Data Compression by Concatenations of Symbol Pairs

Hirofumi Nakamura† and Sadayuki Murashima‡

† Computer Center, Miyakonojo National College of Technology, 473-1 Yoshio-Cho, Miyakonojo-Shi, 885 Japan.
Email: ccnakamu@cc.miyakonojo-nct.ac.jp

‡ Faculty of Engineering, Kagoshima University, 1-21-40 Korimoto, Kagoshima-Shi, 890 Japan
Email: mura@ics.kagoshima-u.ac.jp

Abstract — **For lossless data compression, we propose a method which has analysis and cutting of input symbol string. The analysis is based on concatenating symbol pair appearing repeatedly. Encoding and decoding can be executed in time $O(N)$, where N is the length of compression object.**

I. Introduction

Compression method based on parsing information of symbol string was proposed by Ziv and Lempel[1][2]. These encoders have the following redundancy:
(1) even if a certain symbol string appears repeatedly, each appearance is cut in various ways, and
(2) when N is finite, many enrolled symbol strings are not used later[2].
We tried to improve them as follows[3][4]:
(1) encoder divides symbol strings between the symbols which contact with lower frequency, and
(2) it does not enroll unnecessary symbol strings.

II. Encoding

We use following terms in this paper.
Symbol pair means a pair of two symbols. We express Symbol pair i j as i j merely.
S is the symbol string under analysis.
We express count of a symbol pair in S as *number of appearance*. We express the greatest *number of appearance* of S as $Nmax(S)$.
We express *most frequent symbol pair* as $Pmax(S)$. Here, $Pmax(S)$ is a symbol pair whose *number of appearance* is $Nmax(S)$. If the candidates of *most frequent symbol pair* exist more than or equal to 2, one pair should be chosen as $Pmax(S)$ by a certain rule.
In order to describe the processes of replacements of symbol string in this paper, we use production rules similar to Nevill-Manning et al.[5].
Alphabet of compression object corresponds to terminal symbol. The symbol pair correspond to the digram of their paper[5]. Code table is equivalent to collection of production rules.

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

A. Analysis of Input Symbol String

While the same symbol pair exists at least twice in S, encoder repeats following two operations:
(1) Encoder finds $Pmax(S)$.
(2) Encoder concatenates all $Pmax(S)$ in S. After that encoder treat it as a new symbol. (Encoder replaces all $Pmax(S)$ in S by new single symbol. We express the new symbol as *enrolled symbol.*)

Enrolled symbol is equivalent to non-terminal symbol of production rule.

For example, we consider the compression object
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}.
On S, *number of appearance* of symbol pair a d is the greatest number 4 (=$Nmax(S)$.) At first, we consider a d is equivalent to one new symbol A. Encoder replaces all a d (=$Pmax(S)$) in S by A. Then
S::= a c a b b A c A d b a A d c a A d b
A::= a d
is given. Next, current $Pmax(S)$=A d is replaced by B. Then
S::= a c a b b A c B b a B c a B b
A::= a d , B::= A d
is given. Next, current $Nmax(S)$ is 2. *Number of appearance* of c a and B b is 2. We assume that c a is selected from these as $Pmax(S)$. Encoder replaces c a by 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
is given. At last B b is replaced by D. Next, encoder finishes analysis, because $Nmax(S)$ is 1. As a result, we obtain
S::= a C b b A c D a B C D
A::= a d , B::= A d , C::= c a , D::= B b .

B. Cutting of Input Symbol String

Several methods exist in order to code analyzed result in sequential symbol stream. We show two of them.
At first, we show a simple way to send code table(we express it as SCT.)
For every *enrolled symbol*, encoder sends two symbols that are produced directly from *enrolled symbol.* After the code table, encoder sends analyzed result like
a d A d c a B b e a C b b A c D a B C D e ,
where e is the alphabet(equivalents to non-terminal symbol) that is introduced to indicate an end of sequence.

Next we show another way that does not send code table (we express it as SED.) It does self-referencing cutting, and sends only directions of enrollment instead of code table.

For all of *enrolled symbols*, at the place of its first appearance, encoder change back the *enrolled symbol*

to the symbol pair that is directly produced from the *enrolled symbol*. At this time encoder puts symbol 2 (we call this *enrollment direction*) immediately before the place. This symbol 2 indicates that decoder should enroll symbol pair as a new symbol.

