents development of past reports[10, 11, 12)]. And, in this paper, implementation of the proposed method improving the computing time and the compression ratio is explained. Furthermore the time complexity of proposed method is also mentioned.

2 Encoding

Following terms are used in this paper. *Symbol pair*

[1]General Education Division, National Institute of Technology, Miyakonojo College

means a pair of two symbols. *Symbol pair i j* is expressed as *i j* merely. *S* is the symbol string under analysis. The count of a *symbol pair* in *S* is expressed as the *number of appearance*. The greatest *number of appearance* of current *S* is expressed as $Nmax(S)$. The most frequent *symbol pair* is expressed as $Pmax(S)$. Here, $Pmax(S)$ is a *symbol pair* whose *number of appearance* is $Nmax(S)$. If the candidates of the most frequent *symbol pair* exist more than or equal to twice, one appropriate pair is chosen as $Pmax(S)$.

In order to describe the processes of replacement of symbol strings in this paper, production rules like Nevill-Manning et al.[13] are mainly used instead of tree structures[10, 11]. Alphabet(code of input) corresponds to terminal symbol of production rule. The *symbol pair* corresponds to the digram of the paper of Nevill-Manning et al.[13] Code table is equivalent to collection of production rules and alphabets.

Proposed encoder consists of analysis, cutting and bit stream generation.

2.1 Analysis of Input Symbol String

In this subsection, the replacement of analysis is described. The termination of the iteration is described in subsection 2.4.

While the same *symbol pair* exists at least twice in *S*, encoder can repeat following two operations:

(A1) Encoder finds $Pmax(S)$.

(A2) Encoder replaces all $Pmax(S)$ in *S* with a new single symbol.

The new symbol is expressed as *enrolled symbol*. Encoder treats it as a new symbol. *Enrolled symbol* is equivalent to non-terminal symbol of production rule.

For example, the input data

S::= a c a b b a d c a d d b a a d d c a a d d b

consist of alphabet {a,b,c,d} is considered. On current *S*, the *number of appearance* of *symbol pair* a d is the greatest number 4 (=$Nmax(S)$). At first, it is considered that a d is equivalent to a new symbol A. Encoder replaces all a d (=$Pmax(S)$) in *S* with A. Then

S::= a c a b b A c A d b a A d c a A d b ,

A::= a d

are given.

Next, current $Pmax(S)$=A d is replaced with new symbol B. Then

S::= a c a b b A c B b a B c a B b ,

A::= a d , B::= A d

are given.

```
a c a b b a d c a d d b a a d d c a a d d b
  ∨       ∨   ∨        ∨    ∨  ∨
  C       A   A        A    C  A
               ∨        ∨        ∨
               B        B        B
                ∨                 ∨
                D                 D
```

(a) Construction of Analyzed Result

```
a c a b b a d c a d d b a a d d c a a d d b
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  ∨       ∨   ∨        ∨    ∨  ∨
  2       2   2        2    2  2
               ∨        ∨        ∨
               2        2        2
                ∨                 ∨
                2                 2
```

(b) Analyzed Result with *Enrollment Direction*

```
a c a b b a d c     d b a
1 1 1 1 1 1 1 1     1 1 1
  ∨       ∨   A               C
  2(C)    2(A) 1              1
               ∨        B
               2(B)     1
                ∨                 D
                2(D)              1
```

(c) Before *Re-enrollment*

```
    {a,b,c,d,z, C ,A ,B ,D }
  → {a,b,c,d,z, A',B',C',D'}
```

```
a c a b b a d c     d b a
1 1 1 1 1 1 1 1     1 1 1
  ∨       ∨   B'              A'
  2(A')   2(B') 1             1
               ∨        C'
               2(C')    1
                ∨                 D'
                2(D')             1
```

(d) After *Re-enrollment*

```
    1 a 2 1 c 1 a 1 b 1 b 2 1 a 1 d 1 c
      2 2 1 B' 1 d 1 b 1 a 1 C' 1 A' 1 D' z
 or
    1 211 1 1 211 1 22111 1 1 1 1  1
    a  ca b b  ad c  B'db a C'A'D' z
```

(e) Output

Fig.1 Example of Cutting from Analyzed Result

Next, current $Nmax(S)$ is 2. The *number of appearance* of c a and B b is 2. It is assumed that c a is selected from these as $Pmax(S)$. Encoder replaces c a with new symbol C. Then

S::= a C b b A c B b a B C B b ,

A::= a d , B::= A d , C::= c a

are given.

Next, current $Pmax(S)$=B b is replaced with D. After that, encoder can not continue replacement because $Nmax(S) < 2$. As the result, encoder obtains

S::= a C b b A c D a B C D ,

A::= a d , B::= A d , C::= c a , D::= B b .

Proposed method catches the most frequent symbol string "add"(see also Fig. 1(a)).

If replacement is done while $Nmax(S) \geqq 2$, there are cases that enrollment is excessive. Proposed method uses a revised judgment of termination of analysis based on rough estimation of output length. The judgment(call it **RepeatCheck**) is described in subsection 2.4.

2.2 Cutting of Input Symbol String

2.2.1 Enrollment direction

It is possible to do self-referencing cutting, and to send only directions of enrollment instead of code table.

