PPM compression without escapes

P.M. Fenwick

Department of Computer Science, The University of Auckland, Auckland, New Zealand

Abstract

A significant cost in PPM data compression (and often the major cost) is the provision and efficient coding of escapes while building contexts. This paper presents some recent work on eliminating escapes in PPM compression, using bit-wise compression with binary contexts. It shows that PPM without escapes can achieve averages of 2.5 bits per character on the Calgary Corpus and 2.2 bpc on the Canterbury Corpus, both values comparing well with accepted good compressors.

1 Introduction

Computer algorithms may be characterised by the algorithm (simple or complex), the data structures (simple or complex), and again data structures (large or small). This work described in this paper grew out of considering what data compression might be achieved by combining a simple algorithm and a simple but possibly large data structure.

A long-time standard in text compression is the PPM family ("Prediction by Partial Matching"), initially proposed by Witten et al [1]. Significant improvements were given by Bloom [2], and later in the PAQ family of compressors [3]. A more recent standard is the Burrows-Wheeler algorithm [4, 5], with later survey articles[6, 7] including many citations, and describing improvements to the basic algorithm.

What is shared by all these "improvements" is the increasing complexity of the algorithms. Thus Bloom and the later PAQ compressors include ever-more complex models for statistical prediction and even machine-learning techniques such as neural nets. Improvements to the Burrows-Wheeler algorithms likewise depend on complex statistical modelling and context recovery techniques (the contexts having been lost in the basic transform).

2 PPM compression

To summarise its operation, a PPM compressor builds a table of "contexts", each accompanied by symbols which have been seen to follow the context. As a new symbol may occur at any time, some means must exist to allow augmentation of the context symbols. Usually this means that each and every context must include an *escape* symbol to signal that the coder/decoder must fetch the symbol from a lower order context, which is usually more populous. Many cases require a sequence of escapes to drop down even to order-1 or order-0.

The probability of the escape relative to the other symbols is crucial, but can only be estimated. If the escape is given a low probability it will be expensive to emit; a high probability penalises the emission of the existing, known, symbols.

Some investigations by the author[6], combining Burrows-Wheeler and PPM compression showed that a large part of the output stream was concerned with establishing contexts (in effect handling escapes) rather than actually coding symbols — about 50% for order 4 contexts and perhaps 90% for order 12. The author has therefore looked at the possibility of reducing the number of escapes, or even omitting them entirely.

This current paper presents one aspect of this investigation. In contrast to most other developments, it emphasises very simple modelling and algorithms, even at the cost of relatively high memory usage.

3 Compressors and Results

This section will present a sequence of 3 compressors to illustrate the development of the final technique. The results are summarised, for the Calgary and Canterbury Corpora, in Table 1, including three other compressors for comparison and also one other test file.

3.1 GZIP & BZIP2

Standard GZIP and BZIP2 with default parameters, given as widely-available examples of good compressors.

3.2 PPMref

This is a simple PPM compressor, written by the author as a testbed with no special tricks or optimisations. Escapes are handled much as any other symbols, with no probability prediction. It serves as a reference by which the other experimental compressors are judged.

3.3 BWC (BitWise Compressor)

The simplest possible arithmetic coding model is for a binary alphabet, with two counts. If the counts will each fit into a single byte, we need only two bytes per model and a data structure of millions of models is quite feasible. [1]

The BWC compressor uses an array of 1 million (2^{20}) binary models each with two 1-byte counters and occupying in all 2 Mbyte. This array is indexed by the previous 20 bits of the data stream, with no regard for byte boundaries, giving a binary PPM compressor of order 20.

The result is a surprisingly good compressor, generally within 10–15% of GZIP. There is little benefit in increasing the order; in fact geo is best with lower orders. Increasing the counts to 2 bytes gives only modest improvement (say 1%), while doubling the storage cost.

3.4 BWCV (BitWise Variable Order)

A PPM encoder issues an escape when it must process a symbol which is not present within the context for the current order. This requirement does not apply to a binary order, which *always* contains both valid symbols. But we can also escape if the context itself is absent or invalid. With a binary context this is easily achieved by initialising both counts to zero. An invalid (zero) context can then force an automatic escape to a lower order, *with no explicit escape code emission.*

This is done in the BWCV compressor. All models are initialised to {0,0}, except for the 0-order context, set to {1,1}. The coder tracks down from the highest order (still 20 and binary) until a valid (non-zero) context is found, from which the bit can be emitted. The coder then traces back through the bypassed orders, validating each by first setting its counts to {1,1} and then incrementing the appropriate count. The decoder of course mirrors these actions, following the coder, governed just by the validity of the contexts and with no explicit signalling. The result is a small improvement over the BWC compressor for most files.

The result for pic is especially good, but perhaps not surprisingly as this is essentially bit-stream data which is forced into a byte representation.

[1]A 1-byte model was tested, with constant total of 256, but gave rather poor results.

3.5 PPM BitWise

The two previous compressors (with binary order=20) are equivalent to byte-wise PPM of order only 2.5. While this can be extended by increasing the binary order, even as far as 28 with current RAM sizes, some other aspects become important –

1. The models array becomes increasingly sparse and inefficient in space utilisation.
2. The array becomes quite time-consuming to initialise, especially for sizes of 100 Mbyte (say binary orders of 26–27). The initialisation can easily overwhelm the compression proper.
3. Performance suffers if the models do not fit into a cache or, for *very* large models, into real memory..
4. There is anyway little gain with these relatively modest increases in coding order.

