CHAPTER 8

Symbol-Ranking and ACB Compression

PETER FENWICK

8.1 INTRODUCTION

This chapter brings together two compression techniques which are outside the mainstream of accepted methods. Much of this material derives from the author's own work and draws heavily on previously published material.

1. Symbol-ranking compression is a method loosely allied to statistical techniques such as PPM, but is of much more ancient lineage, dating back to Shannon's original work on information.
2. Buynovski's ACB compressor is a relatively new compressor of extremely good performance, but one which does not fit easily into the more conventional taxonomy of compressors.

Both symbol-ranking and ACB compression are described in Salomon's recent book on data compression [1], the only reference known to the author apart from the original papers and reports.

8.2 SYMBOL-RANKING COMPRESSION

Symbol-ranking compression is directly descended from work by Shannon in 1951 in which he first established the information content of English text at 0.6–1.3 bits per letter [2]. He describes two methods. In both a person (the *encoder*) is asked to predict letters of a passage of English text. He postulates an "identical twin" *decoder*, who, being told of the responses, mirrors the actions of the encoder and recovers the original text. In retrospect the application to text compression is obvious, although probably limited by the processing power of the then-available computers, but if this work had been followed up the history of text compression might have been quite different.

Table 8.1 Shannon's Original Prediction Statistics

Guesses, or Symbol Ranking	Probability
1	79%
2	8%
3	3%
4	2%
5	2%

1. In Shannon's first method, the encoder predicts the letter and is then told "correct" or is told the correct answer.
2. In the second method, the encoder must continue predicting until the correct answer is obtained. The output is effectively the position of the symbol in the list, with the sequence of "NO" and the final "YES" responses a unary-coded representation of that rank or position.
3. A third method is a hybrid of the two given by Shannon. After some small number of failures (typically four to six) the response is the correct answer, rather than "NO." This is a useful compromise as predictions, both human and machine, become unreliable and difficult after more than half a dozen attempts.

This algorithm produces a transformation or recoding of the original text, with an output code for every input symbol and a preponderance of small codes. For his Method 2, Shannon gives the results reproduced in Table 8.1.

The distribution is very highly skewed, being dominated by only one value. This implies a low symbol entropy, which in turn implies excellent compressibility.

"All that is needed" to implement a compressor is some algorithm which can produce a symbol list ranked according to the expected probability of occurrence and a following statistical compressor.

8.2.1 Shannon Coder

Figure 8.1 outlines a compressor based on Shannon's second technique, in which the predictor is required to continue until the correct symbol is found.

The heart of the coder is a "predictor" which somehow estimates the next symbol and is then told whether to revise its estimate; the revision instructions (a sequence of "NO" and "YES" responses to the offered symbols) constitute the coder output. The decoder contains an identical predictor which, revising according to the transmitted instructions (and the same contextual and other knowledge), is able to track the coder predictor and eventually arrive at the correct symbol. The prediction is an ordered list of symbols, from most probable to least probable, and the emitted code is the rank of the symbol in this list. The coded output here is a sequence of 1's and 0's: a 0 for each unsuccessful prediction and a 1 for a correct prediction. Both encoder and decoder step to the next symbol in response to a "1" prediction.

In comparison with later compressors, the distinguishing mark of a symbol-ranking compressor is that it encodes *ranks* rather than *symbols* per se or symbol probabilities. In the example just given the encoded data is just a stream of undifferentiated bits. An improvement might be to use a statistical compressor which conditions the relative probabilities according to the rank (the number of immediately preceding 0's) or to condition these probabilities on the actual symbol context. But the work is always in terms of the rank of a symbol, rather than the symbol at that rank.