It is assumed that analyzed result consists of tree structures and symbols like Fig. 1(b), where 1 means leaf, and 2 means parent node of the two children. This information(1 and 2) of tree structure is expressed as *enrollment direction*. If these are transferred to decoder using prefix notation with symbols, then decoder will accept them from left like this: 1a, 2 1c 1a, 1b, 1b, 2 1a 1d, 1c, 222 1a 1d 1d 1b, ... and so on.

It is assumed that decoder enrolls *symbol pair* as a new symbol when symbol 2 is accepted, then other same sub-trees can be expressed by one leaf. But decoder accepts these from left. So, encoder must do re-numbering of enrolled symbols. The re-numbering is expressed as *re-enrollment*. And the re-numbered symbol is expressed as *outside symbol*. *Outside symbol*s is expressed with prime. C,A,B,D are re-numbered like A',B',C',D'.

Proposed method avoids division strings appearing more frequently, excepting for the first appearance for enrollment. In the former example, output becomes like Fig. 1(e), where z is a terminal symbol that is introduced to indicate the end of sequence.

2.2.2 Cutting based on Multi-branch Tree

It is assumed that another example given by

S_1::= a b c d e b c d f b c d e b c d f g

which is the input string treated in the paper[13] of Nevill-Manning et al. Its analyzed result becomes as follows:

S_1::= a E E g ,

A::= b c, B::= A d, C::= B e, D::= C B, E::=D f .

But there are symbols used for symbol definition only. Symbols A, C and D do not appear in the output

1a 2 2 2 2 2 1b 1c 1d 1e 2 1B 1f 1E 1g 1z.

It is possible to express analyzed result like follows:

1a 4 3 1b 1c 1d 1e 1A' 1f 1B' 1g 1z

by using 1,2,3,4,⋯,$Bmax$ as *enrollment direction*, where $Bmax$ is a constant value. Symbols at most $Bmax$ can be enrolled to a new *enrolled symbol* directory, and unused symbols are not necessary as *outside symbol*. Its decoder can reconstruct code table as follows:

A'::= b c d , B'::= A' e A' f .

Even if decoder's code table is also based on binary tree(like T in Fig. 3 mentioned later), decoder can reconstruct production rules

A::=bc, B::=Ad, C::=Be, D::=CB, E::=Df,

where A,B,C,D and E are *enrolled symbol*s to refer *symbol pair*s in the code table. Decoder needs another table(V in Fig. 2 shown later) which contains

A'::=B , B'::= E

for conversion from *outside symbol* to *enrolled symbol*. Production rules of encoder side and decoder side are a little bit different. But original input string can be uniquely reconstructed, because produced symbols of *outside symbol*s are the same in both sides.

2.3 Bit Stream Generation

Proposed method uses adaptive arithmetic coding[6, 19] for bit stream generation. Values of frequency and accumulation are memorized in a complete binary tree[6]. A frequency value is memorized on a leaf. Each node except leaves memorizes the sum of its children's values. Accumulation value is provided by adding each left brother's value on a course following from a leaf to the root(if left brother does not exist, no value is added).

Two trees are prepared. One for *enrollment direction* and the other for *outside symbol*. We unified output by doing numeration using common variable to affect output for both data.

2.4 Termination of Replacement in Analysis

RepeatCheck decides its return value as follows:

$$\begin{aligned} &\text{RepeatCheck} \equiv \\ &\quad Nmax(S) \geqq 2 \quad \text{and} \\ &\quad roughoutputlength(|S|, C_0, C) > \\ &\quad\quad roughoutputlength(|S| - Nmax(S) + 1, C_0, C+1)\ , \end{aligned}$$

where $|S|$ is the count of elements in current S,

$$\begin{aligned} C_0 &= \langle alphabet\ size \rangle + \langle 1\ (for\ z) \rangle \\ &= K + 1, \end{aligned}$$

$$\begin{aligned} &roughoutputlength(r, c_0, c) \\ &\approx \sum_{\langle symbol\ output \rangle} \log \left\langle \begin{array}{c} size\ of\ code\ table \\ on\ each\ output \end{array} \right\rangle \\ &\quad + \left\langle \begin{array}{c} roughly\ estimated\ output\ size \\ of\ enrollment\ directions \end{array} \right\rangle \\ &\approx (r + c - c_0 + 1) \times\ average(c_0, c) \\ &\quad + (r + c - c_0 + 1) \times\ 1, \end{aligned}$$

the base of log is 2 and

$$average(l, h) = \frac{\sum_{c=l}^{h} \log c}{h - l + 1}.$$

By Stirling's approximation,

$$\begin{aligned} &average(l, h) \\ &\approx \left(h + \frac{1}{2}\right) \log h - \left(l - \frac{1}{2}\right) \log(l-1) - (h-l+1) \log e. \end{aligned}$$

It is assumed that the length of one symbol is $\log C$ bits when size of current code table is C and the length of one *enrollment directions* is 1 bit. And it is assumed that average length of one output is roughly equal to $average(l, h)$, where the initial size of code table is l and last size is h. Appearance probability is disregarded.

For shortening of processing time, estimation is used only when $Nmax(S)$ changes.

3 Decoding

At the decoding side, according to *enrollment direction*, decoder can reproduce code table of *outside symbol*, and can reconstruct original symbol string.

