Symbol ranking text compressors: review and implementation

Peter Fenwick,

Department of Computer Science, The University of Auckland,

Private Bag 92019, Auckland, New Zealand.

peter-f@cs.auckland.ac.nz

Postal address	Department of Computer Science, The University of Auckland, Private Bag 92019, Auckland, New Zealand.
email	peter-f@cs.auckland.ac.nz
Telephone	+ 64 9 373 7599
Fax	+ 64 9 373 7453

Symbol ranking text compressors: review and implementation

Peter Fenwick,

Department of Computer Science, The University of Auckland,

Private Bag 92019, Auckland, New Zealand.

peter-f@cs.auckland.ac.nz

Abstract

Methods used by Shannon in 1951 when investigating the information content of English text are shown to lead to a class of "symbol ranking" text compressors. Included in this class are some recent compressors which combine high performance with completely novel techniques. These are described under the aegis of the symbol ranking classification. Some new compressors, deliberately designed as symbol ranking, are then described. One of these has operated at over 1 Mbyte per second in software and should be suitable for hardware implementation at tens of megabytes per second, or higher.

KEY WORDS lossless compression, Shannon, statistical coding, Burrows-Wheeler Transform, LZP compressor, ACB compressor

Introduction

Until quite recently, the field of lossless data compression was shared by Ziv-Lempel compressors (as production compressors and especially where speed was important) and PPM[1] (more particularly to demonstrate the possibilities of text compression). Apart from a few "outliers" such as DMC and Move-To-Front, the field appeared to be relatively stable and well understood and was documented by Bell et al[1].

Then, from early 1994, several novel text compressors appeared, some with very high performance. They included the "block sorting" compressor of Burrows and Wheeler, Bloom's LZP compressors, and Buynovsky's ACB compressor. The block sorting compressor is clearly related to techniques first described by Shannon in 1951, where he estimated the information content of English text from the statistics of attempts to guess the next letter of text. Shannon's approach can be regarded as the basis of a family of "symbol ranking" compressors which includes some of the new,

[1] A glossary of compression acronyms is included as an Appendix to this paper

unconventional, compressors and some new compressors which explicitly used symbol ranking.

This paper brings together the various members of the symbol ranking family. The next section gives a general review of symbol ranking and introduces compressors which are seen to use the technique. This is followed by more detailed descriptions of three of the most recent, and unconventional, compressors, none of which has had much recognition in the formal, refereed, literature. Finally is a detailed description of a symbol-ranking compressor which gives fast software compression and is suitable for hardware implementation.

History of Symbol Ranking

In 1951 Shannon published his seminal paper on the entropy of English text[2]. While his result of 0.6–1.3 bits per letter is well known and a challenge to all inventors of text compressors, his methods of measuring that entropy are much less recognised.

Shannon used human test subjects who, knowing the preceding text or context, tried to predict the next letter. He used two methods —

1. The subject had one guess and was then told either "correct" or "the answer is …", or
2. The subject had to keep on guessing until the correct answer was obtained.

Some compressors will be seen to use the "Shannon Type-1" coding, while others use "Shannon Type–2" coding, based on the two methods above.

Regarding the sequence of successes and failures as an information source allowed Shannon to predict the entropy of the original text. He also stated that the predictions were a recoding of the original text and that an "identical twin" of the subject who responded in the same way to the known text could recover the original input. It is this last concept which is developed here.

The essence of symbol ranking is that we maintain for each context a list of symbols ranked in the order of their likelihood of appearing in that context. Each input symbol is recoded into its rank in the list and that rank, *N*, is emitted as the code for the symbol. In the decompressor, an identical predictor offers symbols and the *N'*th is accepted as the correct symbol. So far we have just a symbol recoding or transformation, but as the recoded alphabet has a highly skewed frequency

distribution, implying good compressibility, the code can be then processed by a following statistical compressor. The final compressor may use arithmetic coding (as in most of the better compressors), adaptive Huffman or similar discrete codes (Burrows and Wheeler's original compressor and Sewell's latest BZIP) or a combination of the two (Howard and Vitter). In general, arithmetic coding gives better but slower compression, while discrete codes are faster with somewhat poorer compression.

All symbol ranking compressors exploit the skewness of the recoded input text. For example Table 1 is reproduced from Shannon's original paper and shows the number of attempts to obtain the correct symbol. Shannon worked with a mono-case alphabet and test subjects who had extensive experience with English text, to obtain rather better predictions than are observed in equivalent programmed implementations.

