Compression Techniques for Chinese Text

Phil Vines Justin Zobel
Department of Computer Science, RMIT, P.O. Box 2476V
Melbourne 3001, Australia

Abstract

With the growth of digital libraries and the internet, large volumes of text are available in electronic form. The majority of this text is English but other languages are increasingly well represented, including large-alphabet languages such as Chinese. It is thus attractive to compress text written in the large alphabet languages, but the general-purpose compression utilities are not particularly effective for this application. In this paper we survey proposals for compressing Chinese text, then examine in detail the application to Chinese text of the partial predictive matching compression technique (PPM). We propose several refinements to PPM to make it more effective for Chinese text, and, on our publicly-available test corpus of around 50 Mb of Chinese text documents, show that these refinements can significantly improve compression performance while using only a limited volume of memory.

Introduction

Compression of information yields two major benefits. The first is economic: compressed data occupies less disk space and can often be processed more quickly than when uncompressed, since more data can be transmitted or fetched from disk in a given time. The second is indirect: models that achieve good compression demonstrate an accurate understanding of the properties of the data, which is important for applications such as speech recognition, spell checking, and cryptography. Small improvements in performance as compression approaches the entropy have marginal economic impact but can have significant indirect benefit.

Many compression techniques have been proposed for English text [1], of which vast quantities are available as news articles, electronic journals, or on the Web. Currently much less Chinese text is available in electronic form, but it has been estimated that 70% of all Chinese text prepared for printed publication is developed electronically [2]. Compression exploits redundancy and pattern to achieve reductions in size, by coding common symbols in fewer bits than other symbols. How a symbol might be defined in Chinese is not clear, as there are thousands of distinct characters and no delimiters between words.

In this paper we examine techniques for compressing Chinese text, considering both special-purpose methods designed for Chinese and general-purpose compression tools. We show that none of the current techniques performs especially well, either requiring excessive memory or yielding only moderate compression. To establish a new benchmark for compression of Chinese we investigated in detail the most effective general-purpose compression technique, PPM, proposing and testing several refinements to improve its performance on large alphabet languages such as Chinese. In particular we show that the standard techniques for escape prediction and memory management do not work well for Chinese, and propose modifications that cater to the characteristics of Chinese text. We show that this modified PPM allows compression of Chinese to around 7 bits per character using less than 1 Mb of memory, compared to around 8.5 bits per character with previous techniques, and as little as 6.2 bits per character if more memory is available.

	Corpus			
	small	*stories*	*magazines*	*news*
Size (Mb)	0.37	5.98	8.27	34.96
Character occurrences	197,587	3,139,533	4,336,521	18,334,262
Distinct characters	3,458	12,486	5,575	11,210
Distinct pairs	42,176	355,116	335794	813,901
Distinct triples	51,094	703,953	797,540	2,663,202

Table 1: *Statistics of Chinese text collections.*

Characteristics of Chinese Text

The major difference between English and Chinese is that the latter has a much larger set of characters. Over 50,000 characters have been recorded [3], although most of these are archaic. A much smaller number of characters is used in the common coding standards, most of which are 16-bit. Another difference from English text is that Chinese is an agglutinating language. That is, there is no character used to mark word boundaries—readers segment the text into words based on their understanding of the language. Compression techniques take advantage of such text structure, but in Chinese structure is to a significant extent implicit in the semantics of the text, and will not always be evident to a compressor.

To compare compression techniques a test corpus is required, such as the Calgary corpus used to evaluate general-purpose compression algorithms. No such standard corpus is available for Chinese text. We have used four test files which contain a range of different types of Chinese text. The file *small* is a collection of eight issues of the magazine "Lian Yi Tong Xun" (Friendship News), a publication of the Ottawa Chinese students association. The file *magazines* is a further collection of Chinese magazines, in particular Hua Xia Wen Zhai (China News Digest), a weekly digest of world news and articles submitted by Chinese students. The file *stories* is a collection of both modern and classical stories and poems; as many of the poems are from the Song and Tang periods (618–1279 AD), the vocabulary contains a higher proportion of characters which are used less frequently in modern writing. The file *news* contains approximately one year of articles from the Internet news group `alt.chinese.text.big5`. For the purposes of the study all files were converted to the Guo Biao coding standard. Table 1 shows some basic statistics of the files.

Our interest is specifically in Chinese text, so we have stripped embedded ASCII from these files. A production compression scheme would of course have to compress the ASCII along with the Chinese.[1]

Approaches to Text Compression

Compression theory distinguishes two activities: the construction of a *model* that associates a probability with each symbol; and *coding* to produce a compressed representation of data with respect to the probability of each symbol [1].

Compression schemes can be characterised as *semi-static* or *adaptive*. A semi-static approach uses fixed symbols and codes to compress a given file, determined during an initial pass over the data to collect statistics. In adaptive schemes the probability assigned to each symbol changes as the text is encoded.

Once symbol probabilities have been supplied by a model it is necessary to use them to encode the symbols. The two best-known coding techniques are Huffman coding and arithmetic coding. In Huffman coding, a symbol of probability p is assigned a string of bits whose length is an integer approximation to $-\log_2 p$, so that, in effect, p is approximated to an integral power of 2^{-1}. For a predictive model—in which symbol probabilities can be close to unity—an exact coder

[1] The corpus will shortly be available via anonymous ftp from `ftp@cs.rmit.edu.au`.

such as an arithmetic coder can achieve a significant compression advantage. Arithmetic coding represents probabilities as precisely as the supporting hardware allows, encoding the message as a single floating point number that is transmitted incrementally [4]. The only space overhead is the need to include a terminating symbol. In practice, arithmetic coding is more space efficient than Huffman coding—indeed, arithmetic coding is provably near optimal—but is slower.