4 Algorithm

Fundamental algorithms of encoding and decoding are shown in Fig. 2 in Pascal-like language. Meanings of major symbols are shown as follows.

K: Size of alphabet. A symbol whose ordinal number is K is used to indicate the end of sequence. K is 256 for byte data.

RepeatCheck: Judgment of termination of analysis.

```
Procedure ReferenceCount(ci);
  if ci ≦ K then  {no operation}
  else if OutsideSymbol(ci) = nil then
    ReferenceCount(Left(ci)); ReferenceCount(Right(ci));
    Reenroll(T,ci,-1); Tb[ci]:=1;
  else
    Tb[ci]:=Tb[ci]+1;
  endif;
endproc;
function SetOutsideSymbol(ci);
  if ci ≦ K then return(1);
  else
    bl:=SetOutsideSymbol(Left(ci));
    br:=SetOutsideSymbol(Right(ci));
    if bl+br > Bmax and bl > 1 then
      Cout:=Cout+1; Reenroll(T,Left(ci),Cout);
      Tb[Left(ci)]:=bl; bl:=1;
    endif;
    if bl+br > Bmax and br > 1 then
      Cout:=Cout+1; Reenroll(T,Right(ci),Cout);
      Tb[Right(ci)]:=br; br:=1;
    endif;
    if OutsideSymbol(ci) =-1 and Tb[ci] ≧ 2
         or bl+br = Bmax then
      Cout:=Cout+1; Reenroll(T,ci,Cout);
      Tb[ci]:=bl+br; bl:=1; br:=0;
    endif;
    return(bl+br);
  endif;
endfunc;
procedure Write(ci);
  if ci ≦ K then
    WriteDirection(1); WriteSymbol(ci);
  else if OutsideSymbil(ci) =-1 then
    Write(Left(ci)); Write(Right(ci));
  else if OutsideSymbil(ci) ≠ -1 and Tb[ci] ≧ 2 then
    WriteDirection(Tb[ci]);
    Write(Left(ci)); Write(Right(ci)); Tb[ci]:=1;
  else  {OutsideSymbil(ci) ≠ -1 and Tb[ci] < 2}
    WriteDirection(1); WriteSymbol(OutsideSymbol(ci));
  endif;
endproc;
procedure Encode;
  Readfile(S); Initialize; Cin:=K; Cout:=K;
  while RepeatCheck do begin
    cl cr:=Pmax(S); Cin:=Cin+1;
    Enroll(T,cl,cr,Cin); Replace(S,cl,cr,Cin);
  end;
  for ci:=each element of S do ReferenceCount(ci);
  for ci:=each element of S do w:=SetOutsideSymbol(ci);
  for ci:=each element of S do Write(ci); Write(K);
endproc;
```

(a) Encoding algorithm

```
procedure WriteString(ci);
  if ci < K then      WriteAlphabet(ci);
  else if ci > K then
    WriteString(Left(ci)); WriteString(Right(ci));
  endif;
endproc;
function Read;
  t:=ReadDirection;
  if t = 1 then
    ci:=ReadSymbol; if ci > K then ci:=V[ci];
    WriteString(ci); return(ci);
  else
    cl:=Read;
    for i:= 2 to t do begin
      cr:=Read; Cin:=Cin+1; Enroll(T,cl,cr,Cin); cl:=Cin;
    end;
    Cout:=Cout+1; V[Cout]:=Cin; return(Cin);
  endif;
endfunc;
procedure Decode;
  Initialize; Cin:=K; Cout:=K;
  while Read ≠ K do  {no operation} ;
endproc;
```

(b) Decoding algorithm

Fig. 2 Fundamental algorithms of encoder and decoder

Enroll(T,cl,cr,ci): Enroll cl cr to code table T as *enrolled symbol* ci.

Replace(S,cl,cr,ci): Replace every cl cr in S with symbol ci.

WriteDirection(c): Output *enrollment direction* c.

WriteSymbol(c) : Output *outside symbol* c.
OutsideSymbol(c): Return *outside symbol* assigned for *enrolled symbol* c.
Left(c), **Right**(c): Return left and right side of *symbol pair* that enrolled as *enrolled symbol* c.
Reenroll(T,ci,co): Assign co(*outside symbol*) to ci(*enrolled symbol*).
ReadDirection: Input *enrollment direction*.
ReadSymbol: Input *outside symbol*.

5 Implementation and Time Complexity

5.1 Data Structures

5.1.1 Outline of Data Structures

Compression time mainly depends on finding $Pmax(S)$ and replacement. The points of data structures are
(D1) how to get the most frequent *symbol pair* in S from a *number of appearance*(in Fig. 3, $F \to H \to S$) and
(D2) how to get the *number of appearance* from a *symbol pair*($H \to F$). There are many *symbol pair*s whose *number of appearance* are the same, so links are used(Ffd, Ffu, Hfd, Hfu). There are many positions where the same *symbol pair*s are contained in, so links are used(Hpd, Hpu, Spd, Spu).

