Entropy **2011**, *13*, 53-63; doi:10.3390/e13010053

OPEN ACCESS

entropy

ISSN 1099-4300

www.mdpi.com/journal/entropy

Communication

A Unique Perspective on Data Coding and Decoding

Wen-Yan Wang

College of Electronic Engineering, Guangxi Normal University, Yucai Road 15, Guilin 541004, China; E-Mail: wwy@mailbox.gxnu.edu.cn

Received: 25 November 2010; in revised form: 17 December 2010 / Accepted: 18 December 2010 / Published: 27 December 2010

Abstract: The concept of a loss-less data compression coding method is proposed, and a detailed description of each of its steps follows. Using the Calgary Corpus and Wikipedia data as the experimental samples and compared with existing algorithms, like PAQ or PPMstr, the new coding method could not only compress the source data, but also further re-compress the data produced by the other compression algorithms. The final files are smaller, and by comparison with the original compression ratio, at least 1% redundancy could be eliminated. The new method is simple and easy to realize. Its theoretical foundation is currently under study. The corresponding Matlab source code is provided in the Appendix.

Keywords: loss-less compression; information theory; universal information

PACS Codes: 70

1. Overview

Data compression is the art of reducing the number of bits needed to store or transmit data. Compression can be either lossless or lossy. Lossless compressed data can be decompressed to exactly its original value. An example is Morse code, in which each letter of the alphabet is coded as a sequence of dots and dashes. The most common letters in English, like E and T, have the shortest codes, while the least common ones like J, Q, X, and Z are assigned the longest codes [1].

Information theory places hard limits on what can and cannot be compressed losslessly. Lossless compression uses the statistical redundancy of data to compress and later recover original information without distortion. The volume of data in various information systems has rapidly increased, so transmitting data rapidly and easily and storing data without losses have become primary problems in data processing. Lossless data compression technology is an important method to solve this problem [2,3] that has long been applied to many fields, but compressibility is affected by the statistical redundancy of data. Currently prevailing algorithms like WinZip, WinRAR, and other similar tools fail to recompress information that has already been compressed [4–6].

A method for lossless data coding and decoding is proposed in this communication. The new method has passed several experimental tests. Compared with currently available technologies, the method can not only compress source data without losses, but also further compresses data that have already been compressed with other compression tools, decreasing the space occupied by the compressed files and accordingly removing at least 1% redundancy based on the original compressibility (the redundancy removed differs with the data redundancy of the files). Furthermore, the new method features a simple algorithm and is easy to carry out. Chinese patents for this technology have already been applied for [7].

2. Concept of the Coding Algorithm

The concept of this coding algorithm is simple. As we know, if the data consists of contiguous “0s” or “1s”, there is little information entropy, and the data redundancy is huge. This redundancy can be eliminated by some lossless compression algorithms. In the real world, the data that we use in computers consists of contiguous “0s” and “1s”, for example, the data “00101110001100000100” consists of one “0”, two “00s”,one “000”,one “00000”, two “1s”, one “11” and one “111”. In the example, the point is that it has no “0000” sequence, so we can represent “00000” with “0000”. Using this method, 1 bit of the data can be removed. In some communication experiments, it has been found that if the data is source data, and has no process coding, this method is quite workable, and can eliminate more redundancy. If the data is random or processed coding, we can also eliminate a little redundancy, even if only 1 bit. Of course, because of the limit of Shannon’s information theory, our method does not work yet on data which has no redundancy. The concept of this coding method is new and different from the other algorithms, so its theoretical foundation needs to be researched in depth. This is our goal in future work.

3. Detailed Coding Algorithm Implementation

(1) The source data is read in binary mode. The binary sequence is then stored in new Array A. The first binary code of Array A is stored in Variable a.

(2) The length “from 0 until the next bit is 1” is substituted in Array A for all “0” in “from 0 until the next bit is 1”. The length “from 1 until the next bit is 0” is substituted in Array A for all “1” in “from 1 until the next bit is 0.” The sequence is then obtained for new Array B.