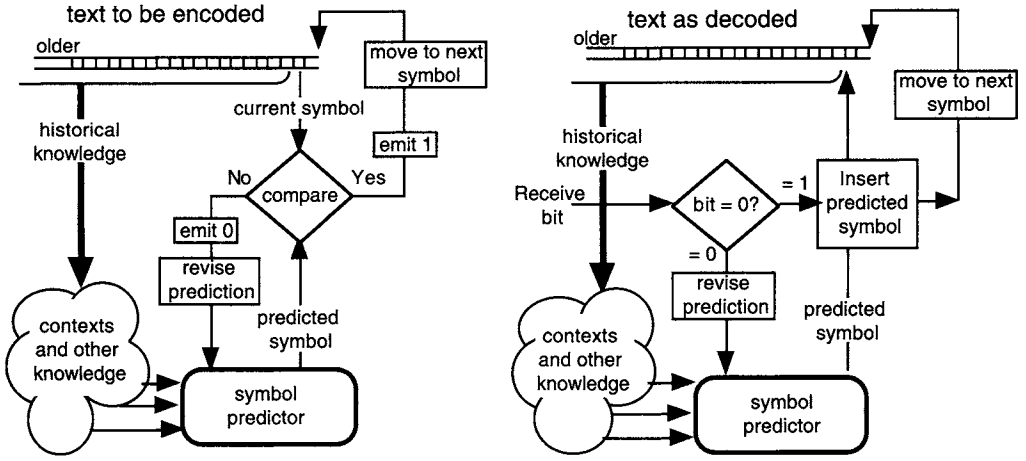

FIGURE 8.1
Shannon's symbol encoder and matching decoder.

8.2.2 History of Symbol-Ranking Compressors

Several compressors can be seen as implementing the methods of symbol-ranking, but with little reference to Shannon's original work.

1. The Move-To-Front compressor of Bentley *et al.* [3] uses words rather than individual characters as its basic compression symbols.
2. Lelewer and Hirschberg [4] use self-organizing lists to store the contexts in a compressor derived from PPM.
3. Howard and Vitter [5] 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. Their final compressor has compression approaching that of PPMC, but is considerably faster.
4. The Burrows–Wheeler Transform, described by Burrows and Wheeler [6], uses a context-dependent permutation of the input text to bring together similar contexts and therefore the relatively few symbols which appear in each of those contexts. A Move-To-Front transformation then ranks the symbols according to their recency of occurrence. The Move-To-Front list acts as a good estimate of symbol ranking.

The initial transformations of the Burrows–Wheeler compressor and the symbol recoding of the new method both produce highly skewed symbol distributions. Methods which were developed for the efficient coding of the Burrows–Wheeler compressors are also applicable to symbol-ranking. It will be seen that the frequency distributions of the recoded symbols are very similar for the two cases.

8.2.3 An Example of a Symbol-Ranking Compressor

Bloom [7] described a family of compressors based on the observation that the longest earlier context which matches the current context is an excellent predictor of the next symbol. His compressors follow Shannon's first method in that he flags a prediction as "correct" or "incorrect"

and follows an incorrect flag by the correct symbol. His method does not guarantee to deliver the most probable symbol, but is quite likely to deliver it or, failing that, will deliver one of the other more probable symbols. His various compressors use different methods of signaling success and of specifying the correct symbol, with different speeds and qualities of compression.

Fenwick [8] extended Bloom's method to produce a "proof of concept" symbol-ranking compressor, implementing the scheme of Fig. 8.1. Although it gives reasonable compression, it is far too slow to be regarded as a production compressor. The visible technique is exactly that of Shannon's second method. The symbol predictor offers symbols in the order of their estimated likelihood and the number of unsuccessful offers is encoded. The decompression algorithm is the obvious converse. An initial statistical decoder recovers the sequence of symbol ranks and the symbol prediction mechanism is then called, rejecting as many estimates as indicated by the rank.

The context-discovery and symbol prediction uses techniques derived from sliding-window LZ-77 parsing and with a fast string-matcher similar to that devised by Gutmann (and described by Fenwick [9]).

