Quad-Byte Transformation as a Pre-processing to Arithmetic Coding

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

Savita Gandhi
Dept. of Computer Science; Gujarat University
Navrangpura Ahmedabad-380009, INDIA

Abstract

Transformation algorithms are an interesting class of data-compression techniques in which one can perform reversible transformations on datasets to increase their susceptibility to other compression techniques. In recent days, due to its optimal entropy, arithmetic coding is the most widely preferred entropy encoder in most of the compression methods. In this paper, we have proposed QBT-I (Quad-Byte Transformation using Indexes) method to be used as a pre-processing stage before applying arithmetic coding compression method. QBT-I is intended to introduce more redundancy in the data and make it more compressible using arithmetic coding. QBT-I transforms most frequent quad-bytes; i.e.4-byte integers. Dictionary of frequent quad-bytes is divided in a group of 256 quad-bytes. Each quad-byte in the dictionary is encoded using two tokens: group number and the location in a group. Group number is denoted using variable length codeword; whereas location within a group is denoted using 8-bit index. QBT-I can be applied on any source; not necessarily text or image or audio. QBT-I is expected to be faster due to 32-bit integer comparison which is faster than 4-byte pattern matching. QBT-I may also compress data along with transformation.

1. Introduction

Data transformation means transforming data from one format to another. When data transformation is applied as a pre-stage to conventional compression, the main purpose of a data transformation is to re-structure the data such that the transformed file is more compressible by a second-stage conventional compression algorithm. Thus, the intent is to use the paradigm to improve the overall compression ratio in comparison with what could have been achieved by using only the compression algorithm.

Arithmetic coding [8, 11, 30] is the most widely preferred efficient entropy coding technique providing optimal entropy. Most of the data compression algorithms like LZ algorithms [22, 28, 31, 32]; DMC (Dynamic Markov Compression) [2, 4]; PPM [15] and their variants such as PPMC, PPMD and PPMD+ and others, context-tree weighting method [29], Grammar—based codes [9] and many methods of image compression, audio and video compression transforms data first and then apply entropy coding in the last step. Earlier-generation image and video coding standards such as JPEG, H.263, and MPEG-2, MPEG-4 relied heavily on Huffman coding for the entropy coding steps in compression; but recent generation standards including JPEG2000 [5, 24] and H.264 [12, 26] utilize arithmetic coding.

With arithmetic coding, further improvement in compression is not possible due to entropy limitations. Better compression can be achieved if the data can be transformed to be more skewed.

Here, we have proposed Quad-Byte Transformation using Index (QBT-I) method with an intention to introduce more redundancy in the data and make it more compressible using arithmetic coding at the second stage as shown in figure 1. Both the transformation and compression algorithms are lossless.

QBT-I transforms most frequent quad-bytes (4-byte integer) using indexes from the dictionary of most frequent quad-bytes.

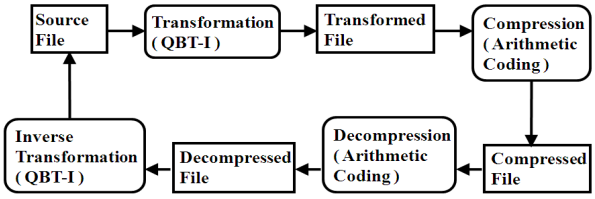

Figure 1Data Transformation before Compression

In general, due to two-stage process of transformation and then compression, compression process is somewhat slower in performance. However, this slowness in the run-time performance is acceptable since the transform will truly skew the data source to allow more effective compression. QBT-I can help to solve this speed problem as follows:

- Quad-byte transformation needs less number of transformations as compared to 1 to 3 byte transformations
- Integer comparison is faster as compared to 4-byte pattern matching

Main advantage of QBT-I is that it is not intended to specific type of data. It can be applied to any source.

2. Literature Review

Most of the research work in data transformation is intended to compress specific type of files like text, image, audio etc. Transformation techniques like DCT and wavelet are used for image files. After transformation, entropy encoding is applied in the final stage. Following research work to transform data is intended to compress text files.

