

---

# A large-alphabet-oriented scheme for Chinese and English text compression



Hung-Yan Gu<sup>\*,†</sup>

*Department of Computer Science and Information Engineering,  
National Taiwan University of Science and Technology, Taipei, Taiwan*

---

## SUMMARY

**In this paper, a large-alphabet-oriented scheme is proposed for both Chinese and English text compression. Our scheme parses Chinese text with the alphabet defined by Big-5 code, and parses English text with some rules designed here. Thus, the alphabet used for English is not a word alphabet. After a token is parsed out from the input text, zero-, first-, and second-order Markov models are used to estimate the occurrence probabilities of this token. Then, the probabilities estimated are blended and accumulated in order to perform arithmetic coding. To implement arithmetic coding under a large alphabet and probability-blending condition, a way to partition count-value range is studied. Our scheme has been programmed and can be executed as a software package. Then, typical Chinese and English text files are compressed to study the influences of alphabet size and prediction order. On average, our compression scheme can reduce a text file's size to 33.9% for Chinese and to 23.3% for English text. These rates are comparable with or better than those obtained by popular data compression packages. Copyright © 2005 John Wiley & Sons, Ltd.**

**KEY WORDS:** text compression; large alphabet; Markov modeling; arithmetic coding

## 1. INTRODUCTION

One notable result was obtained from our previous study [1]. It was shown that when the max prediction order of the probability estimation models is fixed to 1, a Chinese text file can be much better compressed (in compression rate) if the input text is parsed into tokens with a large alphabet instead of a small alphabet. This may be attributed to the fact that Chinese is a large-alphabet language and has more than 10 000 characters. On the other hand, it is interesting to consider whether an English text file would be better compressed if the input text was parsed with a small alphabet

---

\*Correspondence to: Hung-Yan Gu, Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan.

†E-mail: guhy@mail.ntust.edu.tw

instead of a large alphabet. In many previous studies, a small alphabet was chosen to compress English text files [2–4]. This choice is apparently due to English being a small-alphabet language with less than 100 characters. Nevertheless, according to the experiments conducted, our large-alphabet-oriented compression scheme can also compress English text files as effectively as other popular data compression packages such as BZIP2 [5] and PPMd [6]. Therefore, our scheme can improve Chinese text files' compression rates, while not degrading English text files' compression rates where the max prediction order is set to only 2.

In this paper, 'large alphabet' does not mean a word-based alphabet [3,7,8]. We do not consider a word-based large-alphabet approach because we want a more general one that can be applied to both English and Chinese text. An English sentence can be simply parsed into a sequence of words in terms of the delimiter characters, e.g. the blank and punctuation characters. But there are no such delimiter characters between adjacent words in a Chinese sentence, and to parse out words of a Chinese sentence needs semantic knowledge. Therefore, for Chinese text, we just parse input text into tokens of Chinese characters according to the coding rules of Big-5 Chinese code. As for English text, each time we just slice off a single character or two consecutive characters to form a token according to some parsing rules.

The large-alphabet-oriented compression scheme proposed here can be viewed basically as extended from that in our previous work [1]. First, we study and design parsing rules for parsing English text into tokens appropriate for large-alphabet-oriented processing. Secondly, we extend the model's max prediction order from 1 to 2, and relax the memory size restriction in the model's data structure. This model-order extension is essential for obtaining significantly better compression rates for both English and Chinese text. When the model order is extended, the large-alphabet arithmetic coder implemented in our previous work must also be extended.

When an input text has been parsed into tokens, the remaining processing steps are the same for both Chinese and English text. To help get an overview of our compression scheme, we draw the main processing steps within the flowchart in Figure 1. After initialization of the models and the coder, a token is parsed out from the input text for each iteration of the loop. The token parsed out is re-represented with a number  $v$  if the token is the  $v$ th element of the alphabet. If no further token can be sliced off, the loop is broken and compression processing stops after finalizing arithmetic coding. If a token is sliced off, occurrence probabilities are estimated for this token with zero-, first-, and second-order Markov models respectively. Then the estimated probabilities are blended and cumulated. Because of probability blending, our scheme is not a PPM (prediction by partial matching) compression scheme [4,9]. According to the cumulated probability interval, arithmetic coding is performed and common leading bits are outputted [3,4]. In the last step of the loop, the same token is used to update the Markov models. More detailed explanations for the flowchart blocks are given in latter sections.