5.1.2 Details of Data Structures

The relations of data structures are shown in Fig. 3. In the figure, an input string

c a d d b a a d d c a a d d b

consisting of alphabet a,b,c,d and hash function $f(i,j) = (i \times 7 + j \times 3)\ mod\ 16$ for *symbol pair* $i\ j$ are used, where *mod* gives remainder of division. After input of data, initialization of data structures, enrollment of first $Pmax(S) =$ a d to the code table T and replacements of $Pmax(S) =$ a d in S, contents become as Fig. 3. Last valid values are remained in parentheses.

Two-way lists(headers are Ffd and Ffu of array F whose index is the *number of appearance*) are prepared to find one $Pmax(S)$ in time $O(1)$, where $O(\cdot)$ is Bachmann-Landau Big O notation. $Fpos$ is set to last $Nmax(S)$, so encoder can use first element of list $F[Fpos]$ as $Pmax(S)$ in most case. Each two-way list links hash table's buckets whose data in Hv are the same. These lists link *symbol pair*s(impremented by Hl and Hr of hash table H) for each *number of appearance*.

And two-way lists(headers are Hpd and Hpu of H) are prepared to know each position of $Pmax(S)$ in S for replacement in time $O(1)$. These lists link *symbol pair*s' all locations in S for each *symbol pair*.

And hashing are used to memorize the *number of appearance*. The *numbers of appearances* of all *symbol pairs* are memorized and updated instead of counting whenever encoder finds $Pmax(S)$. By using hashing, one updating operation of the *number of appearance* can be done in time $O(1)$. It is efficient for decreasing necessary memory to use hashing instead of two dimensional array indexed by two symbols.

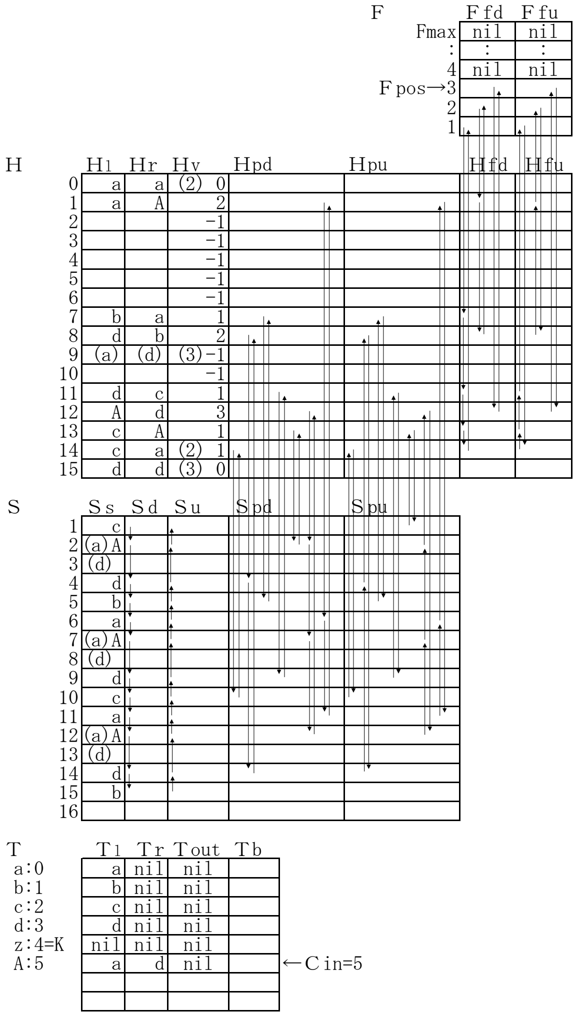

Fig.3 Outline of Data structures

Hash's buckets keep the *number of appearance*s of *symbol pair*s in field Hv when $Hv > 0$. $Hv = -1$ means the bucket is free. All buckets' Hv are -1 when initialization is done. $Hv = 0$ means the bucket does not memorize *symbol pair* but encoder must search followed buckets when hashing is not hit.

$Fmax$ is the size of F. $Fmax$ can be decreased to $\lceil \sqrt{N} \rceil$, where $\lceil x \rceil$ expresses ceiling of x. The element $F[Fmax]$(means $Ffd[Fmax]$ and $Ffu[Fmax]$) links Buckets so that $Hv \geqq Fmax$. In the severe case,

$F[Fmax]$ links at most $\lceil N/Fmax \rceil \leqq \lceil \sqrt{N} \rceil$ buckets. Then to find $Pmax(S)$ for replacement on analysis, encoder searches this link at most $\lceil \sqrt{N} \rceil$ times by linear search in time $O(\sqrt{N})$, because of that $F[Fmax]$ links buckets whose Hv is not smaller than $Fmax$. Total time complexity of finding $Pmax(S)$ does not exceed $O(\sqrt{N}) \times O(\sqrt{N}) = O(N)$.

The number of *symbol pairs* does not exceed both N and $|S|^2$. So the number of necessary buckets of hashing does not exceed both of them. Before the iteration of replacing, the number of necessary buckets does not exceed K^2. It is assumed that encoder does nullify of buckets whose Hv is 1 before iteration of replacing. After that minimum value of valid Hv becomes 2. Then the number of necessary buckets of hashing does not exceed $N/2$. For good management of hashing, hash space is increased at least about 20%. I choose

$$Hmax = \begin{cases} 1.2N & (N \leqq K^2) \\ 1.2K^2 & (K^2 < N \leqq 2K^2). \\ 0.6N & (others) \end{cases}$$

Here, $Hmax$ is $O(N)$.