The algorithm proceeds in several different stages:

1. The preceding data is searched for the longest string matching the current context (the most recently decoded symbols). The symbol immediately following this string is offered as the most probable candidate. So far this is precisely Bloom's algorithm.
2. If the first offer is rejected, the search continues at the original order, looking for more matches which are *not* followed by already offered symbols. As the search proceeds, symbols which have been rejected are added to the *exclusion list* as candidates which are known to be unacceptable.
3. When all available contexts have been searched at an order, the order is reduced by 1 and the search repeated over the whole of the preceding text, from most recent to oldest. Matches followed by an excluded symbol are ignored and any offered symbol is added to the exclusion list.
4. When the order has dropped to zero, the remaining alphabet is searched, again with exclusions. This copy of the alphabet is kept in a Move-To-Front list, rearranged according to all converted symbols to give some preference to the more recent symbols.

Exclusion is handled by the method of Howard and Vitter. A table, indexed by symbol, holds the context or input index at which the symbol was last offered. When a symbol is first considered as a candidate, its current context is compared with that in the table; if the two match the symbol has been already considered within this context and must be excluded. The context of a non-excluded symbol is then saved before the symbol is offered. Advancing to the next position or context automatically invalidates all exclusion entries because none can match the new position.

8.2.3.1 Compression Results

Results for the compressor are shown in Table 8.2, tested on the Calgary compression corpus [10], using a Burrows–Wheeler compressor as a comparison. The symbol-ranking compressor had a "look-back" buffer of 64K symbols and a maximum order of 20 and used a structured coder as developed for the Burrows–Wheeler compressor.

8.2.3.2 The Prediction Process

Some output from an actual encoding of an early version of this chapter is shown in Fig. 8.2. There is a 2-symbol overlap between the two sequences. The actual text is written in boldface type, with the output value just below it (this is the number of wrong estimates for the symbol).

Table 8.2 Results in Compressing the Calgary Corpus, Values in Bits/Character

bib	book1	book2	geo	news	obj1	obj2	paper1	paper2	pic	progc	progl	progp	trans
Fenwick's Burrows–Wheeler (1996) average = 2.34 bpc													
1.946	2.391	2.039	4.500	2.499	3.865	2.457	2.458	2.413	0.767	2.495	1.715	1.702	1.495
Symbol-ranking, 64K, maxOrder = 20, 2.57 bpc													
2.274	3.028	2.485	5.507	2.841	3.793	2.547	2.591	2.686	0.84	2.551	1.702	1.688	1.503

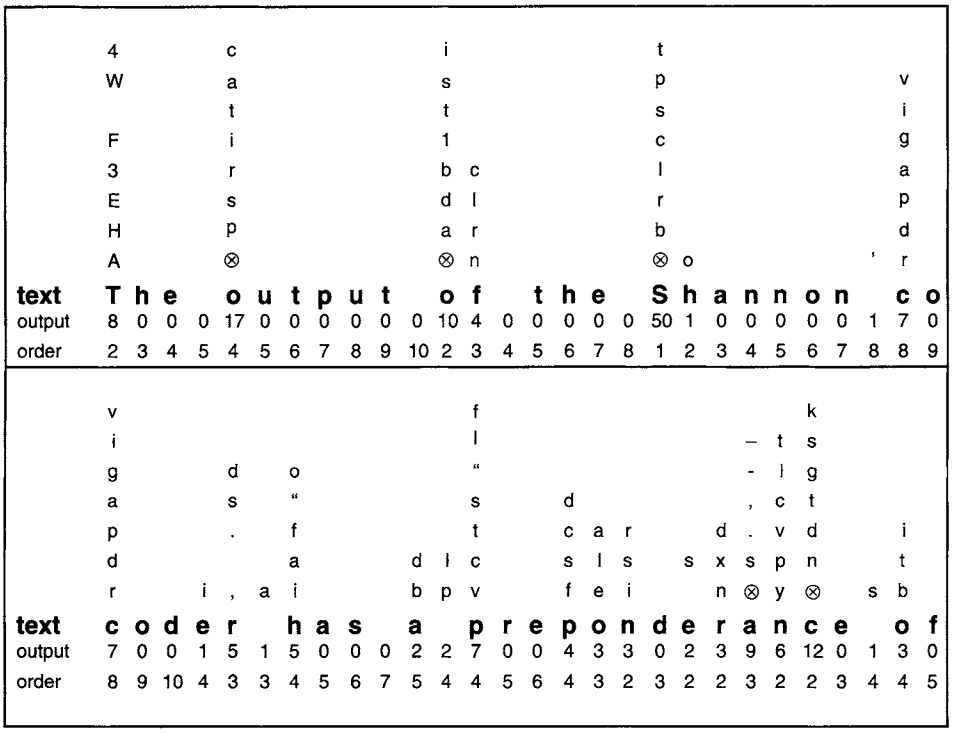

FIGURE 8.2
Illustration of symbol-ranking coder symbol prediction.