Guesses, or symbol ranking	1	2	3	4	5	> 5
Probability	79%	8%	3%	2%	2%	5%

Table 1. Shannon's original prediction statistics

Symbol ranking compression proceeds in two phases, the production of the ranked list of symbols and the coding of the indices into these lists. Compression may be improved by having a better predictor, or by having a better final coder. In the discussion to follow it is often difficult, if not impossible, to separate the two aspects; published results must be accepted as given. In two cases (block sorting or BWT, and sliding window) the internal details were available and the intermediate ranks and rank frequencies could be compared.

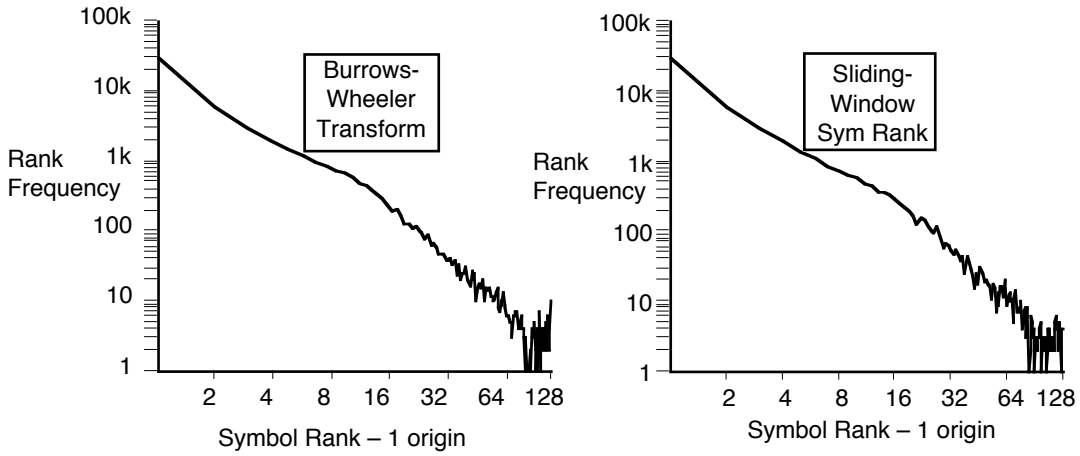

Figure 1. Frequencies of different code ranks for Burrows-Wheeler and Sliding-Window compression.

Symbol ranks from actual compressors for the Calgary Corpus[2] file PAPER1 are given in Figure 1. They emphasis the skewness of the symbol frequencies and confirm the underlying similarity of the two quite different methods.

Several compressors can be seen as implementing the methods of symbol ranking, but without reference to Shannon's original work. An early example is the MTF compressor of Bentley et al[3], which uses words rather than individual characters as its basic compression symbols. Lelewer and Hirschberg[4] describe the use of self-organising lists to store the contexts in a compressor derived from the PPM compressor described by Cleary and Witten[5]. Compressors by Howard and Vitter and by Yokoo are described later.

The significance of symbol ranking and the relation to Shannon's original work became apparent during study of the Burrows-Wheeler algorithm. When Bloom[6] announced his LZP compressors, they too were seen to be an implementation of a "type 1" Shannon encoding. As the concepts of symbol ranking became clearer, Bloom's methods were extended to give the sliding window symbol ranking compressor described later.

Symbol ranking also appears in the "CACM" implementation of arithmetic coding[7]. There the symbols are maintained in a frequency-ordered list with mapping tables to transform symbols to ranks and vice versa. The emitted code is really related to the symbol rank, rather than the symbol itself. The ordering of symbols by frequency is intended to minimise the search length; the fact that it implements an order-0 symbol ranking compressor is largely incidental to the main purpose.

Symbol ranking text compressors are clearly related to predictive coding, delta-coding or ADPCM of analogue data transmission. There, the signal values at preceding sample points are used to predict the signal at the next sample point, for example by polynomial curve fitting, and the error or difference between the prediction and actual value is transmitted. While at first sight there is no obvious way by which a text symbol or byte can be "slightly in error" or "a lot in error", a context-dependent ranked list of symbols provides such a mechanism. A symbol with a small but non-zero ranking is "nearly correct" and has a "small error", while one with a large ranking is clearly unexpected and can be transmitted as a "large error".

Symbol ranking has obvious parallels to PPM compression. Indeed, Cleary et al[8]

[2] The files of the Calgary compression corpus are available by anonymous FTP from ftp.cpsc.ucalgary.ca: /pub/projects/text.compression.corpus/ textcompression.corpus.tar.Z

have shown that PPM data structures lead directly to BWT compression if symbols are ranked according to probability within each context order (with exclusions), and the contexts then traversed from longest to shortest. PPM compressors must maintain two parallel data streams, one for the data proper and one for the context orders. Many of the improvements to PPM come from improving the tracking of the encoding and decoding models so that little context information needs to be transmitted. The hope in developing symbol ranking compressors was that symbol ranking, needing no information at all on the actual context order, might turn out a viable alternative to PPM.