5.2 Time Complexity

R denotes the number of elements in analyzed result S. C denotes the size of code table consisting of alphabet, z and all *enrolled symbols*.

5.2.1 Time Complexity of Input and Initialization

In Fig. 3, $Tl[i], Tr[i]$ and $Tout[i] (i = 0 \sim K-1)$ are filled to make it understandable. But there is no need to use them, because $Tl[i]$ is always $i (i = 0 \sim K-1)$. Initialization of T can be done in $O(1)$.

Data input is done in time $O(N)$.

To set $Sd[m] \leftarrow m+1 (m = 1 \sim N-1)$ and $Su[m] \leftarrow m-1 (m = 2 \sim N)$ are done in time $O(N)$. To set $Hv[h] \leftarrow -1 (h = 0 \sim Hmax-1)$ is done in time $O(N)$, because $Hmax = O(N)$. To update

$$Hv[f'(Ss[m], Ss[m+1])] \leftarrow \begin{cases} 1 & (first\ time) \\ Hv[f'(Ss[m], Ss[m+1])]+1 & (others) \end{cases}$$

with filling Hl, Hr and linking $Ss[m]$ to two-way list(the headers are Hpd and Hpu and the bodies are Spd and Spu) is done for $m = 1 \sim N-1$ in time $O(N)$. Here, the index of the bucket for *symbol pair* $i\ j$ is expressed by $f'(i, j)$. To set $Ffd[p] \leftarrow nil$ and $Ffu[p] \leftarrow nil (p = 1 \sim \lceil \sqrt{N} \rceil)$ are done in time $O(\sqrt{N})$. To link $H[h]$ to $Ffd[Hv[h]]$ and $Ffu[Hv[h]] (h = 0 \sim Hmax-1)$ is done in time $O(N)$.

From these, data input and initialization can be done in time $O(N)$.

5.2.2 Time Complexity of Analysis

At first, $Fpos$ is set to $\lceil \sqrt{N} \rceil$ and Cin(the value is code table's current size minus one) is set to K.

To find $Pmax(S)$, encoder uses F. Encoder removes $Pmax(S)$ from F, nullifies $Pmax(S)$ in H and enrolls $Pmax(S)$ to code table T. These can be done in time $O(1)$. Total count of finding $Pmax(S)$ and that of enrollment are equal. They does not exceed N. So time complexity of analysis does not exceed $O(N) \times O(1) = O(N)$.

After that encoder finds all the place of $Pmax(S)$ in S by using Hpd and Spd, and changes Ss, Sd, Su. Then encoder updates H: decreases Hv, increases Hv and enters new *symbol pairs* to H if needed.

When a *symbol pair* $i\ j$ in S::= ... $u\ i\ j\ v$ ... is replaced with X, the *numbers of appearance* of $u\ i$, $j\ v$ and $i\ j$ decreases, and the *numbers of appearance* of $u\ X$ and $X\ v$ increases. Then symbols in S are moved among the two-way lists constructed by Hpd, Spd, Hpu and Spu. And buckets of H are moved among the two-way lists constructed by Ffd, Hfd, Ffu and Hfu.

One updating of the *number of appearance* is done in time $O(1)$, so replacement of one place is completed in time $O(1)$. The length of S decreases by 1 by the replacement operation of one place in S, so the number of replacement operations in S does not exceed N. On these account, the time complexity of analysis does not exceed $O(N) \times O(1) = O(N)$.

5.2.3 Time Complexity of Cutting

The cutting is done by **ReferenceCount** and **SetOutsideSymbol** in Fig. 2. The calling paths of **ReferenceCount** and **SetOutsideSymbol** are the same as depth-first search path of the trees of analyzed result like Fig. 1(a).

The number of leaves of analyzed result is N. So the time complexity of execution of **ReferenceCount** and **SetOutsideSymbol** is $O(N)$.

5.2.4 Time Complexity of Bit Stream Generation

When the Arithmetic Coding with complete binary tree is used on **Write**, process of the bit stream generation for one symbol can be calculated in $O(\log C)$. And process of the bit stream generation for one *enrollment direction* can be calculated in $O(\log Bmax) = O(1)$. Time necessary for the bit stream generation depends on total process for symbols and *enrollment directions*.

It is expressed as follows.

$$O(\langle Time\ necessary\ for\ Code\ Generation \rangle) = O(R + C - C_0 + 1) \times O(\log C) + O(R + C - C_0 + 1) \times O(1). \quad (1)$$

I do not know the order of Eq.(1) yet. But the order of output length is also expressed as Eq.(1). If output of coding does not expand in the extreme, order of output length is the same as the size of input. So, time complexity of bit stream generation is $O(N)$.

5.2.5 Time Complexity of Decoding

Decoder does not need to use F, H and S. Complexity of its decoding mainly depends on the enrollment of *symbol pair*s and production of original alphabet strings. The time complexity of the enrollment(updating of T) is $O(1)$. Total time complexity of the enrollment of decoder does not exceed $O(N) \times O(1) = O(N)$. The paths of decoder's productions are similar to the depth-first search path of trees of analyzed result like Fig. 1(a). The number of leaves of the trees is N. So the time complexity of the production of decoding is $O(N)$.