Above each symbol are the predictions for that symbol, with the first always at the top. The eighth and subsequent bad estimates are replaced by a single ⊗. Below the output code is the order at which that code is determined, often with an obvious relation to the preceding text.

What is not so easily conveyed is how the order changes during prediction. For example, in predicting the final "e" of "Shannon code," the unsuccessful "i" is predicted at order 11, but the next prediction (successful) is at order 4, with no external indication of the change in order.

While there may be some difficulty in establishing a symbol, the correct text often proceeds with no trouble for several symbols. A human predictor would get "of" with little difficulty and should also get "Shannon" almost immediately from the overall theme. Again, the latter part of "preponderance" should be predictable (there is no other reasonable word "prepon . . . "). The prediction is often almost eerily like that expected from a person, but one with a limited vocabulary and largely ignorant of idiom.

Table 8.3 Code Rank Frequencies for Block Sorting and Symbol-Ranking Compression

Rank	0	1	2	3	4	5	6	7	8	9
Burrows–Wheeler	58.3%	11.3%	5.5%	3.7%	2.8%	2.3%	1.8%	1.6%	1.4%	1.3%
Symbol-ranking	58.9%	11.6%	5.8%	3.8%	2.7%	2.0%	1.6%	1.4%	1.2%	1.1%

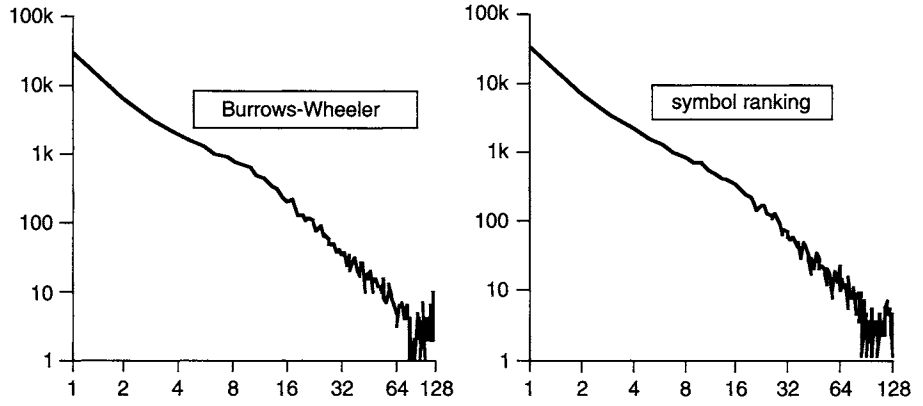

FIGURE 8.3
Frequencies of different code ranks for block sorting and symbol-ranking compression.

8.2.3.3 Burrows–Wheeler and Symbol-Ranking

Both the Burrows–Wheeler compressor and the symbol-ranking compressor produce a sequence of symbol ranks as input to the final compressor. Figure 8.3 shows the frequencies of the symbol ranks for the file PAPER1 using the two methods. At this scale the two are essentially identical, except for minor differences for ranks beyond about 16 where the symbol probabilities are quite low. (This figure uses 1-origin ranks to allow a logarithmic display.)

More detail can be seen in Table 8.3, which shows the relative frequencies of symbol ranks for the two compressors, for the relative frequencies greater than about 1%.

The symbol frequencies of Fig. 8.3 approximate a power law ($\text{freq}(n) = n^{-2}$). The exponent is close to 2 for most text files, is higher for more compressible files, and is lower for less compressible files.