Block Sorting, or Burrows Wheeler Transform

The Burrows and Wheeler "Block Sorting" compressor was first described in a 1994 report by Burrows and Wheeler[9]; later literature tends to describe the algorithm as the "Burrows-Wheeler Transform" (BWT). Improved versions were described by Wheeler[10] and Fenwick[11, 12] and more recently commercial-strength implementations have been released by Seward[13]. Burrows and Wheeler state that the original algorithm combines the speed of Ziv-Lempel methods and the compression of PPM. These claims are supported by experience. Arnavut and Magliveras[14] have shown that the Burrows-Wheeler Transform is but one of a group of permutations and that it may be possible to select a permutation group with better compression and accompany the compressed data with an ordinal group index or a coding of the permutation itself.

The Burrows-Wheeler Transform is unusual in that it compresses a permutation of the original text. The permuted text is processed by a Move-To-Front (MTF) transformation and then by an encoder designed for skew symbol distributions. Complete descriptions of the algorithm are found in references 9, 11 and 12.

In symbol ranking terms, the permutation collects together similar contexts and, therefore, similarly ranked sequences of symbols for adjacent contexts. The algorithm maintains just a single ranked table which it manages as the MTF list, emitting the symbol index as the code before moving the symbol to the front of the list. The MTF list changes gradually throughout the file as it adapts to the changing contexts and is generally a good approximation to the true ranking.

The results with the best arithmetic coding model are shown in Table 2. Discrete codes related to Huffman codes were used by Burrows and Wheeler in their original

paper and by Seward in the final version of BZIP.

Bib	*Book1*	*Book2*	*Geo*	*News*	*Obj1*	*Obj2*	*Paper1*	*Paper2*	*Pic*	*ProgC*	*ProgL*	*ProgP*	*Trans*	*AVG*
1.95	2.39	2.04	4.50	2.50	3.87	2.46	2.46	2.41	0.77	2.49	1.72	1.70	1.50	**2.34**

Table 2. Block Sorting (Burrows-Wheeler Transform)

Buynovsky's ACB compressor

Another recent compressor is the "ACB" algorithm of Buynovsky[15]. It gives excellent compression, the most recent versions of ACB being among the best known text compressors. While its original form is related to LZ–77 compression, with clever ways of improving the phrase coding, an alternative coding is related to symbol ranking.

The input text is processed sequentially, with the compressor building a sorted dictionary of all encountered *contexts*. Each symbol processed defines a new context and dictionary entry. Each context is followed in the input text (and therefore in the dictionary) by its corresponding *content* string or phrase. The symbols preceding the current symbol form the *current context* and the symbol and those following the *current content*.

When processing a symbol the compressor first finds the longest context matching the current context. This index of the context in the dictionary may be called the *anchor* of the context. (The actual position of the anchor in the input text is irrelevant.) Having found the anchor, the compressor searches neighbouring (similar) *contexts* for the phrase from the *contents* with the longest match to the current content. If this best phrase has length λ and occurs a distance δ contexts from the anchor (δ may be negative), the phrase is represented by the couple $\{\delta, \lambda\}$.

The final coding of $\{\delta, \lambda\}$ is rather more subtle and largely determines the final quality of the compression. As an example of one such optimisation the chosen phrase will often match some earlier phrase (closer to the anchor) to some smaller length μ and the encoding is not $\{\delta, \lambda\}$ but $\{\delta, \lambda–\mu\}$. The most recent portion of the context may be used as a conditioning class for the final coding models. This is the basis of Buynovsky's encoding.

Another coding follows by holding λ and manipulating δ. The match length λ defines a phrase length; as we move away from the anchor to the best match some of the phrases of length λ will match phrases which have been seen before, closer to the

anchor. So we encode not δ but ε, the number of *unique* phrases of length λ seen before the matching phrase. This coding uses *phrase exclusion*, as compared with the more usual *symbol exclusion*.

In some ways the operation is best envisaged using the sorted contexts as in the Burrows-Wheeler algorithm; in other ways it is related to LZ–77[16] (δ corresponds to a displacement and the use of contexts reduces the range of δ and thereby improves the coding), or it resembles LZ–78[17], regarding the neighbouring contexts as a dictionary and emitting indices into that dictionary.

Here we prefer to interpret the method as a derivative of a symbol ranking compressor. The anchor and its content provide the best estimate of the phrase to be emitted; the distance of the correct phrase from the anchor is a measure of its ranking with respect to the anchor context. We then have not a *symbol-ranking* compressor, but a *phrase-ranking* compressor. The principle follows immediately from Shannon's technique if we allow the predictions to be phrases rather than single symbols.