6 Comparison with Similar Methods

The comparison data are shown in Table.1. In the method of Nevill-Manning et al.[13], S begins with null, and input symbols are added to S symbol by symbol. On each adding, encoder finds a *symbol pair* that appears twice in S and production rules in all. If such a *symbol pair* exists, encoder makes a new production rule, defines a new non-terminal symbol, and replaces existed *symbol pair*s with the new non-terminal symbol. This method does integration and abolition of production rules if possible. For the example data S mentioned in subsection 2.1, provided analyzed result is shown in Fig. 4(a). This method can output after having analyzed whole input data.

Proposed method, method of Gage and method of Nevill-Manning et al. sometimes give the equivalent analyzed result. An example is S_1::= a b c d e b c d f b c d e b c d f g mentioned in subsection 2.2.2.

In the method of Nagayama et al.[18], on each adding of input symbol to S, encoder looks for a *symbol pair* that appear twice in only S. If such a *symbol pair* exists, encoder replaces the rear with a symbol whose ordinal number corresponds to the location of the first appearance in S. While there is a *symbol pair* which exists twice in S, encoder repeats replacement. This method does not send code table, and does not need to send any direction for enrollment. This is an adaptive method. Analyzed result by this method is shown Fig. 4(b). Numerals(1 to 15) are used as non-terminal symbols here.

Table 1 Comparison data with similar methods

	Nevill-Manning et al.[14, 15]	Nagayama et al.	Gage	Proposed
Initial of S	null	null	whole of each input block	whole of input
Addition to S	symbol by symbol	symbol by symbol	no	no
Finding symbol pair	twice	twice	most frequent ($Pmax(S)$)	most frequent ($Pmax(S)$)
Find in	S and code table	S	S	S
Replacing in	S and code table	S except first appearance	S	S
Rule reduction	yes	no	no	no
Timing of analysis	while inputting	while inputting	after inputting all	after inputting all
Timing of output	after analysis (not adaptive)	while inputting (adaptive)	after analysis (not adaptive)	after analysis (not adaptive)
Time complexity	$O(N)$	$O(N)$	not mentioned	$O(N)$

Gage's method and proposed method are both based on replacement of the most frequent *symbol pair*. Gage's method divides input data into small blocks. Analysis, cutting and generation are iterated block by block. It uses only unused byte code words(less than 256 code words) in each block as *enrolled symbol*s. And it sends code table with only byte code words. So output can be constituted with byte code words. These save necessary memory and processing time[16, 17]. But if size of input data block is very small or very big, the compression ratio becomes bad. Because the number of enrollments becomes few when the block is very small, and the number of enrollments does not exceed 256 even if the block is big.

The experimental result is shown in Table 2. The experiment is done for 14 files of Calgary Text Compression Corpus[6](it is expressed as TCC) . On the experiment, it is assumed that input alphabet is byte code(it is expressed as character or as char). To divide input data into small blocks is necessary for Gage's method.

```
S::= a A b B d A C B D A D b
A::= c a , B::= b a , C::= d d , D::= a C .

a c a b b a d c a d d b a a d d c a a d d b
  V     V     V   V   V     V   V     V
  A     B     A   C   B     C   A     C
                            V         V
                            D         D
```

(a)By the Method of Nevill-Manning et al.

```
S::= a c a b b a d 2 d d 5 6 7 6 d b
1::= a c , 2::= c a , 3::= a b , 4::= b b , 5::= b a ,
6::= a d , 7::= d 2 , 8::= 2 d , 9::= d d , ...

a c a b b a d c a d d b a a d d c a a d d b
              V       V   V     V   V
              2       5   6     2   6
                              V
                              7
 V V V V V V V   V V V   V    V    V  V  V
 1 2 3 4 5 6 7   8 9 10 11    12   13 14 15
```

(b)By the Method of Nagayama et al.

Fig.4 Analyzed Results

Table 2 Compression performance of Gage's method for TCC

Block length [K Bytes]	Compression ratio [bits/char]
1	4.956
4	4.144
16	4.099
64	4.370
256	4.751
1024	4.978

Proposed method uses the *enrolled symbols* whose ordinal numbers follow alphabet size K. Compression ratios of similar methods are shown in Table 3. $Bmax$ is chosen to 12 which is the best value on TCC. Proposed method is better than the methods of Nevill-Manning et al., Nagayama et al. and Gage.

Table 3 Average compression ratios for TCC

Method	Send dictionary	Adaptive
by Nevill-Manning et al.[13)]	3.13[13)]	2.70[13)]
SEQUITUR by Nevill-Manning et al.[14, 15)]	-	2.64[14, 15)]
by Nagayama et al.[18)]	-	3.29[18)]
by Gage*[16, 17)]	4.11	-
Block-sorting[9)]	2.428[9)]	-
Proposed	-	2.46

*Program was prepared by us.

