Achieving Better Compression Applying Index-based Byte-Pair Transformation before Arithmetic Coding

Jyotika Doshi
GLS Inst.of Computer Technology
Opp. Law Garden, Ellisbridge
Ahmedabad-380006, India

Savita Gandhi
Dept. of Computer Science;Guj. Uni.
Navrangpura
Ahmedabad-380009, India

ABSTRACT
Arithmetic coding is used in many compression techniques during the entropy encoding stage. Further compression is not possible without changing the data model and increasing redundancy in the data set. To increase the redundancy, we have applied index based byte-pair transformation (BPT-I) as a pre-processing to arithmetic coding. BPT-I transforms most frequent byte-pairs (2-byte integers). Here, most frequent byte-pairs are sorted in the order of their frequency and groups consisting of 256 byte-pairs are formed. Each byte-pair in a group is then encoded using two tokens: group number and the location in a group. Group number is denoted using variable length prefix codeword; whereas location within a group is denoted using 8-bit index. BPT-I is designed to be applied on any type of source; not necessarily text. More the number of groups considered during transformation, better is the compression. Experimental results have shown around 4.30% additional reduction in compressed file size when arithmetic coding is applied after byte-pair data transformation BPT-I.

General Terms
Data Compression, Algorithms

Keywords
Data Compression, better compression rate, index based byte-pair data transformation, data transformation as a pre-processing to arithmetic coding, arithmetic coding

1. INTRODUCTION
In current days, arithmetic coding [6, 8, 16] is the most preferred entropy coding technique used with most of the compression methods as it provides optimal entropy. Due to its entropy limitations, further improvement in compression is not possible. The only way to achieve better compression using arithmetic coding is to change the data model such that the data becomes more skewed. One way to achieve this is by applying data transformation.

Data transformation transforms data from one format to another. The purpose of a data transformation is here to re-structure the data to make it more compressible by a second-stage arithmetic coding compression algorithm. Here, our intention is to improve the overall compression rate as compared to what could have been achieved by using only arithmetic coding compression algorithm.

Authors of this paper have proposed Quad-Byte Transformation using Index (QBT-I) method [4] with similar purpose. The problem with QBT-I is in computing the frequency of 4GB possible quad-bytes. With byte-pair transformation, the maximum possible values are 65536 only. This makes it easier to compute frequencies of all byte-pairs using simple array data structure. It also enables the computation to be faster due to random access with array.

Like QBT-I [4], BPT-I is also independent of source type. It can be applied to any type of source; may be text, binary file, image, video or any other format. Arithmetic coding method is also applicable to any type of source. Both these transformation and compression methods are reversible, so it provides lossless compression. Moreover BPT-I is faster to execute.

BPT-I transforms most frequent byte-pairs (16-bit integers) considering byte-pair belonging to a group of size 256. Each byte-pair is encoded using group number and its position within a group.

This two-stage process of transformation and then compression is obviously going to be slower. But, this slowness is affordable since the transform truly skews the data source to fulfil our purpose of achieving more compression

2. LITERATURE REVIEW
Majority of the transformation techniques are seen to be source-type specific. Research work in star-based encoding techniques [1, 7, 11, 15], dictionary-based encoding techniques [12, 13, 14] and digram encoding techniques [5, 9, 17] are intended for text files. Another category is of data transformation techniques like DCT and wavelet used on image files.

Transformation techniques BWT [2, 10], BPE[5], digram encoding [17] and ISSDC [9] are intended for text files. However, they can be applied to any type of source.

Burrows Wheeler Transform (BWT) performs block encoding. For each block, BWT requires rotation-sorting-indexing. Thus it is very time consuming and requires better data structures for efficient pattern matching. Another problem with BWT is that it gives better compression only when it is combined with compression techniques Run Length Encoding (RLE) and Move-To-Front (MTF) encoding and then Huffman or arithmetic coding entropy coding.

Methods BPE (Byte Pair Encoding) [5], digram encoding [17] and ISSDC (Iterative Semi-Static Digram Coding) [9] will benefit more only when applied to small-alphabet source like text files having some unused symbols. Moreover, BPE and ISSDC use repetitive algorithms. So they are very time consuming. If the source size is large enough to be accommodated in main memory, it requires repetitive file scanning. Thus they are better only with small sized files and source with small-alphabet.