In the remainder of this section these concepts are considered in the context of previous proposals for compressing Chinese text. We do not have access to the software or data used in these previous experiments; as part of this survey we quote compression figures (as bits per 16-bit Chinese character, or *bpc*), but note that compression performance is highly corpus-dependent and that different data was used in each case, so the figures should be taken as broad indicators only.

Semi-Static Models

Perhaps the simplest example of compression based on a semi-static model is the Unix `pack` utility, which determines individual character frequencies and calculates Huffman codes to represent the data, using a zero-order model. The compression performance is poor, typically around 60%–70% of the size of the original text.

Ong and Huang [5] proposed a dictionary-based scheme in which Chinese characters are assigned either 8-, 12-, 16-, or 20-bit codes, based on frequency distributions observed in the data to be compressed. This is effectively a zero-order model in which the character probabilities, instead of being coded exactly, are rounded to the nearest integral power of 2^{-4}, leading to poor compression due to inexact representation of probabilities. Average compression results are 11.2 bpc. However, this implementation was slightly faster than adaptive LZW [6] compression.

Another semi-static approach to compression is to regard the text as a series of n-grams, that is, strings of some fixed length n. Then the text can be encoded using a semi-static model with one entry for each distinct n-gram. Lua has observed that for Chinese such approaches lead to huge dictionaries that far outweigh any gain in compression [7]. In theory the overhead of the dictionary should diminish as the data files get sufficiently large, but such behaviour was not evident for our files; for *news*, the 3-gram entropy was 6.6 bpc, but including the model the overall compression was 13.1 bpc, making the highly optimistic assumption that 5 bytes is sufficient to store a dictionary entry. (A more realistic estimate is 10 bytes, with 6 for the string and 4 for a counter, giving 19.6 bpc overall, not including any pointers used for housekeeping.) Lua explores various types of n-gram dictionary compression, and, using only 1-grams and semantically-determined 2-grams, compression of 9.2 bpc is reported for a small file; but the dictionary overhead is calculated at 12 bits per entry, a figure that seems improbably small. Experiments with longer n-grams were also described, in which compression of 8.2 bpc was obtained, however it was reported that the procedure was too computationally expensive.

Dictionary-Based Adaptive Models

The two major classes of adaptive coders are the Lempel-Ziv (LZ) adaptive dictionary coders [8] and predictive schemes. The LZ family of coders work by replacing strings of text by pointers to previous occurrences of the same string. Several variations have been developed on this basic theme, including LZW [6], which is the basis of the popular *gzip* compression software; LZW uses pointers into a phrasebook containing a fixed number of strings, allowing the simplification of transmitting pointers only, as initially the phrasebook contains all the one-character strings. Such compressors are generally faster than predictive coders such as PPM (described later) but do not yield the same size reduction.

Recent work by Gu [9] describes an adaptive scheme for Chinese text, in which Chinese characters are divided into 8 groups according to character frequencies obtained from a large collection. The groups contain 1, 2, 8, 32, 128, 512, 2048, and 4096 characters each; the characters in each group are coded using 3, 4, 7, 8, 10, 11, 14, and 16 bits respectively. During compression, any character that occurs more frequently than the least frequent in the the next group is "promoted" to that group, and the other character is "demoted". This highly specific strategy would not be

effective for general-purpose compression, or even for a different large-alphabet language. However, when the compressor is used for Chinese, starting with such a distribution might well do better than starting with a uniform probability distribution. Gu reports compression of 8.3 bpc averaged over several files totalling 600 Kb.

Predictive Models

Predictive models assign a probability to an occurrence of a symbol in a message with regard to the symbol's *context*, usually the sequence of symbols immediately preceding the current symbol [10]. Thus each context has a probability distribution, describing the symbols that can follow the context and their dependent probabilities. The number of preceding symbols used as context is the *order* of a model.

The space required to store a predictive model is a major constraint on the model order. For current hardware and ASCII text, orders of up to 5 or 6 are practical when compressing English text using a character-based model. For other symbol sets, such as the set of words occurring in a long English document, even first-order models are not easy to manage. Recent work has shown that unbounded order models are possible and can yield small improvements in compression [11], but require large volumes of memory.

Tseng and Huang [12], as reported in Chang and Chen [13], use a predictive coding method that codes the two bytes of each Chinese character separately. The first byte is coded on the basis of its frequency with respect to all the other first bytes in the file. The second byte of each code is coded on the basis of its frequency occurrence in the context of the first byte. Huffman codes are employed in both instances. Tseng and Huang describe a refinement based on bit stuffing, which is intended to save one bit per coded character, but it is not clear that this refinement has any effect in practice.

Gu [14] describes a predictive scheme based on Chinese characters. This scheme, which is similar to 16-bit PPM, is discussed later.

Chen and Chen [2] describe a predictive scheme where words rather than characters are the basic units of compression. The text is segmented using statistical information concerning word and individual character probabilities obtained for a large corpus of text. Punctuation characters are considered to be words. The authors consider zero-, first-, and second-order models. In the zero-order model, each word is coded as an independent probability, whereas, in the first-order model, the word is coded with respect to the probability of it occurring in the context of the immediately preceding word. When the word is not known in that context, the system uses an escape code to drop back to the zero-order model.