8.2.4 A Fast Symbol-Ranking Compressor

Software compressors are often required to give excellent compression, with speed being a lesser consideration, especially if all is in software. An alternative requirement is to provide the best possible speed, even at the cost of good compression. This section describes such a compressor, designed for very fast hardware implementation, which was originally described by Fenwick [11]. It is essentially a cache memory and is probably limited in performance by the speed at which data can be supplied or extracted.

The heart of a symbol-ranking compressor is the management of the lists of ranked symbols for each context. The mechanism here is precisely that of a standard set-associative cache with LRU (Least Recently Used) updating, but used in an unconventional manner. A normal cache is concerned only with hits and misses, with the LRU mechanism controlling symbol replacement but otherwise completely private to the cache and irrelevant to its external operation. Here the LRU functions are crucial, with the position of a hit within the LRU list used to encode a symbol. Figure 8.4 shows the compressor, as it might be implemented in hardware.

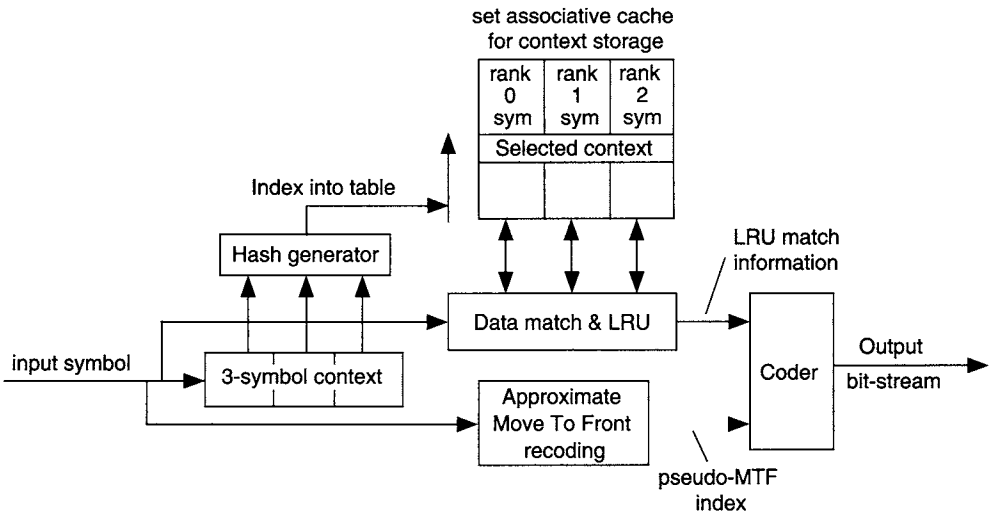

FIGURE 8.4
A hardware constant-order symbol-ranking compressor.

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. (Using a "good" hash function degrades the performance by about 10%, showing that it is best to preserve some of the raw context structure when forming the table index.) The input symbol is matched against the LRU-ordered symbols, the match position is presented to the coder, and the LRU status is updated appropriately, using any standard method such as a permutation vector or physical data shuffling. Earlier versions used 4-way set-association, the 4 bytes fitting conveniently into a 32-bit word, but the 3-way version shown gives very similar performance and is slightly simpler.

The final coding stage accepts the LRU match information and emits it with a defined variable-length coding or with 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 step to the next rank. An incoming symbol which matches the most recent member of the LRU set is emitted as a "rank–0" code. A match to the middle member is emitted as "rank–1" and a match to the oldest of the three members of the set as "rank–2."

Two additional techniques improve compression by about 15%, but make it less relevant for fast hardware implementation. Full details are given in [12] and especially [11], this last including complete code for the compressor.

- A simple and approximate Move-To-Front recoding of the symbol index allows a shorter coding for recent literals.
- Run-length encoding is used for long runs of correctly predicted symbols.

Table 8.4 shows the performance of the compressor, both in the form described and with the two optimizations removed. The compression speed is typically 1 Mbyte per second.