Most of the mentioned data transformation techniques may introduce some compression also. Their main purpose is

obviously to retain enough context and redundancy for later applied compression algorithms to be beneficial.

3. RESEARCH SCOPE

We saw a research scope in having transformation technique that is applicable to any type of source. It should also introduce redundancy in the data so that transformed file is more compressible. Additionally if it can add compression while transforming the source, it will result in faster compression due to smaller file size.

Our prime purpose is to have source-type independent technique to achieve better compression as compared to what we get applying only arithmetic coding.

4. BRIEF INTRODUCTION TO BPT-I

BPT-I is very similar to QBT-I [4] except few differences:

BPT-I is applied on byte-pairs (16 bit integers) instead of quad-bytes (32 bit integers). Frequency of byte-pairs is computed using array data structure instead of binary search tree.

BPT-I first computes frequency of all byte-pairs and then arranges byte-pairs in decreasing order of their occurrence. Then it forms groups considering first 256 byte-pairs in first group, next 256 bytes in second group and so on. Number of groups may be specified by a user. With nGrp number of groups, most frequent (256 x nGrp) byte-pairs are encoded and remaining byte-pairs remains untransformed.

Encoded codeword consists of two tokens: group number and the location of byte-pair within a group. Group number is encoded using variable length prefix codeword and location is denoted using 8-bit index. Redundancy is introduced with 8-bit index location. Larger the number of groups; more is the redundancy in the transformed data. This should lead to better compression using arithmetic coding later.

During reverse transformation, decoder requires to know whether it is reading transformed byte-pair or not. For this, encoder uses prefix code for group codeword as follows:

- Zero(0): denote untransformed byte-pair
- As many 1s as number of groups: denotes last group
- Otherwise, number of 1s denote the group number

Here, prefix codes used for group codeword are 0, 10, 110, 1110, 11110,....,all 1s. For example, for nGrp=1: codeword 0 means no transformation and 1 means transformed byte-pair from 1st (also last) group; for nGrp=3: 0 implies untransformed byte-pair, 10 implies transformed byte-pair from 1st group (in first 256), 110 implies transformed byte-pair from 2nd group and 111 implies transformed byte-pair from group 3.

Maximum length of group codeword is same as number of groups nGrp.

Thus, a byte-pair is transformed using two components <variable-length prefix codeword for group number, 8-bit index codeword>.

8-bit index codeword denotes the position of byte-pair within a group. It introduces redundancy in the transformed data set. To exploit redundancy at the time of arithmetic coding, we have kept group codeword and index codeword in separate files.

Use of variable length code leads to more reduction the size of transformed file. Most frequent codes reside in the initial groups and are assigned shorter prefix code.

Shortest prefix code 0 is used for untransformed integers assuming the worst case of smaller nGrp. Smaller nGrp indicates that only a few byte-pairs are to be transformed. Smaller dictionary sizes helps to speed up the search process during decoding.

5. ALGORITHM

Algorithm uses two output files: **transformed data file** and **code file**.

The **transformed data file** contains the index codewords (for transformed integers only).

Prefix codes denoting group codeword are copied in the **code file**.

The number of bytes in a source file may not be in multiple of size 2, so initial nExtrabytes (= filesize modulo 2) bytes are not processed and output as they are. Transformation is applied to remaining bytes.

The structure of **code file** is as follows:

- Byte 1: nExtrabytes (2 bits) and nGrp (6 bits, maximum 64 groups)
- Byte 2 to nExtrabytes+1: unprocessed initial extra bytes from source file
- Next 2 bytes: Dictionary size d = number of most frequent integers to be stored
- Next 2*d bytes: byte-pairs in descending order of frequency
- Remaining bytes: prefix codes of transformed and untransformed integers

5.1 QBT-I Encoder

1. Setup:
   a. Find source file size, Accept nGrp
   b. nExtrabytes = filesize module 2
   c. Combine nExtrabytes (2 bits) and nGrp (6 bits) in a byte and write in the code file
   d. Read nExtrabytes bytes from source file and write to code file
2. Pass I (Dictionary building)
   a. Scan source file and compute frequency of all byte-pairs
   b. Sort byte-pairs in descending order of the frequency
   c. Output dictionary information in code file
      - Dictionary size = minimum (256 x nGrp, number of integers with frequency > 0)
      - Write dictionary size (using 2-bytes) and those many most frequent byte-pairs in the code file. Keep the dictionary stored in memory for later use in pass II. (One may use data structure like array or binary search tree (BST). BST is more efficient while searching.)