Encoder newly assigns *enrolled symbol for output* to the symbol pair. Ordinal numbers of *enrolled symbol for output* can conflict with *enrolled symbols* of analyzed result. After the first appearance, *enrolled symbols* are replaced by assigned *enrolled symbols for output*.

And encoder puts symbol 1 (we call it *enrollment direction*, too) immediately before all the symbol in S except symbol 2. *Enrollment direction* 1 indicates that decoder does not need to enroll.

In a former example, each *enrolled symbols for output* mean

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

And output is

S::= 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' e .

In following bit stream generation, we call *enrolled symbol for output* merely *enrolled symbol*.

C. Bit Stream Generation

We examined three methods: the method that takes $\lceil \log C \rceil$ bits for code wards when current code table size is C, the method[6][7][8] that makes code words on the basis of complete binary tree(we express it as CBT) and adaptive arithmetic coding[9]. We express these generation methods as GLOG, GCBT and GAC, respectively.

On GAC, we memorize frequency and accumulation values in a CBT. A frequency value is memorized on a leaf[9]. All Nodes except leafs memorize a total value of the child node[9]. Accumulation value is provided by adding each right brother's value on a course following from a leaf to the root (if right brother does not exist, zero is added.) We prepared two binary trees; one for *enrollment direction* and another for *enrolled symbol*. We unified output by doing numeration using common variable to affect output for both data (*enrollment directions* and *enrolled symbols*.)

III. Decoding

At the decoding side, according to *enrollment direction*, decoder can reproduce code table of *enrolled symbol for output*, and can re-construct original symbol string.

IV. Algorithm

We show fundamental encoding and decoding algorithms with SED in Fig. 1.
We show meanings of major symbols in the following.

K: Number of alphabet. K is 256 for Byte data. A symbol of ordinal number K is used to indicate an end of sequence.
RepeatCheck: Judgment of exit analysis.
Enroll(T,ci,cl,cr): Enroll cl cr to code table T as *enrolled symbol* ci.
Replace(S,cl,cr,ci): Replace every cl cr in S by symbol ci.
WriteDirection(c): Output c (*enrollment direction.*)
WriteSymbol(c) : Output c (*enrolled symbol for output.*)
SymbolForOutput(c): Return *enrolled symbol for output* assigned for *enrolled symbol* c.
Left(c), Right(c): Left and right side of symbol pair that enrolled as *enrolled symbol* c.
Append(T,ci,co): Assign co (*enrolled symbol for output*) to ci (*enrolled symbol.*)
ReadDirection: Input of *enrollment direction.*
ReadSymbol: Input of *enrolled symbol for output.*

```
procedure Encode;
begin
  Initialize; Readfile(S); Cin:= K;
  while RepeatCheck do begin
    cl cr:=Pmax(S); Cin:=Cin + 1;
    Enroll(T,Cin,cl,cr); Replace(S,cl,cr,Cin);
  end;
  Cout:= K;
  for ci:=each element of S do Write(ci);
  Write(K);
end;
function RepeatCheck;
begin return( Nmax(S) >= 2 ) end;
procedure Write(ci);
begin
  if ci <= K then
    WriteDirection(1); WriteSymbol(ci);
  else if SymbolForOutput(ci)<>nil then
    WriteDirection(1); WriteSymbol(SymbolForOutput(ci));
  else
    WriteDirection(2);
    Write(Left(ci)); Write(Right(ci));
    Cout:=Cout + 1; Append(T,ci,Cout);
  endif;
end;
```

(a)

```
procedure Decode;
begin
  Initialize; Cout:= K;
  while Read <> K do ;
end;
function Read;
begin
  t:=ReadDirection;
  if t = 1 then
    co:=ReadSymbol; WriteString(co);
  else
    cl:=Read; cr:=Read;
    Cout:=Cout + 1; Enter(T,Cout,cl,cr); co:=Cout;
  endif;
  return(co);
end;
procedure WriteString(co);
begin
  if co < K then
    WriteAlphabet(co);
  else if co > K then
    WriteString(Left(co)); WriteString(Right(co));
  endif;
end;
```

(b)

Fig. 1: Algorithms of Encoder and Decoder
(a)Encoding Algorithm (b)Decoding Algorithm

V. Time Complexity

To memorize *numbers of appearance* of symbol pairs, we use hashing using two keys (symbol pair which constitutes *enrolled symbol.*) If the size of hash table is big enough, complexity of one access is $O(1)$.
In order to find $Pmax(S)$, we prepare two-way lists for each *number of appearance.* One two-way list connects hash table's buckets whose *numbers of appearance* are the same.
On each replacement of symbol pair, encoder renews *numbers of appearance* in hash table, and moves buckets among the two-way lists.