It is difficult to say much more about the ACB compressor, because many of the details which are crucial to the compression have not been released. The final compression has been published and the values are reproduced here for the version ACB_1.29b, Dec 1996.

Bib	*Book1*	*Book2*	*Geo*	*News*	*Obj1*	*Obj2*	*Paper1*	*Paper2*	*Pic*	*ProgC*	*ProgL*	*ProgP*	*Trans*	*AVG*
1.93	2.32	1.94	4.56	2.32	3.50	2.20	2.34	2.34	0.74	2.33	1.50	1.50	1.29	**2.20**

Table 3. Buynovsky's ACB compressor results

Bloom's LZP compressors

Much of the coding cost of LZ–77 compressors lies in the displacement to the matching phrase, which is essentially a random number and difficult to encode efficiently. Bloom's LZP compressors[6] arose from a desire to eliminate the displacement coding of LZ–77 compressors. They retain the sliding window of LZ–77 but, instead of finding the best *forward* match to the incoming symbols as for LZ–77, find the best *backward* match to the current context. The symbol immediately following that best matching context is found to be a good predictor of the next input symbol (60% success rate on a typical text file). The compressor then emits a flag to show either "correct prediction" or "incorrect prediction", with the correct symbol following if necessary. This is a direct implementation of a "Shannon type 1"

compressor.

Bloom emits the length of the matching phrase following the context, which may be regarded as a run-encoding of a succession of correct predictions. He presents several methods of finding the best matching context and several ways of representing the wrongly-predicted symbol. Combinations of these yield a variety of compressors, from ones which give moderate compression at very high speed to one which is among the best known compressors, but much slower. The compression is reported for the Calgary Corpus and speed for an Amiga 3000 (25 MHz 68030)

LZP1 The context table is accessed by hashing an order–3 context (4096 contexts), with each context referred back to its position in a 16 Kbyte sliding window. The output is encoded into byte-aligned combinations of literals match/mismatch flags, and codes for runs of literal matches. (4.15 bit/byte, 250 kbyte/s)

LZP2 This is like LZP1, but with Huffman coding of the output. (3.40 bit/byte, 80 kbyte/s)

LZP3 An order-4 context is used, but with "context confirmation" to ensure that the hashed and current contexts match, and using lower context orders as necessary. The output coding uses a complex of Huffman coders, with context order as a conditioning class. (2.78 bit/byte, 30 kbyte/s)

LZP4 This uses order-5 contexts, with context confirmation but without dropping to lower orders. The final coder uses arithmetic coding with conditioning classes and literals handled by techniques similar to PPM. (2.28 bit/byte, 6 kbyte/s)

Bib	*Book1*	*Book2*	*Geo*	*News*	*Obj1*	*Obj2*	*Paper1*	*Paper2*	*Pic*	*ProgC*	*ProgL*	*ProgP*	*Trans*	*AVG*
4.15	5.72	4.56	6.80	4.78	4.86	3.77	4.53	4.89	1.40	4.15	2.96	2.86	2.64	**4.15**

Table 4. Bloom's LZP1 compressor (highest speed)

Bib	*Book1*	*Book2*	*Geo*	*News*	*Obj1*	*Obj2*	*Paper1*	*Paper2*	*Pic*	*ProgC*	*ProgL*	*ProgP*	*Trans*	*AVG*
1.92	2.35	2.01	4.74	2.35	3.74	2.39	2.38	2.39	0.81	2.39	1.59	1.59	1.34	**2.28**

Table 5. Bloom's LZP4 compressor (best compression)

Other Symbol Ranking Implementations

Howard and Vitter[18] also follow PPM in developing a compressor, but one which explicitly ranks symbols and emits the rank. They show that ranking avoids the need for escape codes to move between orders, and also describe an efficient "time-stamp" exclusion mechanism. Much of their paper is devoted to the final encoder, using

combinations of quasi-arithmetic coding and Rice codes[19]. Their final compressor has compression approaching that of Moffat's PPMC[20], but is considerably faster with the results reproduced in Table 5.

Bib	*Book1*	*Book2*	*Geo*	*News*	*Obj1*	*Obj2*	*Paper1*	*Paper2*	*Pic*	*ProgC*	*ProgL*	*ProgP*	*Trans*	*AVG*
2.19	2.51	2.29		2.78			2.62	2.51		2.68	1.99	1.96	1.88	

Table 5. Results from Howard and Vitter

Yokoo[21] has recently described a compressor which may be regarded as an on-line version of the Burrows-Wheeler compressor, with similarities also to Buynovsky's ACB compressor. It uses explicit symbol ranking, and various discrete final coders; the results of Table 6 are based on Elias γ codes[22].

Yokoo uses a binary tree data structure to develop a sorted ordering of contexts and supplements the tree with a doubly linked list to maintain the lexical order of contexts. He measures the similarity of contexts by the lengths of their common suffixes.