7 Comparison with LZ78 and LZW

Table 4 Comparison data with LZ78 and LZW

File name and size [Bytes]	LZ78 Enroll. [times] and used ([%])	LZ78 Comp. ratio [bits /char]	LZW Enroll. [times] and used ([%])	LZW Comp. ratio [bits /char]	Proposed ($Bmax$ = 2) Enroll. [times] and used ([%])	Proposed Num. of cutting [times]	Proposed Comp. ratio [bits /char]
bib	21407	4.150	26860	3.346	5469	20630	2.461
111261	(45.6)		(48.4)		(80.5)		(2.156)
book1	131007	4.094	154226	3.274	23586	151644	3.012
768771	(43.5)		(43.5)		(96.6)		(2.630)
book2	102421	3.982	120683	3.148	21147	103591	2.550
610856	(47.0)		(47.2)		(90.4)		(2.265)
geo	26211	5.588	42838	6.077	707	50463	5.344
102400	(19.5)		(25.4)		(94.5)		(4.556)
news	73389	4.525	90904	3.757	20076	75582	2.988
377109	(42.9)		(43.4)		(72.4)		(2.596)
obj1	5983	5.538	9067	5.229	1653	8108	4.356
21504	(32.2)		(30.5)		(78.3)		(3.825)
obj2	50858	4.690	68090	4.170	14657	50147	2.885
246814	(45.1)		(44.9)		(70.2)		(2.570)
paper1	12130	4.748	15369	3.775	3561	12347	2.900
53161	(45.1)		(46.0)		(79.6)		(2.602)
paper2	17320	4.474	21331	3.520	4297	18400	2.860
82199	(44.8)		(44.9)		(89.6)		(2.560)
pic	26576	1.131	35057	0.970	6009	36440	0.936
513216	(41.7)		(40.3)		(93.8)		(0.801)
progc	9394	4.852	11978	3.868	2845	9375	2.901
39611	(44.9)		(45.7)		(75.9)		(2.597)
progl	13543	3.957	16524	3.032	4187	11402	2.029
71646	(51.0)		(51.2)		(68.0)		(1.818)
progp	9764	4.057	12016	3.113	3148	7677	1.919
49379	(50.6)		(51.0)		(60.7)		(1.724)
trans	18125	4.123	22440	3.266	5915	12425	1.756
93695	(54.1)		(55.0)		(52.8)		(1.554)

Comparison of data with LZ78 and LZW are shown in Table 4. Each processing time[sec] of coding and decoding is a value measured on the workstation Fujitsu S-4/20H. It is total time used by user and by system, and it is an average value of 10 trials. Experiments are done on the condition that code table's indices are expressed in $\lceil \log c \rceil$ bits (where c is current size of code table), additional characters are expressed in 8 bits on LZ78 and *enrollment directions* are expressed in 1 bit($Bmax$ = 2) on proposed method. On LZ78 and LZW, the number of enrollment is different from the number of cutting at most one. Ratio of used symbol strings(whose size is ≧ 2) are shown in parentheses. It is calculated by

$$\frac{\langle the\ number\ of\ used\ enrolled\ symbol\ strings \rangle}{\langle the\ number\ of\ enrollment\ after\ initialization \rangle}.$$

Entropy is shown in parentheses. It is calculated from symbols' appearance probabilities in whole output. On LZ78, it is the sum of entropy of the part of code table index and the part of additional character. On proposed method, it is the sum of entropy of the part of *enrolled symbol* and the part of *enrollment direction*.

The number of cutting of proposed method is 69%〜193%(average is 109%) of LZ78, 55%〜118%(average is 83%) of LZW. There are increased case and decreased case. Compression ratio does not depend on the number of cutting and average length of cut symbol strings.

The number of enrollment becomes less than 1/3 of both LZ78 and LZW. Ratio of used symbol strings becomes higher on proposed method, because the number of enrollment decreases more than the decrease of cutting number.

8 Other Experimental Results

Compression ratio and execution time of proposed method are shown in Table 5. The execution time per one character is approximately constant.

Table 5 Input data length and compression performance

File length [Bytes]	Compression ratio [bits/char]	Encode time [μ sec/char]	Decode time [μ sec/char]
2^{15}	2.325	14.62	2.38
2^{16}	2.261	15.05	2.01
2^{17}	2.229	14.24	2.05
2^{18}	2.218	15.35	2.17
2^{19}	2.188	15.12	2.12
2^{20}	2.160	14.85	2.07

$K = 256, Entropy = 2.000$

Proposed method replaces the most frequent *symbol pair* in S first. In the present, this does not guarantee the best results. I examined other two variations of proposed method which select a *symbol pair*(instead of $Pmax(S)$) independently of the *number of appearance*.
(a) A method uses the former *symbol pair* in stored order in S and
(b) the other method uses the encountered *symbol pair* while tracing hash space in stored order repeatedly. These do not select *symbol pair* whose *number of appearance* is less than 2.

But the performances of these two methods for TCC were approximately 14% and 28% worse than the proposed method, respectively.

Compression ratios of some existing methods are shown in Table 6. By the data of Burrows et al.[9], we can say its processing time is near to gzip's.

Execution time of encoding of proposed method is not short. But decoding time is short because its process is similar to LZW except for treatment of *enrollment directions* and use of Arithmetic coding.

Table 6 Compression performance for TCC

Method	Average comp. ratio [bits/char]	Total encode time [sec]	Total decode time [sec]
LZ78[1]*	4.279	7.6	4.3
by Gage[16, 17]	4.110	67.9	2.2
compress[3]	3.632	4.5	3.0
LZW[2, 20]	3.610	8.6	4.5
gzip[21]	2.708	15.3	2.9
comp-2[8] (with 3-th order)	2.540	83.9	88.5
comp-2[8] (with 4-th order)	2.467	121.3	125.9
comp-2[8] (with 5-th order)	2.502	103.5	107.4
Proposed	2.460	57.1	9.0

*Program was prepared by us.