3. Pass II (Transformation: Rescanning the source from the beginning after extra bytes)
   a. Let prefix array contain binary numbers 10, 110, 1110,… for nGrp groups
   b. Read 16-bit integer (byte-pair)
   c. Search in dictionary
   d. If found at location k in dictionary
      - Output index = (k modulo 256) in transformed data file
      - Determine group prefix code:
        - Grp = k/256
        - If Grp is the last group, i.e. value of Grp is same as nGrp-1, then write last prefix (i.e. nGrp times bit 1) to prefix code file
        - If Grp is not the last group, write bits of prefix[Grp] to prefix code file
   e. If not found in dictionary, output integer data in the transformed data file as it is and write prefix bit 0 in prefix code file

Repeat steps from b onwards till all integers are scanned.

5.2 QBT-I Decoder

1. Setup
   a. Read nExtrabytes and nGrp from code file
   b. Read initial nExtrabytes bytes from code file and write in output file
2. Dictionary building
   a. Read Dictionary size and corresponding number of 16-bit integers from code file.
   b. Store these most frequent integers in dictionary (in memory) in the order of their arrival. For dictionary, one may use data structures like array or Binary Search Tree.
3. Inverse Transformation:
   a. Fetch prefix code from code file (bits are extracted till either 0 is found or nGrp bits are extracted)
   b. If prefix code is 0 (i.e. untransformed data), read 2-byte integer from transformed data file and write in the output file.
   c. If prefix is not 0, it means transformed data file contains 8-bit index for actual data.
      - Determine the group where the actual data belongs:
        - If prefix is nGrp times 1s (i.e. lastPrefix), Grp = nGrp-1 (i.e. last group)
        - Otherwise, search for prefix in prefix array. If it is found at location k, then Grp = k. (To avoid searching array, count number of leading 1s and then subtract 1 to determine Grp)
      - Determine location of the data in dictionary:
        - Read 1 byte index from transformed data file
        - Location of data in dictionary = Grp*256 + index
      - Write byte-pair from location in dictionary to output file.

Repeat step 3 till end of code file.

6. EXPERIMENTAL RESULTS AND ANALYSIS

Programs for BPT-I and arithmetic coding are written in C language and compiled using Visual C++ 2008 compiler.

Programs are executed on a personal computer with Intel(R) Core(TM)2 Duo T6600 2.20 GHz processor and 4GB RAM.

BPT-I is experimented with number of groups varying from 1 to 8. Experimental results are recorded using average of five runs on each test files. Most of the test files are selected from Calgary corpus, Canterbury corpus, ACT web site. Test files are selected to include all different file types and various file sizes as shown in Table 1.

Table 1. Test Files Used in Experiments

No	File name	Corpus, Description	File Size (Bytes)
1	act2may2.xls	ACT: excel file	13,48,036
2	calbook2.txt	Calgary: text file, "troff" format	6,10,856
3	cal-obj2	Calgary: object file, Mac executable	2,46,814
4	cal-pic	Calgary: CCITT fax file, bitmap image	5,13,216
5	cycle.doc	Own: word doc with images, text, drawing	14,83,264
6	every.wav	ACT: sound file	69,94,092
7	family1.jpg	Own: photograph	1,98,372
8	frymire.tif	ACT: graphics file	37,06,306
9	kennedy.xls	Canterbury: excel	10,29,744
10	lena3.tif	ACT: graphics file	7,86,568
11	linux.pdf	Own: pdf file, large	80,91,180
12	linuxfil.ppt	Own: power-point file with text, drawing	2,46,272
13	monarch.tif	ACT: graphics file	11,79,784
14	pine.bin	ACT: executable	15,66,200
15	profile.pdf	Own: pdf file with text, photos	24,98,785
16	sadvchar.pps	Own: ppt show	17,97,632
17	shriji.jpg	Own: image file	44,93,896
18	world95.txt	ACT: text file	30,05,020
		Total Size (Bytes)	**39,796,037**