Yokoo does not give results for the files PIC or TRANS, but from the other results it is possible to estimate the missing values and an average performance of about 2.8 bits per byte. The estimated values are shown in italics in Table 6.

Bib	*Book1*	*Book2*	*Geo*	*News*	*Obj1*	*Obj2*	*Paper1*	*Paper2*	*Pic*	*ProgC*	*ProgL*	*ProgP*	*Trans*	*AVG*
2.40	2.87	2.48	6.24	2.89	4.77	2.95	2.78	2.80	*0.90*	2.80	1.82	2.09	*1.70*	*2.8* (est)

Table 6. Yokoo's "Sorted Contexts" compressor (γ codes)

A Sliding-Window Symbol Ranking Compressor

This compressor[23] was apparently the first designed to use explicit symbol ranking, recognising the links to Shannon's original work. It was intended as a "proof of concept" compressor, extending Bloom's LZP compressors to handle "Type-2" Shannon encoding. It retains the sliding window to find previously matching contexts, but extends the search to find lower-ranked symbols.

A search as in the LZP compressors finds the best matching context and thence predicts the first symbol; if this symbol is rejected the search continues at the same order or context length for more distant occurrences of that context. The symbols following those contexts are then offered as further candidates, with exclusion ensuring that symbols are offered only once in a particular context. If the text in the window is exhausted before finding the correct symbol, the order is reduced by 1 and

the search resumed from the most recent text. An MTF table is eventually used as an order-0 predictor.

Within each context order the symbols are offered in order of recency, most recent first, which is equivalent to MTF coding within the context. The results are shown in Table 7.

Bib	*Book1*	*Book2*	*Geo*	*News*	*Obj1*	*Obj2*	*Paper1*	*Paper2*	*Pic*	*ProgC*	*ProgL*	*ProgP*	*Trans*	*AVG*
2.22	2.82	2.32	5.49	2.62	3.79	2.43	2.59	2.68	0.82	2.55	1.70	1.69	1.48	**2.51**

Table 7. Results with sliding window symbol ranking

Comments

When symbol ranking compressors were first investigated, following the recognition of relation of the Burrows-Wheeler algorithm to Shannon's work, it was hoped that their avoidance of explicit escape codes might allow improvements over the closely related PPM methods. It was also hoped that the use of explicit contexts might give improvements over the BWT compressor.

The anticipated improvements did not materialise. In retrospect, the BWT algorithm is seen to provide very efficient context analysis and the simple passing on of a symbol-ranking list from one context to the next is usually a very good predictor of symbol order. The ignorance of specific context information seems not too much of a hindrance. In comparison with PPM, the extra step *symbol* → *rank* → *code*, instead of just *symbol* → *code* does seem to degrade compression because some information is lost at each conversion. Again in comparison with PPM, symbol ranking must infer symbol probabilities, whereas PPM is able to maintain better models with explicit probabilities, with correspondingly better compression.

It appears that symbol ranking gives on average about 10% poorer compression than PPM compressors with comparable context analysis methods. The use of discrete output codes in some compressors (Huffman, Elias, etc) gives a further 5–10% compression penalty.

Constant-order Symbol Ranking Compression

Symbol ranking can be used for compressors which combine modest compression with very high speed. The one described here gives compression somewhat better

than the early "compress" or "compact" routines, but achieves very high speeds and should be suitable for hardware implementation at speeds of tens of megabytes per second. It may be compared with the LZRW[24] compressors, with Bloom's LZP1 and LZP2, and with GZIP in its fastest mode.

The heart of the compressor is the management of the lists of ranked symbols for each context. The mechanism is precisely that of a standard set-associative cache with LRU (least recently used) updating, but used in an unconventional manner. In a cache we are concerned only with hits and misses; the LRU is completely private to the cache and is irrelevant to its external operation. Here we are definitely concerned with the LRU functions, with the position of a hit within the LRU list used to encode a symbol. Note that Least Recently Used (LRU) is identical to Move To Front (MTF); LRU replaces the tail list element, whereas MTF expects to use the head element. Figure 2 shows the compressor, as it might be implemented in hardware.