Their test corpus contains one million words and is drawn from a variety of sources. They report that word-based compression achieves compression of 7.4 bpc, while (Chinese) character-based compression achieves only 9.6 bpc. It is not clear what the memory requirements of this approach are; as we show, even character-based first-order predictive compression can require a great deal of memory.

Another predictive scheme for Chinese was suggested by Tsay et al. [15], using a form of Markov model. The authors compare using first- and second-order models trying bytes, nibbles, and sub-nibbles (two bits) as symbol tokens. They conclude that, for their data, a first-order model using byte symbols gives the best compression, averaging 7.7 bpc. Their results were worse for the second-order model, possibly as a consequence of the small size of their two sample files, of 20 Kb and 30 Kb. We have observed in our experiments that considerably longer samples are required to develop good higher-order models. Moreover, such small samples do not provide reliable estimates of compression performance.

Performance of Existing Compression Techniques

To establish a standard for performance of compression techniques on Chinese text we investigated the entropy of the files in our test corpus. We first considered the zero-order entropy of the data

	Corpus			
	small	*stories*	*magazines*	*news*
Model entropies				
Character	9.02	9.55	9.08	9.44
Character-pair	6.84	9.00	7.51	8.09
Segmented word	7.34	8.34	7.51	8.07
Utility performance				
`pack`	11.91 (0.7)	12.56 (10.8)	11.95 (15.2)	12.20 (65.6)
`compress`	10.11 (0.6)	10.72 (8.5)	10.02 (10.5)	10.65 (44.6)
`gzip`	8.66 (0.3)	9.06 (5.0)	8.59 (7.0)	8.55 (30.4)

Table 2: *Compression performance of standard utilities (bpc, decompression time in seconds in parentheses).*

	Corpus			
	small	*stories*	*magazines*	*news*
Character	3,458	5,546	12,486	11,210
Character-pair	42,176	335,371	355,116	813,901
Segmented word	14,277	43,680	51,421	60,981

Table 3: *Numbers of symbols in different models.*

considered as a sequence of two-byte characters, computed as $\sum_{s \in S} -p_s \log_2 p_s$, where p_s is the probability of each symbol s in symbol set S [16]. We also consider the entropy of the data considered as a sequence of character pairs, or digrams, and when segmented into a series of likely "words" (mostly single characters and digrams, but with some trigrams) by a standard dictionary segmentation method [17].

Results are shown in the first block of Table 2. The number of symbols in the models for each of these approaches is shown in Table 3. As can be seen, digram models and segmented-word models lead to similar entropy, but the digram models are much larger. Given reasonable assumptions about model size, the best overall compression is given by the segmented-word models, at some cost in memory compared to single-character models. Digram methods will be poor, outperformed by the single-character and segmented-word models is almost all cases.

We also compressed the test files with three standard compression utilities. The results of these experiments are shown in the second block of Table 2. The utility `pack` uses a semi-static zero-order byte model with Huffman coding. The utilities `compress` and `gzip` both use variants of Lempel-Ziv coding (LZW [6] and LZ77 [8] respectively). As can be seen, only `gzip` achieved compression as good as the character-model entropy, and as we show in the next section can be considerably better. The LZ-style coders rely on local repetition of symbol sequences to achieve good compression, but, compared to English text, such repetition is relatively rare in Chinese. The utility `pack` requires around 0.5 Mb of memory (but only a few tens of kilobytes in principle); the others used just over 1 Mb.

It is interesting to consider how the techniques discussed above might perform on this data. The methods of Ong and Huang [5] and Lua [7] would be decidedly worse than the standard compression utilities, because of the need to either store a large model or code characters with a zero-order model. The method of Tseng and Huang [12] is at best equivalent to zero-order Huffman coding of Chinese characters, and thus will be relatively inefficient. The method of Gu [9] has the drawback of being limited to a relatively small, fixed character set, and is at best an inexact approximation of dynamic Huffman coding [18, 19] or other more principled adaptive techniques. It would thus be surprising if this method was as effective as `gzip`. The best of the variants of Tsay et al. [15] is in principle equivalent to 8-bit PPM with a first-order model; results for 8-bit PPM on this data, shown below, are about as good as `gzip`. Based on our results below, methods like

that of Chen and Chen [2] could be very effective indeed. However, to achieve good compression it is likely that an unrealistic volume of memory is required.

Partial Predictive Matching

One of the most effective general-purpose compression schemes is *partial predictive matching* (PPM) [20, 10], which is adaptive, predictive, uses arithmetic coding to translate model probabilities into codes, and performs extremely well on English text [21]. For these reasons it is attractive to apply PPM to Chinese text.

In standard PPM a symbol s, traditionally a single byte, is encoded as follows. The order of the model—the number of symbols in the context $s_1 \dots s_n$ preceding s—is assumed to be n. If s has previously been observed in the context of $s_1 \dots s_n$ then s and its probability are passed to the arithmetic coder; otherwise, an *escape* code is emitted and PPM encodes s in the order $n-1$ context $s_2 \dots s_n$. In the extreme case of s being a new symbol, $n+1$ escape codes are emitted and s itself is transmitted. Early in the encoding process most contexts and symbols are new, so that many escape codes are emitted, probabilities are inaccurate, and compression is poor; but as more text is seen compression performance improves.