8.3 BUYNOVSKY'S ACB COMPRESSOR

Another recent compressor is the "ACB" algorithm of Buynovsky [13]. It gives excellent compression, the most recent versions of ACB being among the best known text compressors. Its technique is unusual and it does not fit easily into any standard compression taxonomy, although Fenwick [12] did attempt to classify it as a symbol-ranking compressor (or rather a phrase-ranking

Table 8.4 Constant-Order Symbol-Ranking Compressor—64K Contexts, 3 Ranks

bib	book1	book2	geo	news	obj1	obj2	paper1	paper2	pic	progc	progl	progp	trans
Cache symbol-ranking, 64 K contexts, order 3; average = 3.56 bpc													
3.70	3.94	3.41	6.28	3.86	4.76	3.69	3.63	3.61	1.15	3.60	2.65	2.79	2.76
Cache symbol-ranking, simplified hardware; average = 3.95 bpc													
4.04	4.74	4.00	6.15	4.43	5.02	3.80	4.25	4.32	1.34	4.08	3.03	3.13	3.00

compressor). It may be considered also a development of LZ-77 compressors, with the contents used to limit the coverage of the displacement, much as Bloom does in his LZP compressors [7].

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 in the dictionary) by its corresponding *content* string or phrase. The symbols preceding the current symbol form the *current context*. The current symbol itself and the following symbols form the *current content*. A more conventional compressor might find the best matching context and use that context to predict the next text. ACB recognizes that matching context is probably but one of a group of similar contexts and it is therefore surrounded by a cloud of more or less related contents which may be more appropriate.

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 neighboring (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 subtler and largely determines the final quality of the compression. For example, although there is clearly just one *context*, there may be several *contents*, of varying benefit. The benefit of using a longer content phrase must be balanced against the cost of encoding the phrase length and, more importantly, the phrase displacement. Thus sometimes it may be better to specify a short nearby phrase rather than a long remote one while at other times the remote phrase may be preferred.

The operation is illustrated using the dictionary shown in Fig. 8.5 (which is built from an earlier paragraph of this chapter using a case-insensitive sort). The text continues from each "context" to the corresponding "content." Both context and content are truncated here to about 20 characters. Also note that this example uses only a very short input text and that the selected content may be very poorly related to the current context.

- Text is "`is that`"; text is processed as far as "`...is th`" with "`at...`" to come.
- The best *context* match is "`s th`", an order-4 match at index 122 (anchor = 122).
- The best *content* match is "`at`" of length 2 at index 132 ($\delta = +10$, $\lambda = 2$).
- Emit the pair $(\delta, \lambda) = (10, 2)$.

And for another example:

- Text is "`another thing`"; processed to "`...her th`", with "`ing ...`" to come.
- The best context match is "`r th`", at 120 or 121, choose smaller index (anchor = 120) .
- The best content match is "`ing`" of length 4 at index 116 ($\delta = -4$, $\lambda = 4$).
- Emit the pair $(\delta, \lambda) = (-4, 4)$.

index	contexts	contents
115	eding data is search	ed for the longest s
116	longest string match	ing the current cont
117	decoded symbols). Th	e symbol immediately
118	t string matching th	e current context (t
119	deathly following th	is string is offered
120	candidate. So far th	is is precisely Bloo
121	a is searched for th	e longest string mat
122	ing is offered as th	e most probable cand
123	current context (th	e most-recently deco
124	ely Bloom's algorith	mThe preceding data
125	Bloom's algorithmTh	e preceding data is
126	The preceding data i	s searched for the l
127	lowing this string i	s offered as the mos
128	mbols). The symbol i	mmediately following
129	idate. So far this i	s precisely Bloom's
130	So far this is preci	sely Bloom's algorit
131	algorithmThe precedi	ng data is searched
132	). The symbol immedi	ately following this
133	most probable candi	date. So far this is
134	ongest string matchi	ng the current conte
135	iately following thi	s string is offered
136	andidate. So far thi	s is precisely Bloom
137	isely Bloom's algori	thmThe preceding dat
138	following this stri	ng is offered as the
139	for the longest stri	ng matching the curr
140	immediately followi	ng this string is of

FIGURE 8.5
Illustration of ACB compression.