(3) The different elements in Array B are sorted and stored in new Array C in their order of occurrence.

(4) The elements of Array C are sequenced in ascending order and then stored in new Array D.

(5) The digit n is substituted for the n-th element in Array D. The obtained sequence is stored in new Array E.

(6) The elements of Array D correspond one for one to those of Array E, and all elements of Array B are replaced. The obtained sequence is then stored into new Array F.

(7) All elements of Array F are inversely transformed. Unlike in Step (1), the first code of Array F is the code stored in Variable a. The inversely transformed BC sequence is then stored in new Array G.

(8) The elements in Array D are processed by "deducting the adjacent previous item from the next item", and the results are stored in new Array H. The first position where the element is equal to or bigger than 2 in Array H is determined, and the position in Array H is set to n. All elements before the position n in Array D are deleted, and the remaining elements are stored in new Array I in order.

(9) Array I and Array G are saved as binary files; this is the result of source file lossless compression, where:

In Step (5), the digit n begins from 1.

In Step (8), the element position n begins from 1.

In Step (1), the source file can be any data that have not been compressed, *i.e.*, data in common file formats, and can also be data that have already been compressed with PAQ, WinRK, WinZip, WinRAR, or other tools.

4. Detailed Decoding Algorithm Implementation

(1) The BC sequence in Array G is read. The first binary code of Array G is stored in Variable g.

(2) The length "from 0 until the next bit is 1" is substituted in Array A for all "0" in "from 0 until the next bit is 1". The length "from 1 until the next bit is 0" is substituted in Array A for all "1" in "from 1 until the next bit is 0". The obtained sequence is then stored in new Array F.

(3) The different elements in Array F are sorted and stored in new Array E in their order of occurrence.

(4) The elements in Array E are sequenced in ascending order, and the sequence is stored in new Array D.

(5) Array I is read, and the number of elements in Array I is set to i. The elements in Array I are substituted for the i elements in Array D from the end. The obtained sequence is stored in new Array C.

(6) The elements in Array D correspond one for one to those of Array C. All elements in Array F are replaced. The obtained sequence is stored in new Array B.

(7) All elements in Array B are compared with those in Step (1). The first code of Array B is the code stored in Variable g, and the inversely transformed BC sequence is stored in new Array A. Array A is saved as a source data.

5. Illustration

Assume that there is a data W to code.

5.1. The Lossless Compression for W Is Given as Follows:

(1) The file is read in binary mode (A = {00100101110100110000011101011110}, a = 0);

(2) The length "from 0 until the next bit is 1" is substituted in Array A for all "0" in "from 0 until the next bit is 1". The length "from 1 until the next bit is 0" is substituted in Array A for all "1" in "from 1 until the next bit is 0." The obtained sequence is then stored in new Array B (B = {21211311225311151});

(3) The different elements in Array B are sorted and stored in new Array C in their order of occurrence (C = {2135});

(4) The elements in Array C are sequenced in ascending order, and the obtained sequence is stored in new Array D (D = {1235});

(5) The digit n is substituted for the n-th element in Array D, and the obtained sequence is stored into new Array E (E = {1234});

(6) The elements in Array D correspond one for one to those in Array E. All elements in Array B are replaced, and the obtained sequence is stored in new Array F (F = {21211311224311141});

(7) All elements in Array F are inversely transformed unlike in Step (1). The first code of Array F is the code stored in Variable a, and the inversely transformed BC sequence is stored in new Array G (G = {0010010111010011000011101011110});

(8) The elements in Array D are processed by "deducting the adjacent previous item from the next item", and the result is stored in new Array H. The first position where the element is equal to or bigger than 2 in Array H is determined, and the position in Array H is set to n. All the elements before the position n in Array D are deleted, and the remaining elements are stored in new Array I in order (H = {112}, I = {5});

(9) Array I and Array G are saved as binary files; this is the result of source file lossless compression.