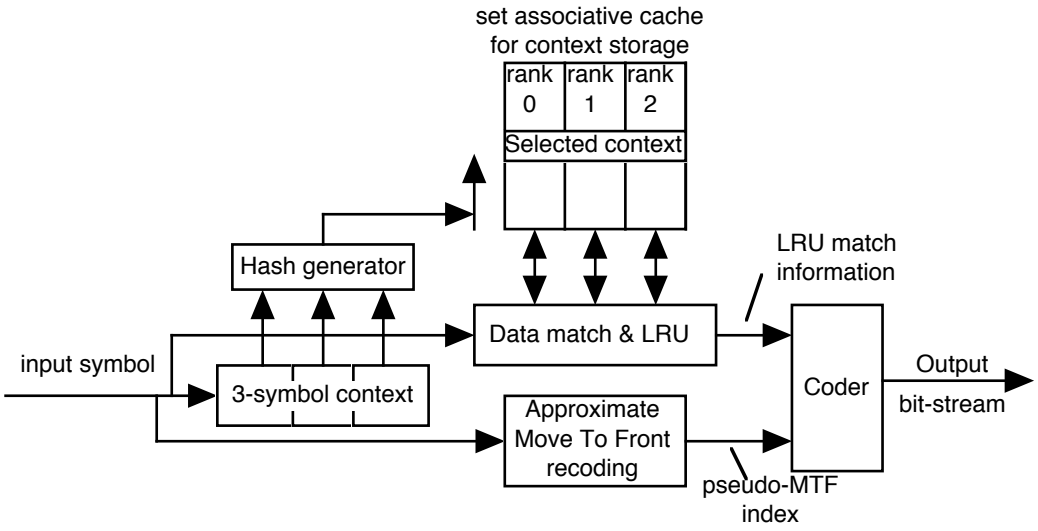

Figure 2. A hardware constant-order symbol ranking compressor

The three previous symbols are used as an order-3 context, with the 6 low bits of each symbol concatenated as an index to access one line of the cache[3] . The input symbol is matched against the LRU-ordered symbols, the match position is presented to the coder and the LRU status updated appropriately.

The final coding stage accepts the LRU match information and emits it with a defined variable-length coding, or the symbol itself if there is no LRU match. The coding is essentially unary, with a 0 for a correct prediction and a 1 for a failure and

[3] This approach gives about 10% better compression than a "good" hash function. It seems that it helps to retain some of input structure when forming the hash index.

step to the next rank. It is shown[12] that this simple coding is well suited to the actual symbol distribution. Two additional techniques give a slight improvement in compression

1. A Move-To-Front recoding of the symbol allows a shorter code for the more frequent symbols. While full MTF recoding is far too expensive in data movement, a similar effect can be achieved by exchanging the symbol with one halfway to the front of the MTF list[25]. A "short literal" of 5 bits (actually a 5 bit index into the MTF table) gives the best performance. Other literals are transmitted as the 8-bit index. Dithering the last 3 bits of the exchanged position gives an improvement of about 5%.
2. Runs of more than about 16 zeros are transmitted as an actual run length. The number of bits in the run length is sent as a "long literal" but with a value in the range of a short literal; no true long literal indices have these values.

The full coding is shown in Table 8, together with the code frequencies for compressing PAPER1 with 64K contexts. Compression for the whole Calgary Corpus is shown in Table 9. With 16K contexts the compression degrades by about 2%.

0		Rank-0 match	45%
10	*xxxxx*	short literal (index < 32)	28%
110		Rank–1 match	15%
1110		Rank–2 match	7%
1111	*xxxxxxxx*	Long literal; also long run of Rank–0	4%

Table 8. Output coding for constant-order compressor

Bib	*Book1*	*Book2*	*Geo*	*News*	*Obj1*	*Obj2*	*Paper1*	*Paper2*	*Pic*	*ProgC*	*ProgL*	*ProgP*	*Trans*	*AVG*
3.74	3.95	3.43	6.28	3.90	4.81	3.70	3.70	3.66	1.15	3.71	2.70	2.95	2.80	**3.60**

Table 9. Constant-order symbol ranking compressor –
64K contexts,maximum rank = 2

Increasing the maximum rank to 3 (4-way set association) improves the compression of text files by about 2%, but degrades the compression of binary files, leaving a similar overall result. Less compressible files improve as the maximum rank is reduced. Coding only ranks 0 and 1 with 64K contexts gives 6.10 bit/byte for GEO and 4.66 with OBJ1, improvements of 6% over coding up to rank = 3.