Cleary [11] has shown that the initial model order can vary with each prediction. In this approach, PPM*, the initial model order for each prediction is the lowest one that yields only one prediction. If there is no such deterministic context the longest context is used.

The number of contexts that may have to be maintained by a model of order n for a language with an alphabet of K symbols is K^n. Up to K distinct symbols can occur in each context, so that, with the dynamic data structure used for the model in our implementation of PPM, up to K^{n+1} nodes are required. (This figure is pessimistic for higher-order models—for example, of the 94^3 three-character sequences that might be found in ordinary ASCII text, only a fraction occur in practice; we observed just under 30% in 2 Gb of the English-language TREC data [22].)

The probability for each symbol in a context is computed as the number of previously observed occurrences of the symbol in the context divided by the total number of observed occurrences of any symbol in the context, which must include an estimate of the probability of observing a *novel* symbol, that is, one previously unseen in that context. There are several methods for estimating the number of occurrences allocated to the escape [23]. A successful method for English is to estimate the likelihood of observing a new character as equal to the number of characters observed once or more; this is method C of Bell et al. [1]. With this method the escape probability can never exceed 50% once a context has been observed.

Moffat [24] has shown that by limiting contextual information to a small amount of prior text, English can be effectively compressed using a third-order model. In this scheme, when the model runs out of memory it *flushes* the context information previously built up, and rebuilds the prediction contexts; to avoid the unduly poor compression that occurs when the model is accumulating contexts, it can be first trained on a buffer of immediate prior data after each flush. Moffat has also shown that, by increasing the amount of contextual information remembered, the model is better able to predict the next character and thus compression improves [24].

PPM uses an integer range R to approximate a real interval. To avoid overflows, when the total of the occurrence counts in a context reaches a predefined *halving limit*, all of the counts are halved, taking care to ensure that no counts are reduced to 0. (A variation on this is to start with the counts scaled up so that they occupy all of the available integer range, but this is not practical if there are a large number of symbols in each context.) Halving has another benefit: it prevents the model from stagnating. Characters that were frequently seen early in the text but have not occurred since have their counts progressively halved as other characters are seen, reducing their estimated likelihood as new text is processed. Changing the halving limit forces the model to give more or less weight to recently seen characters.

Since inevitably some characters only occur once in a given context, repeated halving can result in more frequent characters having the same count as rare characters—namely 1. To reduce the impact of this problem, it is preferable to *increment* in larger units, say 8.

	Corpus			
	small	*stories*	*magazines*	*news*
First-order	9.30 (4.4)	9.92 (80.0)	9.47 (94.3)	10.06 (858.2)
Second-order	8.66 (9.4)	8.69 (156.7)	8.13 (190.0)	8.56 (794.6)
Third-order	8.91 (12.3)	8.66 (204.9)	8.26 (268.7)	8.52 (1149.6)

Table 4: *PPM performance with 8-bit coding and* 2^{16} *nodes (bpc, decompression time in seconds in parentheses).*

No. of nodes ($\times$1,024)	Corpus			
	small	*stories*	*magazines*	*news*
64	8.91	8.66	8.26	8.52
128	7.96	8.44	7.84	8.06
256	7.56	8.20	7.50	7.58
512	7.56	8.06	7.17	7.36
1,024	7.56	7.90	6.90	7.10
2,048	7.56	7.68	6.70	6.98

Table 5: *Third-order PPM with 8-bit coding and increased memory (bpc).*

Most implementations encode bytes, but this is an arbitrary choice and any unit can be used, within the constraints of memory size and model order.

For English, contexts and symbols are quickly repeated, so that, after only a few kilobytes of text, good compression is achieved and contexts of as little as three characters can give excellent compression. As byte-oriented PPM is a general method that gives good results not only for English text but for a wide variety of data types, an obvious option is to apply it directly to Chinese text. Results are shown in Table 4, using 8-bit PPM with 2^{16} nodes (or 896 Kb of memory). These results are slightly better than those produced by the compression utilities listed in Table 2, although at over 4 bits per byte they do not approach the 2.2 bits per byte that the same implementation can yield for English text [24].

In two of the four files the second-order model gives better results than the third-order model. Higher order models take a longer time to accumulate contexts with probabilities that accurately reflect the distribution, so that, when memory is limited, the model spends most of its time in the "learning" phase, where it emits large numbers of escape codes and is unable to make profitable use of the contexts it is accumulating. Thus we observe poorer compression because such contexts do not reappear sufficiently often before the model needs to be flushed and rebuilt—over 800 rebuilds were required for *news* with a third-order model. Reloading the model with immediate prior text after each flush is unlikely to be helpful, since the problem is that there is not sufficient memory to hold a model that makes accurate predictions. It follows that increasing the amount of memory available for storing contexts could be expected to improve compression performance. Table 5 shows results with increased memory. The code and data for the 2^{21} nodes in the final line of the table required over 50 Mb of memory to execute, and moreover accesses to the data structure do not exhibit any locality; so over 50 Mb of real memory is required. Even so, the model had to be flushed several times.

Our interest is in allowing compression to proceed in more modest amounts of memory, and for the rest of the paper focus on examining the performance that can be achieved in around 1 Mb, or about the memory consumed by 2^{16} nodes.[2] As noted above, PPM* implementations described by Cleary [11] and modified by Bunton [20] give slightly improved compression but require much more memory, potentially up to one node for each character in the text to be compressed.

[2]For 2^{16} nodes or fewer, 16 bit pointers can be used, yielding a space saving of about 35%.