Analysis: the comparison between Array A and Array G indicates that the number of binary codes in Array G is two less than that in Array A; *i.e.*, the space occupied by Array G is 2 bits less than that occupied by Array A. In this example, the length of Sequence A of the source file is 33 bits and that of Sequence G of the compressed file is 31 bits. This method thus removes 6% redundancy from Sequence A and saves 6% space.

5.2. The Decompression of the Compressed File W Is Given as Follows:

(1) The BC sequence in Array G of the compressed data is read (G = {0010010111010011000011 101011110}, g = 0);

(2) The length "from 0 until the next bit is 1" is substituted in Array A for all "0" in "from 0 until the next bit is 1". The length "from 1 until the next bit is 0" is substituted in Array A for all "1" in "from 1 until the next bit is 0". The obtained sequence is then stored in new Array F (F = {21211311224311141});

(3) The different elements in Array F are sorted and stored in new Array E in their order of occurrence (E = {2134});

(4) The elements in Array E are sequenced in ascending order, and the sequence is stored in new Array D (D = {1234});

(5) Array I is read, and the number of elements in Array I is set to i. The elements in Array I are replaced for the i elements in Array D from the end. The obtained sequence is stored in new Array C (I = {5}, i = 1, C = {1235});

(6) The elements in Array D correspond one for one to those of Array C. All elements in Array F are replaced, and the obtained sequence is stored in new Array B (B = {21211311225311151});

(7) All elements in Array B are inversely transformed unlike in Step (1). The first code of Array B is the code stored in Variable g. The inversely transformed BC sequence is stored in new Array A. Array A is saved as a source data (A = {00100101110100110000011101011110}).

6. Coding Source Code in Matlab

Coding source code in Matlab is given in the Appendix. More information can be obtained on our website [8].

7. Benchmarks

A data compression benchmark measures compression ratio over a data set, and sometimes memory usage and speed on a particular computer. Some benchmarks evaluate only file size, in order to avoid hardware dependencies. Compression ratio is often measured by the size of the compressed output file, or in bits per character (bpc), meaning compressed bits per uncompressed byte. In either case, smaller numbers are better: 8 bpc means no compression, while 6 bpc means 25% compression or 75% of original size [1].

7.1. Calgary Corpus

The Calgary corpus is the oldest compression benchmark still in use. It was created in 1987 and described in a survey of text compression models in 1989 [9]. It consists of 14 files with a total size of 3,141,622 bytes. These files are listed in Table 1:

Table 1. Calgary Corpus files and their descriptions.

Size (bytes)	File Name	Description
111,261	BIB	ASCII text in UNIX “refer” format—725 bibliographic references.
768,771	BOOK1	Unformatted ASCII text—Hardy: Far from the Madding Crowd.
610,856	BOOK2	ASCII text in UNIX “troff” format—Witten: Principles of Computer Speech.
102,400	GEO	32 bit numbers in IBM floating point format—seismic data.
377,109	NEWS	ASCII text—USENET batch file on a variety of topics.
21,504	OBJ1	VAX executable program—compilation of PROGP.
246,814	OBJ2	Macintosh executable program—“Knowledge Support System”.
53,161	PAPER1	UNIX “troff” format—Witten, Neal, Cleary: Arithmetic Coding for Data Compression.
82,199	PAPER2	UNIX “troff” format—Witten: Computer (in) security.
513,216	PIC	1728×2376 bitmap image (MSB first): text in French and line diagrams.
39,611	PROGC	Source code in C—UNIX compress v4.0.
71,646	PROGL	Source code in Lisp—system software.
49,379	PROGP	Source code in Pascal—program to evaluate PPM compression.
93,695	TRANS	ASCII and control characters—transcript of a terminal session.