Table 2. Contribution of most frequent byte-pairs

% contribution of k most frequent byte-pairs				
No.	**Filename**	**total pairs**	**k=256**	**k=512**
1	act2may2.xls	674018	78.78	84.53
2	calbook2.txt	305428	82.36	92.89
3	cal-obj2	123407	71.51	81.13
4	cal-pic	256608	95.86	97.78
5	cycle.doc	741632	67.50	69.61
6	every.wav	3497046	02.52	04.73
7	family1.jpg	99186	04.13	06.63
8	frymire.tif	1853153	69.16	76.60
9	kennedy.xls	514872	83.58	95.11
10	lena3.tif	393284	04.50	08.06
11	linux.pdf	4045590	40.65	43.83
12	linuxfil.ppt	123136	58.93	64.71
13	monarch.tif	589892	15.60	22.90
14	pine.bin	783100	54.61	65.06
15	sadvchar.pps	898816	07.55	08.59
16	shriji.jpg	2246948	01.96	03.07
17	world95.txt	1502510	78.38	90.53

To compress with arithmetic coding, we have used AC-nShft multi-bit processing implementation [3]. It is faster than conventional implementation of arithmetic coding.

Table 2 shows that most of the data is covered by most frequent 256 or 512 byte-pairs in majority of test files except in files like jpg and wav. It means increasing number of groups may not have significant benefit in reduction of file size in most of the files.

Table 3. Transformed File Size after BPT-I

No	**Source Size**	**Transformed File Size (Bytes) After Applying BPT-I Data Transformation**			
	(Bytes)	**nGrp=1**	**nGrp=4**	**nGrp=6**	**nGrp=8**
1	1348036	901850	927867	924258	924365
2	610856	398020	397399	397475	398495
3	246814	174515	172202	171976	172913
4	513216	299830	326046	326816	327832
5	1483264	1075857	1119995	1118831	1119567
6	6994092	7343748	7205958	7163378	7156294
7	198372	207200	204846	204833	205675
8	3706306	2656797	2621920	2597842	2596275
9	1029744	664300	659818	661330	661585
10	786568	818536	796773	791352	791314
11	8091180	6952741	7011228	6995715	6994121
12	246272	189623	189714	189390	190279
13	1179784	1162006	1109190	1099571	1099192
14	1566200	1236940	1183351	1172965	1172821
15	2498785	2594535	2565203	2558612	2558356
16	1797632	1842640	1836728	1833892	1834218
17	4493896	4731298	4692106	4680844	4679660
18	3005020	2015663	1960299	1955828	1957200
	39796037	**35266099**	**34980643**	**34844908**	**34840162**

Table 3 shows transformed file size (bytes) after applying BPT-I with varying number of groups. It is observed that larger nGrp results in smaller transformed files. Due to larger prefix codes with large nGrp, it may start deteriorating later as seen for nGrp=8, but it also increases the redundancy due to index code.

Figure 1 represents overall compression rate graphically. Figure 2 shows compressed file size of individual test files when BPT-I is applied with nGrp=1. Here also improvement is seen even with only 256 byte-pairs transformed.

Table 4 shows the overall compression rate and BPS as a result of compression (i) using only arithmetic coding (AC) and (ii) using AC after applying data transformation with BPT-I at pre-processing stage. It also shows total compression time and data transformation time.

It is observed from Table 4 that around 4.30% more reduction is obtained in compressed file size when arithmetic coding is applied after byte-pair data transformation BPT-I using most frequent 1024 byte-pairs (nGrp=4). It can be observed that after nGrp=4, increasing number of groups does not show significant improvement in compression rate.

Table 4. Overall Compression Rate and BitsPerSymbol (BPS) using AC only and AC after BPT-I

	Overall Compression Rate (%)	**Overall BPS**	**Total Compression Time (seconds)**	**Total Transformation Time (seconds)**
AC	16.77	6.658	17.488	
nGrp=1	20.443	6.365	41.512	25.148
nGrp=2	20.633	6.349	51.323	34.895
nGrp=3	20.937	6.325	61.317	44.972
nGrp=4	21.085	6.313	70.625	54.385
nGrp=5	21.148	6.308	79.176	62.964
nGrp=6	21.168	6.307	88.635	72.443
nGrp=7	21.166	6.307	96.594	80.433
nGrp=8	21.144	6.308	106.614	90.357