	Corpus			
	small	*stories*	*magazines*	*news*
limit of 1,024, increment of 16				
first-order	7.20	7.69	7.07	7.38
second-order	7.38	7.79	7.32	7.44
third-order	7.60	7.94	7.55	7.67
limit of 64, increment of 8				
second-order	8.20	8.67	8.09	8.24

Table 6: *PPM performance with 16-bit coding and* 2^{16} *nodes (bpc).*

Modifying PPM for Chinese Text

For Chinese text, the byte-oriented version of PPM is not predicting characters, but rather halves of characters. It is reasonable to suppose that modifying PPM to deal with 16-bit characters should enable the model to more accurately capture the structure of the language, and hence provide better compression.

We have identified several changes that need to be made to the PPM implementation described above to allow effective 16-bit coding of Chinese. First, the halving limit needs to be modified: the number of 16-bit characters that can occur in a context is much greater than of 8-bit characters, so a larger probability space is required. Second, in conjunction with this change the increment should also be increased, to force more frequent halving and prevent the model from stagnating. Our experiments suggest that a halving limit of 1,024 and an increment of 16 are appropriate. Third, the method described above for estimating escape probabilities may not be appropriate since so many characters are novel; this is discussed further below. Fourth, model order must be chosen.

Results of our experiments with 16-bit coding are shown in Table 6. (Only compression efficiency is reported; speed is considered later.) Changing the unit of encoding from 8 bits to 16 bits results in a small increase in node size, of less than 10%; thus 2^{16} nodes can still be stored in less than 1 Mb of memory. We have also shown in the last line of the table second-order performance with the ASCII halving limit of 64 and increment of 8; the changes to these values have made a significant difference. In effect, use of 16-bit characters has allowed deeper contexts to be captured for only a small increase in memory expenditure.

For even the longest file, first-order compression has worked best. In all of the files PPM had frequently to escape to lower contexts, particularly in the third-order model, because in the majority of cases characters in the longest context are novel. Even in the second-order model, only 25% of characters were coded in the highest context (suggesting that the method used to calculate the escape probability was not appropriate, since with the current method of calculation it cannot exceed 50%; this is considered further below). It is thus unlikely that use of PPM* would yield improvements in compression. For comparison, with English text encoded in bytes with a third-order model 86% of characters are encoded in the highest context [1]. Yet it is the higher-order models that hold the most promise for yielding good compression. For this reason we have concentrated on improving the performance of compression with the second-order model.

Our results are consistent with those of Gu [14], who evaluates zero-order and first-order prediction with a PPM-like compressor on two files of around 200 Kb each. However, as our results show, compression of larger files requires use of a memory management scheme, an issue that Gu does not appear to have considered.

The skewed distribution of character occurrences in Chinese means that many characters occur only once in a context. Such occurrences occupy space in the model but contribute little to its predictive power. This suggests that the flushing mechanism may be problematic. In the context of

English, flushing discards old information to make space for new information, which can be quickly learnt. In the context of Chinese, there is more to learn; yet much of what is learnt is effectively noise—characters that occur in a context by chance. The "learning phase" phenomenon is not as simple as seeing most of the contexts that occur in the text. Some contexts occur frequently, and so characters which occur in these contexts can be coded compactly, but this efficiency is offset by seeing many novel contexts that require an escape to lower-order contexts. It would appear that the key to the design of good contextual models for Chinese is the judicious use of memory. In the next section we explore a modification to PPM that uses a different mechanism for memory management.

The scheme described by Chen and Chen [2] is similar to PPM, but is based on text segmented into words rather than on characters. As Table 3 shows, such as scheme must deal with a set of tens of thousands of symbols. With unlimited memory, it is likely that segmented text would lead to excellent compression. However, assuming only moderate volumes of memory are available, managing even a character-based model can be problematic; we believe that, because of the number of distinct symbols, use of a word-based model is unlikely to be valuable.

Memory Management for PPM

The implementation of PPM described above uses a simple memory management strategy: all information is discarded when the available space is consumed. We propose instead that the model be *pruned*—that is, individual nodes should be removed if by some criteria they are deemed to be unlikely to be necessary. Since the majority of the nodes are used to represent characters in the highest contexts, we can simplify the structure management by only applying this removal criteria to these nodes, then remove any nodes in lower contexts that no longer have any children. The problem is then to decide what these criteria might be.

Williams has investigated this idea, particularly in relation to his Dynamic History Predictive Compression (DHPC) algorithm [25]. His work was mainly concerned with heuristics used to determine when to add contexts to a model. There are two simple criteria that might be used to choose nodes to discard: distance since last observation, and total occurrence count. With the distance approach, a field is added to each node to store the byte position at which it was last observed. When memory is exhausted, this information can be used to discard characters that are no longer being used. Use of distance is conceptually similar to the technique, with flushing, of reinitialising the model on a buffer of immediate prior text. In the flushing case, the count (and hence probability) associated with a node depends solely on the number of occurrences in the prior text used to rebuild the model. In contrast, with distance-based pruning a node that is retained has its total occurrence count. If the text changes only slowly, this may be a better estimate of its probability than that obtained from a relatively small sample of immediate prior data.

Tests on the *magazines* file with distance-based pruning showed a modest improvement of 0.1–0.2 bpc when using a model of 2^{18} or more nodes, but compression declined for smaller model sizes, as can be seen in the the second line of Table 7. It appeared that the approach was too coarse a pruning mechanism, and that it was not desirable to discard information about all events that had not occurred since a certain point in the message. Williams [25] also found that the use of distance information did not contribute significantly to compression.