- Burrows Wheeler Transform (BWT) described in [3, 16] performs block encoding.
- Star family transformation encodes words of different length. According to Dictionary-Based Multi-Corpora Text Compression system by Weifeng Sun, Amar Mukherjee, Nan Zhang [23], to gain a much better compression performance for the backend data compression algorithm, only letters [a..z, A..Z] are used to present the codeword. Star Transform [10], Length Index Preserving Transform (LIPT) [1, 17], and StarNT [23] are some of the transformation techniques that fall in this category.
- Transformation using index position of words in dictionary is performed by Intelligent Dictionary Based Encoding (IDBE) [21], Enhance Intelligent Dictionary Based Encoding (EIDBE) [19] and Improved Intelligent Dictionary Based Encoding (IIDBE) [20] methods.
- Two-byte transformation is applied in BPE (Byte Pair Encoding) [7], digram encoding and ISSDC (Iterative Semi-Static Digram Coding) [14].
- LZ family of algorithms fall in the category of techniques known as Sliding Window based techniques.

Even though above techniques are intended for text files, methods like BPE and digram encoding can be applied to any type of source. But, they will benefit more when applied to small-alphabet source like text files.

Many of the present day transformation techniques, along with transforming data, may introduce some compression also. Additionally they retain enough context and redundancy for compression algorithms to be beneficial.

2.1. BWT

BWT (Burrows-Wheeler Transform), named after its inventors Michael Burrows and David Wheeler, is introduced in 1994 in research report [3]. BWT is a block sorting algorithm used in data compression techniques such as bzip2 [18]. BWT generates output file with long sequence of same character repeating consecutively. Burrows and Wheeler recommend combining BWT with ad-hoc compression techniques Run Length Encoding (RLE) and Move-To-Front (MTF) encoding and then Huffman coding to provide one of the best compression ratios available on a wide range of data. Efficient variation of BWT uses arithmetic coding instead of Huffman coding in the last stage of entropy coding.

BWT is a lossless data compression algorithm that operates on blocks of data. For each block, it performs rotation-sorting-indexing. It is very time consuming and requires better data structures for efficient pattern matching. Bigger blocks will generate longer runs of repeats, leading to improved compression. At the same time, the sorting operations will slow down the speed considerably. BWT will usually have O(N logN) performance where N is block size. Burrows and Wheeler point out that a suffix tree sort can be done in linear time and space [16].

Much of research work has been done on the BWT and its different variations are proposed from time to time. Some of them include a variation in suffix tree constructions for faster transform by Weiner [27], McCreight [13], Ukkonen [25].

2.2. Star Transform

The Star Encoding, which is also called a changing skill, is introduced by Kruse and Mukherjee [10].

To generate codewords for representation of words, Star encoding uses a large static dictionary of commonly used words expected in the input files. The dictionary is partitioned into 22 disjoint sub-dictionaries based on the word length i ($1 \leq i \leq 22$), assuming the maximum length of an English word to be 22 letters. Words in sub-dictionary are arranged in the decreasing order of their frequency. Codeword for the first word at index 0 is encoded using * repeated i times. The next 52 words are assigned codeword that is a sequence of (i-1) characters * followed by a single alphabet letter from {a, b, …, z, A, B, ..., Z} respectively. For next 52 words at index 53 to 104, codeword is having first *, 2nd lowercase/uppercase alphabet letter, and then * repeated (i-2) times.

A source file thus transformed can be used by conventional data compression algorithms for better compression. In entropy coding compression methods like Huffman, the most frequent character '*' is compressed using only 1 bit codeword. Arithmetic coding also uses shortest possible number of fraction bits for most probable symbol. Large number of repeated *s will give better effect even in RLE. In the LZW algorithm, the long sequences of '*' and spaces between words allow efficient encoding of large portions of pre-processed text files.

2.3. Length Index Preserving Transform (LIPT)

Length Index Preserving Transform (LIPT) has been published by F. Awan and A. Mukherjee [1]. Star-Encoding does not work well with bzip2 because the long runs of '*' characters are removed in the first step of the bzip2 algorithm [6].

With LIPT, concept of sub-dictionaries and building dictionaries is same as that with Star encoding.

LIPT differs from Star encoding in generating codewords for transforming English words. In LIPT, the codeword is made up of three components <*,LengthChar,OffSet>. A word prefixed with '*' denotes that it is an encoded word. A word not found in the dictionary is left unaltered, and thus does not have a '*' as prefix. Second component 'LengthChar' denotes the length of the actual word. Length is represented using characters <a-z> corresponding to length <1-26>. Third component 'Offset' represents the index of the actual word in sub-dictionary. Index is also represented using letters of alphabet. Offset of first word is "a", 26th word is "z", 27th word is "A", 52nd word "Z", 53rd word "aa" and so on.