9 Conclusions

A data compression method based on replacement of *symbol pairs* are described. Encoder reads whole input data at first. It sends directions of enrollment instead of code table. Processing time of encoding and decoding are order of data size. Decoding can be done by one pass process. Decoding is relatively fast compared to the encoding.

Themes of my future study are theoretical analysis of coding performance and application to image data compression.

Acknowledgment

I express thanks to Prof. Sadayuki Murashima (Kagoshima Univ.) for discussion, and to Prof. Hirosuke Yamamoto (Tokyo Univ.), Mr. Mitsuharu Arimoto (Tokyo Univ. Doctor Course Student), Prof. Tsutomu Kawabata (Univ. of Electro-Communications) and Mr. Mitsugu Kurizono (Kagoshima Univ., at that time avobe all) for offering useful information on this study.

References

1) J. Ziv, and A. Lempel, “Compression of Individual Sequences via Variable-rate Coding,” IEEE Trans Inf. Theory, vol.24, no.5, pp.530–536, 1978.

2) T.A. Welch, “Technique for High-performance Data Compression,” Computer, vol.17, no.6, pp.8–19, 1984.

3) P. Jannesen, D. Mack, J. Orost, J.A. Woods, K. Turkowski, S. Davies, J. McKie, and S.W. Thomas, Compress Version 4.2.3, Anonymous ftp from gatekeeper. dec.com: /pub/misc/ ncompress-4.2.3 .

4) H. Yokoo, “An Ziv-Lempel coding Scheme for Universal Source Coding,” IEICE Trans. Fundamentals, J68-A, no.7, pp.644–671, July 1985, in Japanese.

5) H. Yamamoto, and K. Nakata, “On the Improvement of Ziv-Lempel code and its Evaluation by Simulation-[II],” IEICE Tech. Report, CS84-135, pp.1–8, Jan. 1985, in Japanese.

6) T.C. Bell, J.G. Cleary, and I.H. Witten, “Text Compression,” Prentice-Hall, New Jersey, 1990.

7) J. Ziv, and A. Lempel, “A Universal Algorithm for Sequential Data Compression,” IEEE Trans. Inf. Theory, vol.23, no.3, pp.337–343, 1977.

8) M. Nelson, “Arithmetic coding and statistical modeling,” Dr. Dobbs Journal, Feb. 1991, Anonymous ftp from ftp.web.ad.jp: /pub/mirrors/Coast/msdos/ddjmag/ ddj9102.zip .

9) M. Burrows, and D.J. Wheeler, “A Block-sorting Lossless Data Compression Algorithm,” System Research Center Research Report, 124, May 1994.

10) H. Nakamura, and S. Murashima, “The Data Compression based on Concatenation of Frequent Code neighbor,” Proc. 14-th Sympo. on Inf. Theory and Its Appl., pp.701–704, Dec. 1991, in Japanese.

11) H. Nakamura, and S. Murashima, “The Data Compression based on Concatenation of Neighboring Characters Which Emerge Repeatedly,” IEICE Trans. Fundamentals, vol.J79-A, no.7, pp.1319-1323, July 1996, in Japanese

12) H. Nakamura, and S. Murashima, “Data Compression by Concatenations of Symbol Pairs,” Proc. IEEE Intr. Sympo. on Inf. Theory. and Its Appl., pp.496–499, Sept. 1996.

13) C.G. Nevill-Manning, I.H. Witten, and D.L. Maulsby, “Compression by Induction of Hierarchical Grammars,” Proc. Data Comp. Conf., pp.244–253, 1994.

14) C.G. Nevill-Manning, and I.H. Witten, “Detecting sequential structure,” Proc. Programming by Demonstration Workshop, Machine Learning Conf., pp.49–56, 1995.

15) C.G. Nevill-Manning, and I.H. Witten, “Compression and explanation using hierarchical grammars,” Computer Journal, Vol.40, No.2/3, pp.103–116, 1997.

16) P. Gage, “A New Algorithm for Data Compression,” The C Users Journal, vol.12, no.2, pp.23–38, Feb. 1994.

17) P. Gage, translated by A. Kida, “A New Algorithm for Data Compression,” C Magazine, vol.7, no.1, pp.21–27, Jan. 1995.

18) Y. Nagayama, S. Ito, and T. Hashimoto, “A Lossless Data Compression Algorithm Based on Digram,” Proc. 18-th Sympo. on Inf. Theory and Its Appl., pp.573–576, Oct. 1995, in Japanese.

19) I.H. Witten, R.M. Neal, and J.G. Cleary, “Arithmetic coding for data compression,” Commun. ACM, Vol.30, No.6, pp.520–540, June 1987.

20) X1 Player’s Zun, “Data Compression Program using Lempel-Ziv method,” I/O, vol.13, no.5, May 1988, in Japanese.

21) J. Gailly, Gzip Version 1.2.4, Anonymous ftp from prep.ai.mit.edu: /pub/gnu/gzip-1.2.4.tar.gz .