The second of the criteria is to remove all nodes for which the occurrence count is below a given threshold. To ensure that a reasonable number of nodes are removed at each iteration, removal should be iterated with an increasing threshold until enough nodes have been discarded; for our experiments we defined "enough" to be one quarter. A problem with this approach is that it can lead to stagnation in the model, as recent characters might be discarded in favour of other characters that were once frequent, and because pruning reduces the incidence of halving. This approach is illustrated in the third line of Table 7. A refinement of this approach is to halve all count values at each pruning, so that old information has a reduced count and is eventually discarded. Results with this approach are shown as the fourth line of Table 7, showing only an inconsistent improvement. Although an increased number of characters were being coded

	Corpus			
	small	*stories*	*magazines*	*news*
Second-order PPM				
Standard 16-bit	7.38	7.79	7.32	7.44
Distance pruning	7.10	8.25	7.39	8.58
Count pruning	7.04	7.95	6.91	8.31
Count with halving	7.07	7.99	7.08	8.21
Incremented escape	7.04	7.58	6.90	7.33
First-order PPM				
Incremented escape	7.10	7.49	6.78	7.12

Table 7: *PPM with 16-bit coding and* 2^{16} *nodes (bpc).*

using higher-order contexts compared to flushing—up from 25% to 28%—compression was worse, suggesting problems with the estimation of escape probabilities. None of the standard methods for estimation [1] are likely to be superior, so we therefore considered new methods.

Estimating the Escape Probability

As noted above, the high frequency of use of escape suggests that the standard techniques for estimating escape probabilities may not be appropriate. Our initial implementation of PPM used method C, where the probability of seeing a novel character is estimated as

$$\text{pr(novel)} = \frac{r}{n + r}$$

where r is the number of distinct characters seen in the context and n is the total number of character occurrences in the context. These values in turn must be estimated, due to the effect of halving; this is done by incrementing an explicit counter for r when a novel character is observed, and by incrementing an explicit counter for n (actually for $n + r$) whenever any character is observed. Both n and r, like other character counts, need to be halved.

With flushing, PPM need only be concerned with incrementing r when novel characters are seen. However, with pruning PPM also needs to consider r when nodes are removed. In the experiments described above, the approach used when pruning a character from a context was to simply decrement n by the count of the character. Supposing that the counts for deleted characters summed to n_d, this means that the probability of seeing a novel character is estimated as

$$\text{pr}'\text{(novel)} = \frac{r}{(n - n_d) + r}$$

Note that $\text{pr(novel)} \leq \text{pr}'\text{(novel)}$. With this approach, when a pruned character reappears the value of r is incremented again, because the character is once again novel in this context.

Another possibility is to consider theoretical approaches to estimation of escape counts. Witten and Bell show that the Poisson distribution can be used to estimate the probability of a novel symbol from the counts of symbols that have occurred one or more times [26]. But this method X suffers from three problems in the context of our experiments. First, after pruning there are no observations of characters that had occurred once, so that estimated escape probabilities are negative! Second, true estimation of these counts is difficult in the presence of halving. Third, for the higher orders there were typically only a few observed characters in each context, a statistically insufficient number to give reasonable confidence in the estimated escape. Witten and Bell have also observed this last problem, noting that method X “frequently breaks down” [26], and provide ad hoc solutions, with mixed results.

Moreover, these approaches do not make good use of the extra information that pruning makes available. Proceeding from the assumption that (prior to any flushing or pruning) the estimated

escape probability is an accurate reflection of the likelihood of seeing novel characters, then after the removal of some character `c`, it should be an estimate of seeing other novel characters *or* of seeing `c`. Under this assumption, the count for `c` should be added to r when `c` is pruned, so that escape probabilities are estimated as

$$\text{pr}''(\text{novel}) = \frac{r + n_d}{(n - n_d) + (r + n_d)} = \frac{r + n_d}{r + n}$$

Note that $\text{pr}'(\text{novel}) \leq \text{pr}''(\text{novel})$. Re-occurrence of `c` in this context agains increment r, an unavoidable consequence of `c` having being pruned (but note that the previous contributions of `c` to r are diluted by halving). Nonetheless, experiments with this approach gave much better results, and are shown as the second-last line of Table 7.

These results can be further improved. For flushing, first-order 16-bit PPM gave the best results—though not as good as those of second-order 16-bit PPM with pruning, so we tested first-order 16-bit PPM with pruning. Results are shown in the last line of Table 7 and as can be seen the results are an improvement on the line above. Thus we have not succeeded in our aim of making optimal use of second-order prediction, but we have further improved the possible compression of Chinese text, by almost 1.5 bpc compared to using third-order 8-bit PPM. It seems that 2^{16} nodes is insufficient for a second order model to operate effectively, even with pruning. Given larger model sizes, second-order models become more effective than first order; for example, with 2^{18} nodes (approximately 4 Mb) we achieved an improvement of 2.0 bpc using a second-order model, and 1.6 bpc using a first-order model, compared to third-order 8-bit PPM.

The combined effect of these changes is a scheme that yields a consistent improvement in compression when compared to the flushing approach. The results presented so far have been for models of 2^{16} nodes, but similar improvements in compression relative to the flushing model were achieved for all numbers of nodes tested. Figure 1 shows compression results obtained. It can be seen that the pruning approach obtains the same compression performance as flushing using less than one third of the memory.