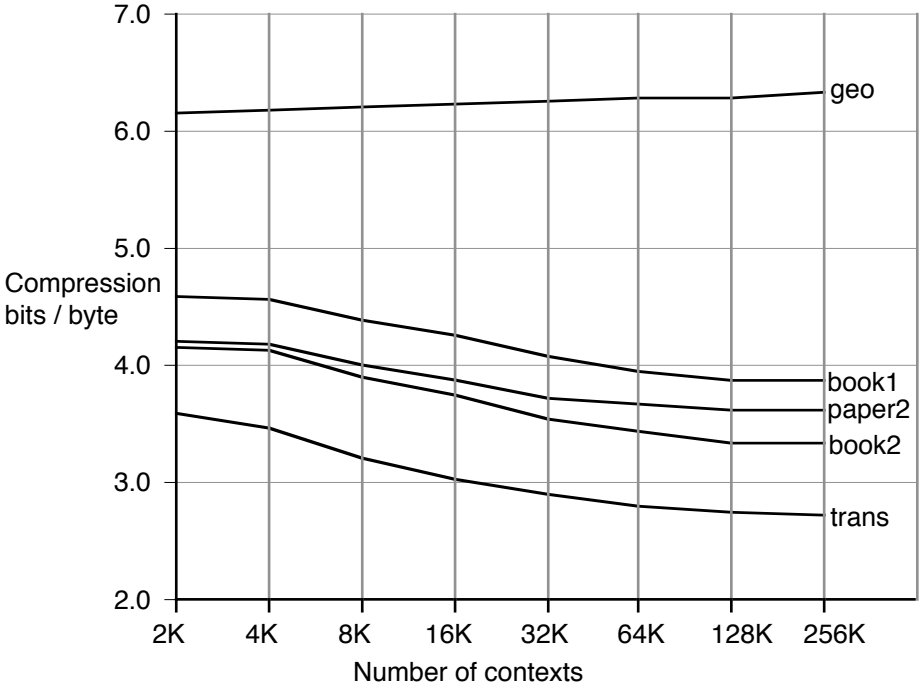

Figure 3. Constant-order compressor, compression v number of contexts

Results for representative files of the Calgary corpus are shown in Figure 3. Useful compression requires only a few thousand contexts for most files, but GEO and other binary files are quite anomalous, with the compression being better with fewer contexts.

The main feature of this compressor is its speed. On a 275 MHz DEC Alpha, a software implementation compresses at about 1 Mbyte/s. Using well-proven cache techniques and discrete components such as field-programmable logic arrays and fast static RAM, a hardware implementation with a speed of 30 Mbyte/s should be possible with little trouble. A completely integrated design should be able to run at several times that speed.

Conclusions

Symbol ranking is an interesting "new/old" compression method, which seems to have been rediscovered in several guises. For the current context it prepares a table of possible symbols, ranked according to their likelihood in that context; the rank is emitted as recoding of the symbol and, having a highly skewed distribution, is amenable to a compact representation. The compressor normally uses contexts of variable order, but with the interesting feature that neither those contexts nor their orders are ever released to the coder.

Of the compressors described, the block-sorting or Burrows-Wheeler Transform (BWT) is probably the best as a general file compressor. It achieves very high speed and compression which is very close to that of the best PPM compressors. A version of symbol ranking especially suited to hardware implementation gives moderate compression and should be capable of operating at very high speeds.

Acknowledgements

This work was started while the author was on Study Leave at the University of California–Santa Cruz, the University of Wisconsin–Madison and at the University of Western Australia and continued at the University of Auckland. It was supported by the University of Auckland Research Grant A18/XXXXX/62090/F3414032. The support of all of these institutions is gratefully acknowledged.

Thanks are due to many people for their advice on the algorithms described here.

- David Wheeler and Mike Burrows for their assistance with their algorithm
- Georgii Buynovsky and Leo Broukhis for describing the operation of the ACB compressor
- Charles Bloom for comments on all of the algorithms, but especially his own LZP compressors
- Julian Seward for his BZIP compressor, a 'clean room' implementation of the published block-sorting implementations

Appendix : Compression Acronyms

It seems that any work on data compression is necessarily accompanied by a plethora of acronyms. To aid understanding, the acronyms used in this paper are collected here. The algorithms discussed in this paper are marked by an asterisk. Some of the other algorithms have references in the text.

ACB * A text compressor developed by George Buynovsky[15].

BWT * Burrows–Wheeler Transform[9]. A compressor based on a permutation of the input, followed by a MTF operation and a statistical compressor.

BZIP * A BWT implementation by Julian Seward

DMC Dynamic Markov Coding. A compressor based on a Markov State representation of the input.

GZIP An implementation of LZ-77, released through the Free Software

Foundation

LRU Least Recently Used. A version of MTF, especially as used when finding a candidate for replacement (using the list tail).

LZ-77 A compression technique following from work of Ziv and Lempel in 1977[16]. The incoming text is matched against "phrases" in the preceding text and a phrase replaced by the distance to the reference and its length.

LZ-78 A compression technique described by Ziv and Lempel in 1978[17], using a dictionary of previously seen phrases. Phrases are replaced by indices into the dictionary.

LZP * A family of compressors developed by Charles Bloom[6], initially as a derivative of LZ-77.

LZRW A family of LZ-77 derived compressors which emphasise compression speed[24].

MTF Move To Front. A symbol recoding technique in which a symbol is coded by its position in a list and the symbol then moved to the front of the list. Frequent symbols tend to stay near the head of the list, with small code values.