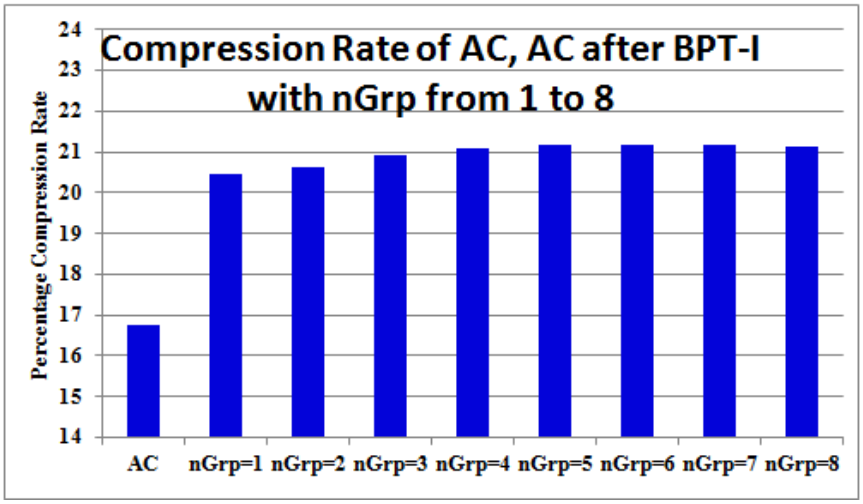

Figure 1. Overall Compression Rate (%) using only AC, using AC after BPT-I with varying nGrp

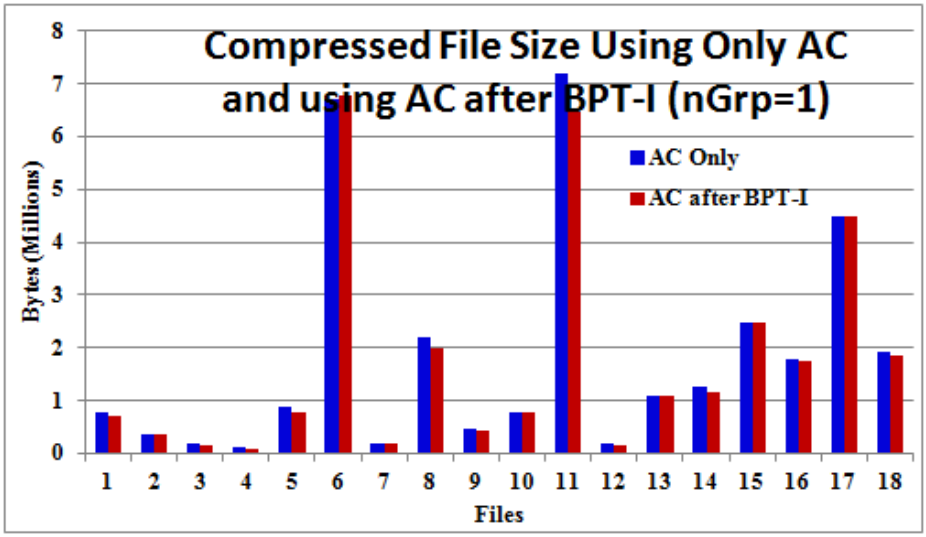

Figure 2. Compressed File Size using only AC, using AC after BPT-I with nGrp=1

7. CONCLUSION

With BPT-I data transformation applied before arithmetic coding, our purpose of achieving better data compression is achieved. Using BPT-I at a pre-processing stage of arithmetic coding, more than 4.3% overall data compression is achieved over compression using only arithmetic coding.

8. REFERENCES

[1] F. D. Awan, N. Zhang, N. Motgi, R. T. Iqbal, A. Mukherjee. "LIPT: A reversible lossless text transform to improve compression performance", Proceedings of the IEEE Data Compression Conference (DCC'2001), pp. 481, March 27–29, 2001

[2] T.C. Bell, A. Moffat, "A Note on the DMC Data Compression Scheme", Computer Journal, vol. 32(1), pp.16-20, 1989

[3] M. Burrows,D. J. Wheeler. "A block-sorting lossless data compression algorithm", Digital Systems Research Center, Research Report 124, Digital Equipment Corporation, Palo Alto, California, May 10, 1994

[4] G.V. Cormack, R.N. Horspool, "Data Compressing Using Dynamic Markov Modeling", Computer Journal, vol. 30(6), pp.541-550, 1987

[5] Jyotika Doshi and Savita Gandhi, "Computing Number of Bits to be processed using Shift and Log in Arithmetic Coding", International Journal of Computer Applications 62(15):14-20, January 2013, Published by Foundation of Computer Science, New York, USA. BibTeX