7.2. Large Text Compression

The Large Text Compression Benchmark consists of a single Unicode encoded XML file containing a dump of Wikipedia text from March 3, 2006, truncated to 1,000,000,000 bytes (the enwik9 file). Its stated goal is to encourage research into artificial intelligence, specifically, natural language processing. As of February 2010, 128 different programs (889 including different versions and options) had been evaluated for compressed file size (including the decompression program source or executable and any other needed files as a zip archive), speed, and memory usage. The benchmark is open, meaning that anyone can submit results [1].

Programs are ranked by compressed size with options selecting maximum compression where applicable. The best result obtained is 127,784,888 bytes by Dmitry Shkarin on July 21, 2009 for a customized version of Durilca using 13 GB memory. It took 1,398 seconds to compress and 1,797 seconds to decompress using a size-optimized decompression program on a 3.8 GHz quad core Q9650 with 16 GB memory under 64 bit Windows XP Pro. The data was preprocessed with a custom dictionary built from the benchmark and encoded with order 40 PPM. Durilca is a modified version of PPMonstr by the same author. PPMonstr is a slower but better at compressing than the PPMd program which is used for maximum compression in several archivers such as RAR, WinZip, 7-Zip, and FreeArc [1].

7.3. Maximum Compression

The maximum compression benchmark [10] has two parts: a set of 10 public files totaling 53 MB, and a private collection of 510 files totaling 301 MB. In the public data set (SFC or single file compression), each file is compressed separately and the sizes added. Programs are ranked by size only, with options set for best compression individually for each file. The set consists of the following 10 files as in Table 2:

Table 2. Maximum Compression test files and its description (unit: byte).

Size	File name	Description
842,468	a10.jpg	A high quality 1152×864 baseline JPEG image of a fighter jet.
3,870,784	acrord32.exe	X86 executable code—Acrobat Reader 5.0.
4,067,439	english.dic	An alphabetically sorted list of 354,941 English words.
20,617,071	fp.log	Web server log, ASCII text.
3,782,416	mso97.dll	X86 executable codes from Microsoft Office.
4,168,192	ohs.doc	Word document with embedded JPEG images.
4,149,414	rafale.bmp	1356×1020 16 bit color image in 24 bit RGB format.
4,121,418	vcfiu.hlp	OCX help file—binary data with embedded text.
2,988,578	world95.txt	ASCII text—1995 CIA World Factbook.

As of December 31, 2009 the top ranked program, giving a total size of 8,813,124 bytes, is PAQ8px, a context mixing algorithm with specialized models for JPEG images, BMP images, X86 code, text, and structured binary data. WinRK 3.1.2, another context mixing algorithm, is top ranked on four of the files (.txt, .exe, .dll, .pdf). WinRK uses a dictionary which is not included in the total size. 208 programs are ranked. Zip 2.2 is ranked 163rd, with a size of 14,948,761 bytes [10].

In the second benchmark or MFC (multiple file compression), programs are ranked by size, compression speed, decompression speed, and by a formula that combines size and speed with time scaled logarithmically. The data is not available for download. Files are compressed together to a single archive. If a compressor cannot create archives, then the files are collected into an uncompressed archive (TAR or QFC), which is then compressed. In the MFC test, paq8px is top ranked by size. FreeArc is top ranked by combined score, followed by NanoZip, WinRAR and 7-Zip. All are archivers that detect file types and apply different algorithms depending on the type [10].

7.4. Benchmark

Table 3 shows the experimental results using Durilca and PAQ8px to compress the English Wikipedia file and the Calgary Corpus. Table 4 shows the maximum compression benchmark result which used WinRK3.1.2 and PAQ8px.

Table 3. Experimental results with Durilca, PAQ8px and new coding (unit: byte).

	Original Size	Durilca	PAQ8px v69 -7	New Coding
calgary.tar	3,152,896	666,216	598,151	598,133
enwiki	1000,000,000	127,377,411	132,045,026	127,377,288

Table 4. Experimental results with WinRK, PAQ8px and new coding (unit: byte).