PPM Prediction by Partial Matching. A compressor described by Cleary and Witten[5] which records contexts and the probabilities of symbols being seen in those contexts.

PPMC A version of PPM described by Moffat[20]; for some years it was the best known compressor

References

1. Bell, T.C., Cleary, J. G., and Witten, I. H., "*Text Compression*", Prentice Hall, New Jersey, 1990
2. Shannon, C.E. "Prediction and Entropy of Printed English", Bell System Technical Journal, Vol 30, pp 50–64.
3. Bentley, J.L., Sleator, D.D., Tarjan, R.E. and Wei,V.K. "A locally adaptive data compression algorithm", Communications of the ACM, Vol 29, No 4, pp 320–330.
4. Lelewer, D.A., Hirschberg, D.S., "Streamlining Context Models for Data Compression", Data Compression Conference, DCC-91, pp 313–322.
5. Cleary, J.G. and Witten, I.H. "Data compression using adaptive coding and partial string matching", IEEE Trans Communications, COM-32, vol 4, pp 396–402 Apr 1984.
6. Bloom, C., "LZP: a new data compression algorithm", Data Compression Conference, DCC'96
7. Witten, I., Neal, R., and Cleary, J., "Arithmetic coding for data compression", *Communications of the ACM*, Vol 30 (1987), pp 520-540.

8. Cleary, J.G., Teahan W.J. and Witten, I.H. “Unbounded length contexts for PPM”, *Data Compression Conference DCC-95* pp52–61, Snowbird Utah, March 1995.

9. Burrows, M. and Wheeler, D.J. “A Block-sorting Lossless Data Compression Algorithm”, SRC Research Report 124, Digital Systems Research Center, Palo Alto (1994). `gatekeeper.dec.com/pub/DEC/SRC/research-reports/SRC-124.ps.Z`

10. Wheeler, D.J. , private communication. (Oct ’95) This result was also posted to the comp.compression.research newsgroup. The files are available by anonymous FTP from `ftp.cl.cam.ac.uk/users/djw3]`

11. Fenwick, P.M. “Block sorting text compression”, Australasian Computer Science Conference, ACSC’96, Melbourne, Australia, (Feb). ftp.cs.auckland.ac.nz /out/peter-f/ACSC96.ps

12. P.M. Fenwick, “The Burrows–Wheeler Transform for Block Sorting Text Compression — Principles and Improvements”, *The Computer Journal,* Vol 39, No 9, pp 731–740, 1996.

13. Seward, J. “The BZIP compressor” posted to comp.compression.research newsgroup, (1996), later released by Free Software Foundation.

14. Arnavut, Z. and Magliveras, S.S., “Block Sorting and Compression”, Data Compression Conference, DCC’97, pp181-189

15. Buynovsky, G., “Associativnoe Kodirovanie”, (“Associative Coding”, in Russian), “Monitor”, Moscow, No 8, 1994, pp. 10–19.

16. Ziv,J. and Lempel, A., “A universal algorithm for sequential data compression”, *IEEE Trans. Information Theory*, IT-23, No 3, pp 337–343 May 1977

17. Ziv,J. and Lempel, A., “Compression of individual sequences via variable length coding”, *IEEE Trans. Information Theory*, IT-24, No 5, pp 530–536 Sep 1978

18. Howard, P.G., Vitter, J.S., “Design and Analysis of Fast Text Compression Based on Quasi-Arithmetic Coding”, Data Compression Conference, DCC-93, pp 98–107.

19. Rice, R.F. ,“Some Practical Universal Noiseless Coding Techniques”, Jet Propulsion Laboratory, JPL Publication 79-22, Pasadena California Mar 1979

20. Moffat, A. “Implementing the PPM data compression scheme”, IEEE Trans. Comm., Vol 38 No 11, pp 1917–1921, 1990

21. Yokoo, H., “An Adaptive Data Compression Method Based on Context Sorting”, Data Compression Conference, DCC-96, Snowbird, Utah, pp 160–169.

22. P. Elias, “Universal Codeword Sets and Representations of the Integers”, *IEEE Trans. Info. Theory*, Vol IT 21, No 2, pp 194–203, Mar 1975

23. P.M. Fenwick, “Symbol Ranking Text Compression with Shannon Recodings”, *J.UCS* Vol 3, No 2 pp 70–85 Feb 1997.

24. Williams, R.N., An extremely Fast Ziv-Lempel Data Compression Algorithm”, Data Compression Conference, DCC-91, Snowbird, Utah, p362 ff.

25. Fenwick, P.M. , “A New Technique for Self Organising List Searches”, *Computer Journal*, pp 450–454, Oct. 1991.