2.4. Star New Transformation (StarNT)

StarNT differs from earlier Star family transforms with respect to the meaning of the character '*'. In LIPT, first character '*' denotes the encoded codeword; whereas in StarNT, first character '*' denotes that the word is not encoded.

StarNT uses a semi-static single dictionary of words. Most frequently used 312 words are listed in the beginning of the dictionary in the decreasing order of their frequency of occurrence. The remaining words are sorted according to their lengths in decreasing order. This enables words with longer lengths to be encoded using shorter length codeword. Words with same length are sorted in the decreasing order of their frequency of occurrence.

Here the words are encoded with codeword of maximum three characters. The first 26 words in the dictionary are assigned "a", "b", …, "z" as their codewords. The next 26 words are assigned "A", "B", …, "Z". The 53rd word is assigned "aa", 54th "ab" and so on up to "ZZ". Thereafter the words are assigned codeword "aaa", to "ZZZ".

Use of maximum 3-character codeword reduces the size of the transformed intermediate file, thus the encoding/decoding time of the backend compression algorithm can be minimized. StarNT results in better compression ratio. StarNT is faster than LIPT both in transform encoding module and in transform decoding module.

2.5. Intelligent Dictionary Based Encoding (IDBE)

Shajeemohan and Govindan [21] proposed an encoding strategy called Intelligent Dictionary Based Encoding (IDBE) which offers better rate of compression. Words in the dictionary are encoded using two components <length of the codeword, codeword>. ASCII characters 33-250 are assigned as the codeword for the first 218 words in the dictionary. For the remaining words, permutation of two of the ASCII characters in the range of 33-250 is assigned as the codeword. For the left out words, if any, permutation of up to four of the ASCII characters is assigned. The length of the codeword is represented by the ASCII characters 251-254 with 251 for a code of length 1, 252 for length 2 and so on. Thus the words of maximum length 4 are encoded.

A better compression is achieved by using IDBE as the preprocessing stage for the BWT based compressor.

Senthil and Robert [19] brought a variation in IDBE and called it Enhanced Intelligent Dictionary

Based Encoding (EIDBE). In EIDBE, words in the input text are categorized as two letter words, three letter words and so on up to twenty two letter words. A dictionary is created with these words sorted by length in ascending order, followed by sorting on frequency of occurrence in descending order. First 199 words in each segment have single ASCII character (from 33 – 231) code. Code assigning for the rest of the tokens is same as in IDBE. The actual codeword consists of < word length, code>. Length is represented by the ASCII characters 232 – 253; 232 for two- letter words, 233 for three-letter words and so on up to 253for 22-letter words.

Senthil and Robert [20] also presented Improved Intelligent Dictionary Based Encoding (IIDBE). IIDBE uses dictionary same as EIDBE, codeword is also same <length, code> but code is determined as with starNT. Here authors suggested two operations for the first stage of pre-processing, first transforming the text into some intermediate form with IIDBE scheme and then applying BWT. The pre-processed text is then piped through a Move-To-Front encoder stage, followed by a Run Length Encode stage, and finally through an Entropy encoder, which is usually arithmetic coding.

Thus, IDBE, EIDBE and IIDBE can be considered as a pre-processing to bzip2.

2.6. Digram Coding

In digram coding, the dictionary consists of all letters of the source alphabet followed by as many pairs of letters, called digrams, as can be accommodated by the dictionary size.

A dictionary is built before starting encoding. In this semi-static dictionary, all of the individual characters are added to the first part of the dictionary and the most frequently used digrams are added to the second part of the dictionary. If the source contains n individual characters, and the dictionary size is d, then the number of digrams that can be added to the dictionary is $d - n$.

Digram coding uses fixed length code to encode symbols and digrams using the index position in the dictionary as codeword. Number of bits to be used for index codeword depends on dictionary size.

Altan Mesut and Aydin Carus [14] in their Iterative Semi-Static Digram Coding (ISSDC) use repeated digram coding. Like digram coding, all of the used characters and most frequently used two character digarms in the source are found and inserted into a dictionary in the first-pass, compression is performed in the second-pass. With ISSDC, this two-pass process is repeated several times. At each iteration, particular number of elements is inserted in the dictionary until the dictionary is full. Each two-pass iteration needs to scan the file two times. To reduce the need of repeated file i/o operations, authors of ISSDC have suggested to read the file once and store it in main memory for repeated use. For large files, this may not be feasible.