	Original Size	WinRK 3.1.2	PAQ8px v69 -7	New Coding
LOGFILE	20,617,071	271,628	254,008	271,611
ENGLISH TEXT	2,988,578	330,571	351,923	330,548
SORTED WORDLIST	4,067,439	393,704	384,073	393,688
HLP	4,121,418	415,522	395,343	415,504
MS-WORD DOC FILE	4,168,192	688,237	482,509	482,507
BITMAP	4,149,414	569,053	528,480	569,038
JPG/JPEG	842,468	812,700	637,114	812,684
EXECUTABLE	3,870,784	896,364	895,533	895,531
DLL (EXECUTABLE)	3,782,416	1,236,643	1,261,335	1,236,629
PDF	4,526,946	3,549,197	3,554,078	3,549,184

8. Discussion and Conclusions

The test data for the Large Text Compression Benchmark is the first 1,000,000,000 bytes of the English Wikipedia dump on March 3, 2006. It is 1.1 GB or 4.8 GB after decompressing with bzip2 [11]. Results are also given for the first 108 bytes, which is also used for the Hutter Prize [12]. These files can be downloaded from the prize website [12]. Durilca is the number one and can compress it to 127,377,411 bytes [13]. Now, using our algorithm, 123 bytes of redundancy can be eliminated in the 127,377,411 byte Durilca file.

This communication proposes a unique coding method. Its theoretical foundation is under study, but the method is already workable. It is different from other loss-less compression algorithms, so, it can compress the data which has processed by those other algorithms too. Using the compression benchmark samples, compared with the current technologies, the new method can not only compress source data without losses but also further compress data that have already been compressed by the other algorithms. Table 1 and 2 had been shown that at least 1% redundancy can be removed from the compressed data. As we know that the redundancy in the compressed data is hard to eliminate because of the information theory limit. Dr. Matt's PAQ is an excellent data compression algorithm, and ranks first in the compression field. As for the other algorithms, it is very hard to find the redundancy in the data after PAQ compression, but our algorithm can do it in some types of data, like the Calgary Corpus data, EXECUTABLE data or MS-WORD DOC FILE data. It has the same effect as WinRK. The new coding method also features a simple algorithm and is easy to carry out. The disadvantage is that the theoretical foundation still needs study, and is the subject of ongoing research.

Acknowledgements

This work was supported by the National Science Foundation of China (No. 30973418), Guangxi High school Buildup of Key Laboratories Project (No. 200912) and Guangxi Normal University 2009 Young Core Teacher Foundation. Thanks are also due to Matt Mahoney, Dmitry Shkarin, Malcolm Taylor and Werner Bergmans for their excellent works.

References

1. Mahoney, M. Data compression explained. Available online: http://mattmahoney.net/dc/dce.html (accessed on 1 December 2010).
2. Wang, W.-Y. A lossless compression method for JPEG based on shuffle algorithm. *IASP* **2009**, *1*, 131–132.
3. Wang, W.-Y. New method to analysis noise in speech signal. In *Proceedings of the Pacific-Asia Conference on Circuits (PACCS)*, Chengdu, Sichuan, China, 4 May 2009.
4. Wang, W.-Y. A novel loss-less compression method on JPEG. *China Patent 200810073769.0*. 17 February 2008.
5. Wang, W.-Y. *Modern Software Development Technology*; Atomic Energy Press: Beijing, China, 2010.
6. Yan, J.-W. *Digital Image Process-Matlab*; Beijing-National Defense Industry Press: Beijing, China, 2007.
7. Wang, W.-Y. New method to loss-less compress information. *China Patent 2010105154690*, 5 October 2010.
8. Wang, W.-Y. Data compression Lab 601 (in Chinese). Available online: http://www.ee.gxnu.edu.cn/compression/ (accessed on 1 November 2010).
9. Wikipedia. Calgary corpus. Available online: http://en.wikipedia.org/wiki/Calgary_Corpus (accessed on 23 February 2010).
10. Bergmans, W. Maximum compression. Available online: http:// www.maximumcompression.com (accessed on 3 November 2010).
11. Wikipedia. Wikimedia downloads. Available online: http://download.wikipedia.org/enwiki/20060303/enwiki-20060303-pages-articles.xml.bz2 (accessed on 12 September 2006).