[6] Jyotika Doshi, Savita Gandhi, "Quad-Byte Transformation as a Pre-processing to Arithmetic Coding", International Journal of Engineering Research & Technology (IJERT), Vol.2 Issue 12, December 2013, e-ISSN: 2278-0181

[7] M. Dyer,D. Taubman, S. Nooshabadi, "Improved throughput arithmetic coder for JPEG2000", Proc. Int. Conf. Image Process., Singapore, pp. 2817–2820, Oct. 2004

[8] Philip Gage, "A New Algorithm For Data Compression", The C Users Journal, vol. 12(2)2, pp. 23–38, February 1994

[9] P. G. Howard, J. S. Vitter, "Arithmetic coding for data compression", Proc. IEEE. , vol.82: pp.857-865, 1994

[10] J. C. Kieffer, E. H. Yang, "Grammar-based codes: A new class of universal lossless source codes", IEEE Trans. Inform. Theory, vol. 46, pp. 737–754, 2000

[11] H. Kruse, A. Mukherjee. "Preprocessing Text to Improve Compression Ratios",Proc. Data Compression Conference, pp. 556, 1998

[12] G. Langdon, "An introduction to arithmetic coding", IBM Journal Research and Development, vol. 28, pp. 135-149, 1984

[13] Detlev Marpe, Heiko Schwarz, Thomas Wiegand, "Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard", IEEE Trans. On Circuits and Systems for Video Technology, vol. 13(7), pp. 620-636, July 2003

[14] Altan Mesut, Aydin Carus, "ISSDC: Digram Coding Based Lossless Dtaa Compression Algorithm", Computing and Informatics, Vol. 29, pp.741–754, 2010

[15] Moffat, "Implementing the PPM Data Compression Scheme", IEEE Transactions on Communications, vol.38, pp.1917-1921, 1990

[16] M. Nelson, "Data Compressin with the Burrows-Wheeler Transform", Dr. Dobb's Journal, pp. 46-50, Sept 1996 available at http://marknelson.us/1996/09/01/bwt/

[17] Radescu R., "Lossless Text Compression Using the LIPT Transform", Proceedings of the 7th International Conference Communications 2008 (COMM2008), ISBN 978-606-521-008-0., pp. 59-62, Bucharest, Romania, 5-7 June 2008

[18] Senthil S, Robert L, "Text Preprocessing using Enhanced Intelligent Dictionary Based Encoding (EIDBE)", Proceedings of Third International Conference on Electronics Computer Technology, pp.451-455, Apr 2011

[19] Senthil S, Robert L, "IIDBE: A Lossless Text Transform for Better Compression", International Journal of Wisdom Based Computing, vol. 1(2), August 2011

[20] Shajeemohan B.S, Govindan V.K, "Compression scheme for faster and secure data transmission over networks", IEEE Proceedings of the International conference on Mobile business, 2005

[21] Storer J. A., Szymanski T. G., "Data Compression via Textual Substitution", Journal of ACM Vol. 29(4), pp. 928-951, Oct 1982

[22] W. Sun, A. Mukherjee, N. Zhang, "A Dictionary-based Multi-Corpora Text compression System", Proceedings of the 2003 IEEE Data Compression Conference, March 2003

[23] S. Taubman and M. W. Marcellin, "JPEG2000: Image Compression Fundamentals", Standards and Practice. Norwell, MA: Kluwer Academic, 2002

[24] T. Wiegand, G. Sullivan, G. Bjontegaard, A. Luthra, "Overview of the H.264/AVC video coding standard", IEEE Trans. Circuits Syst.Video Technol., vol. 13(7), pp. 560–576, Jul 2003

[25] T. Welch, "A Technique for High-Performance Data Compression", IEEE Computer, vol. 17(6), pp. 8-19, June 1984

[26] M. J. Willems, Y. M. Shtarkov, T. J. Tjalkens, "The context-tree weighting method: Basic properties", IEEE Trans. Inform. Theory, vol.41, pp. 653–664, May 1995

[27] H. Witten, R. M. Neal, J. G. Cleary, "Arithmetic coding for data compression", Commun. ACM, vol. 30(6), pp. 520–540, 1987

[28] H. Witten, Alistair Moffat, Timothy C. Bell, "Managing Gigabytes-Compressing and Indexing Documents and Images", 2nd edition, Morgan Kaufmann Publishers, 1999