When the symbol alphabet is large, as is the case with Chinese, only a modest number of symbols can be coded in the highest order unless vast amounts of memory are used. In such circumstances, the accurate calculation of the escape probability is crucial to good compression performance. In other schemes, the escape probability represents at best an educated guess as to what the frequency of appearance of previously unobserved symbols might be. However in the pruning model, most of the "novel" symbols have actually been observed, and so it is possible to build up an escape probability estimate that combines the guesswork with actual evidence.

Performance

Our initial implementation was somewhat slow, decoding around 5–10 Kb per second. This was because many of the contexts contain many predictions, which were searched linearly because the arithmetic coder requires the cumulative probability of all symbols in the context up to the one being coded (the coder imposes a symbol ordering), and a linear scan of the symbols is a straightforward way to calculate this. This approach is only a problem in popular contexts. It is largely solved by the Fenwick tree [27], a structure by which cumulative counts (probabilities) can be calculated in $\log_2 n$ operations. Our revised implementation made use of Fenwick trees to store the context information. Timed on a Sun Sparc 10 Model 512, it can decompress at about 60 Kb per second or 4 Mb per minute—roughly the same speed as standard PPM but much slower than `gzip`, which can decompress around 90 Mb per minute. Our code is not highly refined and we expect that further speed improvements would be possible, but, as for other implementations of PPM, speed comparable to that of `gzip` is unattainable.

One significant drawback of the Fenwick tree is that it is not possible to implement exclusions, since each different symbol that is excluded results in a different cumulative probability count. Without exclusions, performance degrades by 0.1–0.2 bpc, giving compression that is still rather better than that of other methods.

Conclusions

We have investigated compression of Chinese text, a large alphabet language with relatively little structure. Previous methods for compressing Chinese have focused on two alternative approaches to modelling, use of a dictionary and use of context for prediction. The dictionary approaches are hampered by the large number of distinct character strings—and hence large model size—in even short Chinese texts, and are not as effective as the standard general-purpose compression utilities. In contrast, the adaptive, predictive methods can yield much greater compression. We have examined PPM, the most effective compression technique for English text, and have shown that it is highly effective for Chinese.

We have identified several changes that need to be made to PPM to yield the greatest compression of Chinese text within given memory bounds. In particular, the unit of encoding should be changed from 8 bits to 16 bits; more use should be made of the probability space available in arithmetic coding; new methods need to be found to estimate the escape probabilities; and it is helpful to have a memory management scheme that allows learning of contexts over long texts, to improve the model's prediction capabilities. We have investigated both of the last two options in detail, and have used them to yield small but consistent improvements in compression, achieving the same compression performance in one third of the memory.

Together our modifications allowed PPM to reduced the space needed to represent Chinese text by almost 1.5 bits per character within 1 Mb of memory. With more memory even greater compression is available, to as little as 6.2 bits per character for one data set, or a saving of 2.5 bits per character compared to the utility `gzip` or over 3 bits per character compared to Huffman coding. We expect that our modifications to PPM to yield similar improvements in compression for other large alphabet languages, such as Japanese and Korean.

We have observed that our implementation of PPM tuned for Chinese works well on English texts, particularly long texts, with second-order 16-bit PPM compressing about as well as fifth-order 8-bit PPM. This is a welcome discovery if only because in practice Chinese text often contains embedded text from other languages, but it also suggests that our methods could be successfully applied to other domains. Also, not only could our refinements to PPM potentially be applied to character-level compression of English text, but they might be applied to word-level predictive compression of English text, another application in which large symbol sets are to be managed in limited memory.

Acknowledgements

We thank Alistair Moffat, for his advice and for the initial implementation of PPM used in our experiments, and George Fernandez. This work was supported by the Australian Research Council.

References

[1] T.C. Bell, J.G. Cleary, and I.H. Witten, *Text Compression*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[2] J.N. Chang, J.S. Chang, and M.H. Chen, 'Using word-segmentation model for compression of Chinese text', *Computer Processing of Chinese and Oriental Languages*, **9**(1), 17–30 (1995).

[3] Zhonghua Bookstore, *Kang Xi Zidian*, seventh edition, 1989. In Chinese.

[4] I.H. Witten, R.M. Neal, and J.G. Cleary, 'Arithmetic coding for data compression', *Communications of the ACM*, **30**(6), 520–541 (1987).

[5] G. H. Ong and S. Y. Huang, 'Compression of Chinese text files using a multiple four-bit coding scheme', T.S. Ng C.S., Yeo and Yeo S.P. (eds.), *Communications on the move ICCS/ISITA*, Singapore, 1992, pp. 1062–1066.

[6] T.A. Welch, 'A technique for high performance data compression', *IEEE Computer*, **17**, 8–20 (1984).

[7] K. T. Lua, 'Compression of Chinese text', *International Conference on Chinese Computing '94 (ICCC94)*, Singapore, June 1994, pp. 367–375.

[8] J. Ziv and A. Lempel, 'A universal algorithm for sequential data compression', *IEEE Transactions on Information Theory*, **IT-23**(3), 337–343 (1977).

[9] H.-Y. Gu, 'A new Chinese compression scheme combining dictionary coding and adaptive alphapbet–character grouping', *Computer Processing of Chinese and Oriental Languages*, **10**(3), 321–335 (1997).