2.7. Byte-Pair Encoding

Byte Pair Encoding (BPE) presented by P Gage [7] is a simple universal text compression scheme based on the 2-byte pattern-substitution.

Byte pair encoding works by finding the most common pair of bytes in a file, and replacing all such pairs with a single unused byte. This substitution information is also stored with the compressed data. This process is repeated using the output of previous iteration as an input. BPE encoder stops when either no byte-pair occurs more than once or no unused characters are left.

BPE decodes most frequent byte-pairs using unused symbols in the source file, thus using 8-bits per 16-bit byte-pair. If there are no zero-frequency (i.e. unused) symbols, it will not be beneficial.

3. Research Scope

Transformation methods Star encoding, LIPT, StarNT use letters of English alphabet (a..z, A..Z) in the codeword. Methods IDBE, EIDBE, IIDBE can exploit the unused symbols like ASCII values 129 to 255 in text source files for most frequent or longer text patterns. All these methods are using dictionary of words or patterns and requires better data structures and pattern matching algorithms for efficiency.

BWT is very slow due to the need of rotations, sorting and mapping. It gives good compression only when later applied sequence of MTF, RTF and entropy encoding.

Byte-pair encoding, digram encoding and its variations can be applied to any type of source, but they will be beneficial only for small-alphabet source files like text.

Digram encoding and its variation ISSDC give better compression only when the source alphabet is small. If all 256 1-byte symbols are used in the source, the dictionary size needs to be longer than 256 words and each 1-byte character will be encoded using more than 8 bits. ISSDC has an additional drawback of the entire source to be in memory due to its two-pass multi-iterations. So, it can be applied to small size source files.

BPE decodes most frequent byte-pair using unused symbol in the source file, thus using only 8-bits per 16-bit byte-pair. It is also a repetitive process, repeating till there are no unused symbols or no repeated symbols. Thus BPE is beneficial

only when the source is having some unused symbols in it.

As mentioned before, arithmetic coding is the widely used entropy encoding method used with most of the compression methods. So, we saw a research scope here to have transformation method that can be applied to any type source data and introduce redundancy to skew the distribution for getting better compression using arithmetic coding later. There is also a scope to decrease the transformation time.

4. Proposed Transformation Method QBT-I

Proposed method QBT-I transforms 4-byte integers. It has following advantages over existing methods:

- It can be used with any type of source; not limited to text.
- Data transformation is expected to be faster due to following:
  - No pattern matching algorithms are needed during transformation or inverse transformation. Integer comparison is faster to execute as compared to matching a pattern of 4 bytes.
  - 4-byte transformation needs fewer transformations.

QBT-I transforms most frequent quad-bytes. It uses Index based transformation. QBT-I first prepares the dictionary of quad-bytes sorted in decreasing order of their occurrence. The dictionary is then logically divided into groups of 256 quad-bytes. Number of groups may vary and can be specified by a user. If number of groups is nGrp, then the dictionary size is to accommodate (256 x nGrp) quad-bytes.

Each quad-byte found in the dictionary is then encoded using two tokens; group number and the location of quad-byte within a group. Group number is denoted using variable length prefix codeword and location is denoted using 8-bit index. Quad-bytes in different groups may be at same location index. Thus, redundancy is introduced due to 8-bit index codeword denoting the location of quad-bytes. More the number of groups; more is the redundancy and better is the compression achieved using arithmetic coding later.

For decoder, it needs to know whether it has to reverse transform the quad-byte or not. Assuming the worst case of majority of integers not available in the dictionary, encoder uses group codeword 0 to notify that quad-byte integer is not transformed. As explained in Table 1, variable length prefix code starting with bit 1 denotes that an integer is found in the dictionary and is encoded using the index position within a group.

Thus, a quad-byte integer is transformed using two components <group code, index code>.

Group code starting with 0 implies no transformation, with as many 1s as the number of groups implies the quad-byte from the last group and otherwise it implies quad-byte in other groups.

For quad-bytes found in dictionary, 8-bit index codeword will introduce redundancy in the dataset. To exploit redundancy at the time of compressing data using arithmetic coding, it is advisable to keep group code and index code separate in a file or in files.