Table 8.5 Results from Buynovski's ABC Compressor

bib	book1	book2	geo	news	obj1	obj2	paper1	paper2	pic	progc	progl	progp	trans
Buynovski's ACB compressor (ACB_2.00c) Average = 2.217 bpc													
1.957	2.348	1.955	4.565	2.344	3.505	2.223	2.352	2.354	0.759	2.342	1.516	1.510	1.304

The coding is subject to various conditioning classes and other optimizations whose details are not disclosed. The result is, however, excellent, as shown in Table 8.5; Buynovski's ACB compressor is one of the best developed to date.

As a final comment on the ACB compressor, note that it encodes phrases into *pairs* of numbers. As there is always some inefficiency in encoding a value into a compact representation, the need to encode two values may be a disadvantage of the ACB method.

ACKNOWLEDGMENTS

Much of the material of this chapter was originally published in the following two papers. The author acknowledges and thanks the publishers for permission to include this material.

1. Springer-Verlag and the editorial board of *Journal of Universal Coding*, for "Symbol Ranking Text Compression with Shannon Recodings" by P. M. Fenwick, 1997, *Journal of Universal Coding*, Vol. 3, No. 2, pp. 70–85.
2. John Wiley for "Symbol Ranking Text Compressors: Review and Implementation" by P. M. Fenwick, 1998, *Software–Practice and Experience*, Vol. 28, No. 5, pp. 547–559, April 1998.

8.4 REFERENCES

1. Salomon, D., 2000. "*Data Compression: The Complete Reference*, 2nd Ed., Springer-Verlag, New York.
2. Shannon, C. E., 1951. Prediction and entropy of printed English. *Bell Systems Technical Journal*, Vol. 30, pp. 50–64, January 1951.
3. Bentley, J. L., D. D. Sleator, R. E. Tarjan, and V. K. Wei, 1986. A locally adaptive data compression algorithm. *Communications of the ACM*, Vol. 29, No. 4, pp. 320–330, April 1986.
4. Lelewer, D. A., and D. S. Hirschberg, 1991. Streamlining Context Models for Data Compression, *Data Compression Conference*, pp. 313–322. IEEE Computer Society, Los Alamitos, California.
5. Howard, P. G., and J. S. Vitter, 1993. Design and Analysis of Fast Text Compression Based on Quasi-Arithmetic Coding, *Data Compression Conference*, pp. 98–107. IEEE Computer Society, Los Alamitos, California.
6. Burrows, M., and D. J. Wheeler, 1994. A Block-Sorting Lossless Data Compression Algorithm, SRC Research Report 124, Digital Systems Research Center, Palo Alto, CA, May 1994. Available at `gatekeeper.dec.com/pub/DEC/SRC/research-reports/SRC-124.ps.Z`.
7. Bloom, C., 1996. LZP: A New Data Compression Algorithm, *Data Compression Conference*. Vol. 3, No. 2, pp. 70–85, IEEE Computer Society, Los Alamitos, California.
8. Fenwick, P. M., 1997. Symbol ranking text compression with Shannon recodings, *Journal of Universal Computer Science*, Vol. 3, No. 2, pp. 70–85, February 1997.
9. Fenwick, P. M., 1995. Differential Ziv–Lempel text compression. *Journal of Universal Computer Science*, Vol. 1, No. 8, pp. 587–598, August 1995. Available at http://www.iicm.edu/JUCS.
10. Bell, T. C., J. G. Cleary, and I. H. Witten, 1990. *Text Compression*, Prentice Hall, Englewood Cliffs, NJ. The Calgary Corpus can be found at ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus.
11. Fenwick, P. M., 1996. A Fast, Constant-Order, Symbol Ranking Text Compressor, Technical Report 145, Department of Computer Science, The University of Auckland, Auckland, New Zealand, April 1996.
12. Fenwick, P. M., 1998. Symbol ranking text compressors: Review and implementation. *Software—Practice and Experience*, Vol. 28, No. 5, pp. 547–559, April 1998.
13. Buynovsky, G., 1994. Associativnoe Kodirovanie. ["Associative Coding," in Russian], *Monitor, Moscow*, No. 8, pp. 10–19.