[10] J.G. Cleary and I.H. Witten, 'Data compression using adaptive coding and partial string matching', *IEEE Transactions on Communications*, **COM-32**, 396–402 (1984).

[11] J.G. Cleary, W. Teahan, and I.H. Witten, 'Unbounded length contexts for PPM', J.A. Storer and M. Cohn (eds.), *Proc. IEEE Data Compression Conference*, Snowbird, Utah, 1995, pp. 52–61.

[12] L.Y. Tseng and T.H. Huang, 'A predictive coding method for Chinese text file', *Journal of Computers*, **2**(3), 18–23 (1990).

[13] H.K. Chang and S.H. Chen, 'Extended predictive data coding scheme for Chinese text files', *Computer Processing of Chinese and Oriental Languages*, **7**(2), 257–271 (1993).

[14] H.-Y. Gu, 'Large-alphabet Chinese text compression using adaptive Markov model and arithmetic coder', *Computer Processing of Chinese and Oriental Languages*, **9**(2), 111–124 (1995).

[15] M.K. Tsay, C.H. Kuo, R.H. Ju, and M.K. Chou, 'Modelling for Chinese text file compression', *International Conference on Industrial Applications of Computing*, Alexandria, Egypt, 1992, pp. 205–208.

[16] C.E. Shannon, 'Prediction and entropy of printed English', *Bell Systems Technical Journal*, **30**, 55 (1951).

[17] V. B. H. Nguyen, P. Vines, and R. Wilkinson, 'A comparison of morpheme and word based document retrieval for Asian languages', R. R. Wagner and H. Thoma (eds.), *International Conference on Database and Expert System Applications — DEXA 96*, Zurich, Switzerland, 1996, pp. 24–33. Springer.

[18] D.E. Knuth, 'Dynamic Huffman coding', *Journal of Algorithms*, **6**, 163–180 (1985).

[19] J.S. Vitter, 'Algorithm 673: Dynamic Huffman coding', *ACM Transactions on Mathematical Software*, **15**(2), 158–167 (1989).

[20] S. Bunton, 'Semantically motivated improvements for ppm variants', *Computer Journal*. To appear.

[21] A. Moffat, J. Zobel, and N. Sharman, 'Text compression for dynamic document databases', *IEEE Transactions on Knowledge and Data Engineering*, **9**(2), 302–313 (1997).

[22] D. Harman, 'Overview of the second text retrieval conference (TREC-2)', *Information Processing & Management*, **31**(3), 271–289 (1995).

[23] T.C. Bell, I.H. Witten, and J.G. Cleary, 'Modeling for text compression', *Computing Surveys*, **21**(4), 557–592 (1989).

[24] A. Moffat, 'Implementing the PPM data compression scheme', *IEEE Transactions on Communications*, **38**(11), 1917–1921 (1990).

[25] R.N. Williams, *Adaptive Data Compression*, Kluwer Academic, Norwell, Massachusetts, 1991.

[26] I.H. Witten and T.C. Bell, 'The zero frequency problem: Estimating the probabilities of novel events in adaptive text compression', *IEEE Transactions on Information Theory*, **37**(4), 1085–1094 (1991).

[27] P.M. Fenwick, 'A new data structure for cumulative probability tables', *Software—Practice and Experience*, **24**(3), 327–336 (1994). Errata published in 24(7):677, July 1994.

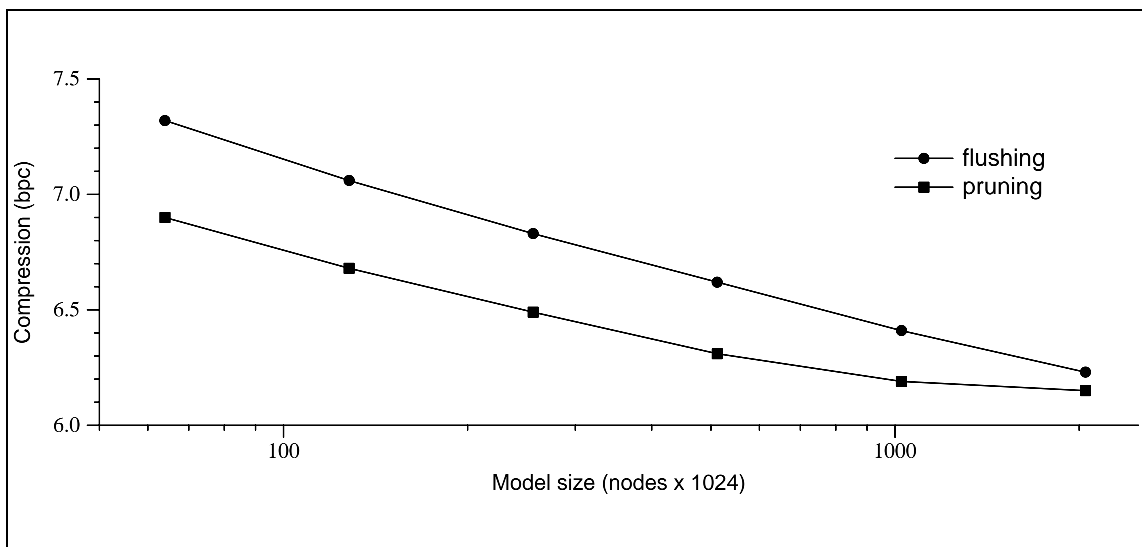

Figure 1: *Compression performance for pruning and flushing on* magazines.