Use of variable length code helps to reduce the size of transformed file. Here most frequent codes are in the initial groups and are assigned shorter prefix code. Shortest prefix code 0 is used for untransformed integers assuming smaller dictionary size. Smaller dictionary sizes will speedup the search process.

Table 1 Prefix Code and Index Code for Quad-byte found in Dictionary

Group 1 Prefix code: If last group, $(1)_2$ If not last group, $(10)_2$	Integer Data	D0	D1	D2	...	D255
	Location	0	1	2	...	255
	Index Codeword	0	1	2	...	255
Group 2 Prefix code: If last group, $(11)_2$ If not last group, $(110)_2$	Integer Data	D256	D257	D258	...	D511
	Location	256	257	258	...	511
	Index Codeword	0	1	2	...	255
Group 3 Prefix code: If last group, $(111)_2$ If not last group, $(1110)_2$	Integer Data	D512	D513	D514	...	D767
	Location	512	513	514	...	767
	Index Codeword	0	1	2	...	255
…						

5. Conclusion

In this paper, our hypothesis is that the proposed method QBT-I of quad-byte data transformation will give better compression when used at pre-processing stage with arithmetic coding and will take relatively less time due to the need of fewer transformations and use of integer comparison instead of pattern matching.

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

[6] R. Franceschini, H. Kruse, N. Zhang, R. Iqbal, A. Mukherjee, "Lossless, Reversible Transformations that Improve Text Compression Ratio," Project paper, University of Central Florida, USA, 2000

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

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

[9] 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

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

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

[12] 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

[13] E. M. McCreight. "A space-economical suffix tree construction algorithm". Journal of the ACM, 23(2), PP.262–272, 1976

[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] Rexline S.J, Robert L. "Dictionary Based Preprocessing Methods in Text Compression - A Survey", International Journal of Wisdom Based Computing, vol. 1(2), August 2011

[19] 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

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

[21] 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

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

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

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

[25] Ukkonen. "On-line construction of suffix tress", Algorithmica, vol. 14(3), pp. 249–260, 1995

[26] 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

[27] P. Weiner, "Linear pattern matching algorithms", In Proceedings of the 14th IEEE Annual Symposium on Switching and Automata Theory, The University of Iowa, pp.1–11, 1973.

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

[29] 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

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

[31] J. Ziv, A. Lempel, "Compression of individual sequences via variable rate coding", IEEE Transactions on Information Theory, IT-24(5), pp.530-536, 1978

[32] J. Ziv, A. Lempel. "A Universal Algorithm for Sequential Data Compression", IEEE Trans. Information Theory, IT-23, pp.337-343, 1977

	BWT	**Star encoding**	**LIPT**	**StarNT**	**IDBE**	**EIDBE**	**IIDBE**	**Digram encoding**	**ISSDC**	**BPE**
Source Type	Any	Text	Text	Text	Text	Text	Text	Any	Any	Any
Dictionary	---	static, 22 sub-dict	static, 22 sub-dict	semi-static, single	semi-static	semi-static	semi-static	semi-static	semi-static	---
Size of token to be encoded	block	word upto 22 letters	word upto 22 letters	Word	Word	word	word	digram	digram	2 bytes
Matching	string	string	string	String	String	string	string	string or integer	string or integer	string or integer
comparison time per token	high	O(Sub-Dict-size)	O(Sub-Dict-size)	O(Dict- size)	O(Dict-size)	O(Dict-size)	O(Dict-size)	O(Dict-size)	O(Dict-size)	single 2-byte comparison
Code length	For Block	variable length: word-size	variable length: <*, word length, index>	variable length: index with max. 3-letters	variable length: <1-byte codeword length, codeword >	variable length: <1-byte word length, codeword >	variable length: <1-byte codeword length, codeword >	fixed, depends on dictionary size	fixed, depends on dictionary size	1 byte
Redundancy using	Index	*	index, length	index, length	index, length	index, length	index, length	index	index	substitution
Compresssion methods that can be applied later	MTF, RLE and then Huffman or arithmetic coding	RLE, LZW, Huffman, Arithmetic coding	Huffman or Arithmetic Coding		Pre-processing to BWT, Later MTF and RLE and entropy encoding			Huffman or Arithmetic coding		

Drawback	needs better data structures for sorting, comparing	only for text source	benefits only with small-alphabet source	repetitive, benefits only with small size source file and small alphabet source	repetitive, benefits only when source have some unused symbols, i.e. for small alphabet source