12. Hutter, M. Prize for compressing human knowledge. Available online: http://prize.hutter1.net/ (accessed on 11 July 2010).
13. Mahoney, M. Large text compression benchmark. Available online: http://www.mattmahoney.net/dc/text.html (accessed on 17 June 2010).

Appendix: Matlab Code

```
function [I10]=xsc(I)
m=length(I);
imp=I(1);
I2=zeros(m,1);
k=1;
for i=1:m-1
I2(k)=I2(k)+1;
if xor(I(i),I(i+1))==1
  k=k+1;
end
end
if xor(I(m),I(m-1))==1
  I2(k+1)=1;
  k=k+1;
end
if xor(I(m),I(m-1))==0
  I2(k)=I2(k)+1;
end
k=find(I2==0,1,'first');
k=k-1;
clear I;
I3=zeros(k,1);
I3=I2(1:k);
clear I2;
Im=max(I3);
I4=zeros(Im,1);
for i=1:length(I3)
I4(I3(i))=I4(I3(i))+1;
end
I4t=I4;
I4t(find(I4t==0))=[];
if length(find(I4==0))==0
  fprintf('\n error !   \n\n')
end
if length(find(I4==0))>0

I5=find(I4(find(I4==0,1,'first'):length(I4))>0);
```

```
I5=I5-1;
Flag1dec=ceil(log2(length(I5)));
Flag1bin=dec2bin(Flag1dec-1,4);
%Flag1bin=str2num(Flag1bin(:));
Flag1bin=Flag1bin-48;
Flag1bin=Flag1bin';
Flag2bin=dec2bin(length(I5)-1,Flag1dec);
Flag2bin=Flag2bin-48;
Flag2bin=Flag2bin';
Flag3dec=ceil(log2(max(I5)));
Flag3bin=dec2bin(Flag3dec-1,5);
Flag3bin=Flag3bin-48;
Flag3bin=Flag3bin';
I6=dec2bin(I5-1,Flag3dec);
I7=[];
for i=1:length(I5)
I7=[I7 I6(i,1:Flag3dec)];
end

Flag4bin=I7-48;
Flag4bin=Flag4bin';
I8=[Flag1bin;Flag2bin;Flag3bin;Flag4bin];
Lc=0;
for i=1:length(I4t)
    Lc=Lc+I4t(i)*i;
end
Lcs=Lc+length(I8);
Flag5dec=8-mod(Lcs,8);
Flag5bin=dec2bin(Flag5dec,8);
Flag5bin=Flag5bin-48;
Flag5bin=Flag5bin';
I8=[Flag5bin;I8];
if Flag5dec==8
   Flag5dec=0;
end
Lcss=Lcs+8+Flag5dec;
CR=Lcss/m;

fprintf('\n rate  %f %% \n\n',CR*100);
First0Pos=find(I4==0,1,'first');
I9=I5;
I9m=length(I9);
I9=I9+First0Pos;
I9(1:I9m,2)=First0Pos:First0Pos+I9m-1;
```

```
for i=1:I9m
  I3(find(I3==I9(i,1)))=I9(i,2);
end
I10=zeros(Lc,1);
if imp==0
  i=2;
  j=I3(1)+1;
  endflag=0;
  while endflag==0
    I10(j:j+I3(i)-1)=1;
    if i+1>length(I3)
      break;
    end
    j=j+I3(i)+I3(i+1);
    i=i+2;
    if i>length(I3)
      break
    end
     end
end
if imp==1
  i=1;
  j=1;
  endflag=0;
  while endflag==0
    I10(j:j+I3(i)-1)=1;
    if i+1>length(I3)
      break;
    end
    j=j+I3(i)+I3(i+1);
    i=i+2;
    if i>length(I3)
      break
    end
      end
   end
I10=[I8;I10];
end
end
```