Activities of one replacement operation is completed in time $O(1)$. The length of S decreases by 1 by replacement operation of one place in S. Number of replacement operation doesn't exceed N. On these account, the complexity of coding is $O(N)$.

Complexity of decoding mainly depends on the decision of original symbols. It is $O(N)$.

When the Arithmetic Coding with CBT is used, complexity of Arithmetic Coding is $O(R \log C)$, where R denotes cutting number and C denotes number of symbols including *rolled symbol.* It is equal to order of output length except a code table. We do not know orders of R and C yet. But, if output of coding does not expand in extreme, output length is $O(N)$.

VI. Comparison with Similar Methods

In the method of Nevill-Manning et al.[5], S begins with null, and input symbols are added to S by one symbol. On each adding, encoder finds a symbol pair that appears twice in S and production rules made already. If such a symbol pair exists, encoder makes new production rule, and replaces symbol pair by the new non-terminal symbol. It does integration and abolition of production rules if possible.

For the example mentioned above, next result is provided

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 .

This method can output after having analyzed whole of compression object. It uses Arithmetic Coding for bit stream generation.

Proposed method and their method sometimes give the same analyzed result. For example, input string treated in their paper[5] gives the same result.

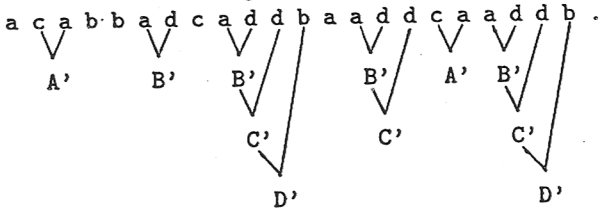
In the method of Nagayama et al.[10], S begins with null, and input symbols are added to S by one symbol. On each adding, encoder looks for a symbol pair that appear twice in only S. If such a symbol pair exists, encoder replaces second (and after second, too) appearance by a symbol whose ordinal number corresponds to the location of the first appearance in S.

This method does not send code table, and does not need to send any direction for enrollment. This is the adaptive method that can output while inputting. By this method, next analyzed result is provided

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 , ...

Numerals are used as non-terminal symbols here.
This method uses the encoder for positive integers[11] for bit stream generation. It does not use appearance frequency of output symbols.

We show the structure of result analyzed by proposed method in the following



Proposed method always watches the whole S, and production rules are settled in order of *number of appearance.* Production rules only increase, and need not be modified during the analysis.
Proposed method replaces *most frequent symbol pair* first. In the present algorithm, this does not guarantee of best results.
We examined a method which selects symbol pair for replacement independent of the *number of appearance.* But the performance of this method was approximately 10 percent worse than the proposed method.

VII. Experimental Result

We experimented for 14 files of Calgary Text Compression Corpus (we express it as TCC) [9]. Unit of measure of compression ratio is bits/char. Processing time (second) of coding and decoding is a value measured on the workstation Fujitsu S-4/20H. They are average values of 10 trials. When we do not mention file name later on, compression ratio is the average value of TCC 14 files, and processing time is the total

Tab. 1: Compression Performance for TCC 14 Files by Proposed Method (SED+GAC)

file name	size (bytes)	ratio	encode time	decode time
bib	111261	2.175	2.5	0.3
book1	768771	2.629	15.4	2.5
book2	610856	2.265	11.8	1.7
geo	102400	4.454	2.7	0.4
news	377109	2.611	7.9	1.2
obj1	21504	3.871	0.9	0.1
obj2	246814	2.577	5.2	0.8
paper1	53161	2.625	1.5	0.2
paper2	82199	2.573	2.0	0.3
pic	513216	0.802	6.5	0.6
progc	39611	2.632	1.3	0.2
progl	71646	1.847	1.7	0.2
progp	49379	1.758	1.4	0.2
trans	93695	1.584	2.2	0.3
average or total		2.472	62.9	9.1

value of TCC 14 files.
Compression ratio of proposed method is good compared with the methods of Nevill-Manning et al. and Nagayama et al.