We now move to a combination of byte-wise PPM to establish the context and bitwise compression within each byte, with results shown in the PPMbit column. The byte contexts are determined as usual for PPM. Each context contains an array of 256 binary models, all except for the zero-order byte context initialised to 0. Each bit is processed as with the BWCV compressor, dropping down the *byte* order to find a valid context for that bit. It is quite usual to find one byte being handled by several different byte-order models.

In essence the array of models is indexed by initially 0, then the left-most bit, then the left-most two bits and so on until the right-most bit uses a model selected by the preceding 7 bits. This gives a small bit-wise PPM structure within each byte. The actual implementation uses a prefix 1-bit on the index to provide information on the bitwise order, to prevent bytes with leading zeros all mapping to the 0-model until the first 1 is processed. (With some leading zero bits, the leftmost bit is always processed by model indices of 1, 2, 4 etc, the sequence splitting as 1-bits are encountered.) All counts at the order-0 context are initialised to 1.

Contexts with only a single symbol are treated specially; the array of 256 code models (512 byte) is created only for multiple symbol contexts. As most contexts are at the highest order, and most of these encode only a single symbol, we reduce memory requirements by usually 40–60%, to those shown in Table 1. A byproduct is that single-context bytes can be copied rather than coded (with a signalling flag) and this improves compression by about 1%.

Another interesting aspect of this compressor is that the best results use orders 0, 1, 2, and 4, omitting order 3. Most of the "filtering" or elimination of symbols from the context occurs between orders 1 and 2 (at least for text files). It seems to be more effective to improve the count statistics at order 2, rather than emit bits from the "better" order 3 which, being used less frequently, accumulates poorer statistics.

This compressor is nearly as good as the reference compressor on most files, and compares well with the "standard" compressors.

Table 1 includes comparative results for the (small!) 100MByte file often used with the PAQ compressor family. In "their" terms, the output file from the last column is about 32.8Mbytes, compared with 17–20Mbyte for the best of the far more complex PAQ compressors. But the compression compares well with the standard GZIP and BZIP2 compressors. This section includes compression times for enwik8 on a 2.2GHz x86 processor.

Finally, Table 1 shows the memory requirements for each of the test files, using the final "PPM bit" compressor.

References

[1] Cleary, J.G. Witten, I.H. “Data compression using adaptive coding and partial string matching”, *IEEE Trans Communications*, COM-32, vol 4, pp 396–402, 1984.

[2] Charles Bloom, “Solving the Problems of Context Modeling”, *informally published report*, see `http://www.cbloom.com/papers/`

[3] Mahoney, Matt “Data Compression Programs”, `http://mattmahoney.net/dc/`

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

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

[6] Fenwick, P.M., “Burrows-Wheeler Compression”, *Lossless Compression Handbook*, Ed Khalid Sayood, Academic Press, 2003, pp 169–193.

[7] Fenwick, Peter, “Burrows-Wheeler Compression: Principles and reflections”, *Theor. Comp. Science*, Vol 387, No 3 (Nov 2007), pp 200–219.

Table 1: Calgary and Canterbury Corpus Results

	GZIP	BZIP	PPMref	BWC	BWCV	PPM bit	
			reference	order 20	var order	bpc	Mbyte
Calgary Corpus							
bib	2.521	1.975	1.994	2.820	2.769	2.078	3.9
book1	3.261	2.420	2.326	2.890	2.873	2.461	14.2
book2	2.707	2.062	2.068	2.849	2.841	2.150	12.6
geo	5.351	4.447	4.850	5.939	5.719	5.039	15.4
news	3.073	2.516	2.480	3.400	3.383	2.625	14.4
obj1	3.840	4.013	3.882	4.594	4.995	4.026	3.7
obj2	2.646	2.478	2.590	3.529	3.618	2.638	14.2
paper1	2.796	2.492	2.406	3.195	3.135	2.603	3.0
paper2	2.896	2.437	2.364	3.000	2.950	2.573	3.7
pic	0.880	0.776	0.867	0.819	0.808	0.877	6.4
progc	2.681	2.533	2.477	3.248	3.187	2.622	2.6
progl	1.817	1.740	1.844	2.489	2.448	1.883	2.4
progp	1.822	1.735	1.816	2.488	2.435	1.872	2.0
trans	1.621	1.528	1.649	2.532	2.520	1.633	2.9
Average	2.708	2.370	2.401	3.128	3.120	2.506	
Canterbury Corpus							
bible	2.354	1.671	1.703	2.321	2.325	1.696	12.5
csrc	2.253	2.180	2.153	2.874	2.808	2.195	0.8
ecoli	2.313	2.158	1.948	2.010	2.008	2.005	0.2
excl	1.600	1.009	1.293	1.608	1.604	0.953	8.1
fax	0.880	0.776	0.867	0.819	0.808	0.877	6.4
html	2.600	2.479	2.418	3.238	3.144	2.515	1.7
lisp	2.664	2.758	2.596	3.413	3.291	2.628	0.4
man	3.318	3.335	3.076	4.230	4.073	3.286	0.6
play	3.128	2.529	2.475	2.911	2.858	2.713	5.3
poem	3.241	2.416	2.328	2.789	2.772	2.463	9.0
sprc	2.703	2.705	2.742	3.545	3.472	2.766	3.5
tech	2.715	2.015	1.998	2.747	2.731	2.099	8.2
text	2.863	2.272	2.226	2.768	2.727	2.394	5.0
world192	2.344	1.584	1.711	2.803	2.798	1.743	28.4
Average	2.498	2.135	2.110	2.720	2.673	2.167	
enwik8	2.801	2.321	– – –	2.965	2.978	2.099	78.2
PC time (sec)		55	220	49	61	100	