Tab. 2: Compression Ratio for TCC

method	sending code table	not sending code table
Nevill-Manning et al.[5]	3.13[5]	2.70[5]
Nagayama et al.[10]	-	3.29[10]
Prop(SCT/SED+GLOG)	3.328	2.871
Prop(SCT/SED+GCBT)	3.254	2.806
Prop(SCT/SED+GAC)	3.018	2.472

Tab. 3: Compression Performance for TCC

method	ratio	encode time	decode time
LZ78[2](GLOG)*	4.279	7.6	4.3
LZW[12][13](GLOG)	3.610	8.6	4.5
compress[14]	3.632	4.5	3.0
gzip[15]	2.708	15.3	2.9
comp-2[16] (with 4th order)	2.467	145.6	140.0
Block-sorting[17]	2.428[17]	-	-
Prop(SED+GLOG)	2.871	58.6	3.4
Prop(SED+GCBT)	2.806	60.0	3.5
Prop(SED+GAC)	2.472	62.9	9.1
Prop(SED+GAC+NAT)	2.466	63.3	9.1
Prop(SED+GAC+EST)	2.465	62.9	9.0
Prop(SED+GAC +NAT+EST)	2.459	63.3	9.0

*:Program was prepared by us.
NAT: Encoder and decoder enroll *enrolled symbols for output* based on list structure instead of binary tree (number of child is limited to best value.)
EST: RepeatCheck decides its return value by rough estimation of effect of replacement.

VIII. Conclusions

We described a data compression method based on concatenation of symbol pair. Encoder reads whole input data at first and enrolls only necessary symbol strings into code table. Encoder does not send a code table to decoder. It sends only directions of enrollment instead of code table. Processing time of encoding and decoding is $O(N)$. Decoding is fast. Now we are trying adaptive method that can output while inputting.

Themes for future study are theoretical analysis of performance and modification of the proposed method so that it includes concatenations of two symbols which exist apart.

References

[1] J.Ziv and A.Lempel: "A Universal Algorithm for Sequential Data Compression," *IEEE Information Theory*, 23, 3, pp.337-343, 1977.

[2] J.Ziv and A.Lempel: "Compression of Individual Sequences via Variable-rate Coding," *IEEE Information Theory*, 24, 5, pp.530-536, 1978.

[3] H.Nakamura and S.Murashima: "The Data Compression based on Concatenation of Frequentative Code neighbor," *The 14th Sympo. on Information Theory and Its Application*, pp.701-704, Dec. 1991, in Japanese.

[4] H.Nakamura and S.Murashima: "The Data Compression based on Concatenation of Neighboring Characters Which Emerge Repeatedly," *IEICE Trans. (A)*, in Japanese, in print.

[5] C.G. Nevill-Manning, I.H. Witten and D.L. Maulsby: "Compression by Induction of Hierarchical Grammars," *Proc. of Data Comp. Conf.*, pp.244-253, 1994.

[6] H.Yokoo: "An Improved Ziv-Lempel coding Scheme for Universal Source Coding," *IEICE Trans. (A)*, J68-A, no.7, pp.644-671, Jul. 1985, in Japanese.

[7] 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.

[8] P.Tischer: "A Modified Lempel-Ziv-Welch Data Compression Scheme," *Austrian Computer Science Communications*, vol.9, no.1, pp.262-272, 1987.

[9] T.C.Bell, J.G.Cleary and I.H.Witten: Text Compression, Printice-Hall, 1990.

[10] Y.Nagayama, S.Ito and T.Hashimoto: "A Lossless Data Compression Algorithm Based on Digram," *The 18th Sympo. on Information Theory and Its Application*, pp.573-576, Oct. 1995, in Japanese.

[11] H.Yamamoto and H.Ochi, "A New Asympotically Optimal Code for the Positive Integers," *IEEE Trans. on Inform. Theory*, vol.IT-37, no.5, pp.1420-1429, sep. 1991.

[12] T.A.Welch: "Technique for High- performance Data Compression," *Computer*, 17, 6, pp.8-19, 1984.

[13] X1 Player's Zun: "Data Compression Program using Lempel-Ziv method," *I/O*, 13, 5, 1988, in Japanese

[14] 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 gatekeeper. dec.com: /pub/misc/ ncompress-4.2.3

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

[16] Mark 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

[17] M.Burrows and D.J.Wheeler: "A Block-sorting Lossless Data Compression Algorithm," *System Research Center Research Report*, 124, May 1994.