## 2. LARGE-ALPHABET-ORIENTED PARSING

For the representation of Chinese text in computers, Big-5 code is the commonly adopted one in Taiwan. In Big-5 code, there are 13 943 Chinese characters and symbols defined. Each character is represented with two consecutive bytes. The first byte has a value from 161 to 254, and the second byte has a value from either 64 to 126 or 161 to 254. Therefore, ASCII characters are allowed to be scattered in a text file where Big-5 code is used to represent Chinese text.

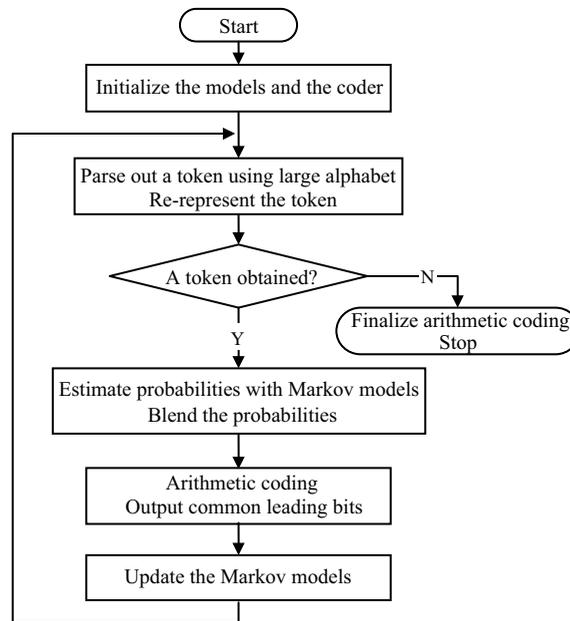


Figure 1. Main flow of the large-alphabet-oriented compression scheme.

For Chinese text parsing, a byte is taken from the input text and checked to see if it is greater than or equal to 161. If not, this byte is sliced off as an ASCII-character token. If it is greater than or equal to 161, a successive byte is taken immediately and its value is checked to see whether it is within one of the two allowed ranges. If not, the second byte is put back to the input stream and the first byte is sliced off as an ASCII-character token. If yes, the two bytes indeed represent a Chinese character and are treated as a token. When an ASCII-character token is parsed out, its re-representation value is assigned the same as its original value. When a Chinese-character token is parsed out, its re-representation value is computed as  $(B_1 - 161) * 157 + (B_2 - B_r) + 256$ . Here,  $B_1$  and  $B_2$  are the values of the first and second bytes respectively, and  $B_r$  is set to be 64 or 98 depending on whether  $B_2$  is within the range 64–126 or 161–254. Hence, a Chinese character is re-represented with a value from 256 to 14 198 and stored as a double-byte integer. The alphabet used here for Chinese text compression is therefore as large as 14 199 characters.

For English text parsing, a usual way is to slice off each byte from the input stream as a token. Such a method needs only a small alphabet of 256 characters, and is small-alphabet-oriented parsing. Here, large-alphabet-oriented parsing is considered. A type of large-alphabet-oriented parsing is to slice off a word as a token. But this parsing method will result in very irregular token lengths from 1 to 15 (or more) bytes, and the total number of different tokens or the alphabet size cannot be known in advance. If the elements of an alphabet cannot be known in advance, some kind of alphabet-adjusting

mechanism must be implemented to let the sender inform the receiver what new element is to be added and when it is to be added. Such an alphabet-adjusting mechanism will incur an overhead and degrade the compression rate if the prediction order of the model is lower than one. On the other hand, a word-based prediction model of an order higher than zero is very complicated and will be difficult to implement. This difficulty may explain why we find, from an Internet search, no word-based compression software that supports a prediction order higher than zero and can be downloaded for testing.

Therefore, we study other parsing methods in order that tokens have more regular lengths and the alphabet size can be known beforehand. The parsing methods studied here for English text have been determined by the way that Chinese text is parsed. That is, we restrict a token to only two possible lengths, 1 or 2 bytes. To parse out a double-byte token, we need to define a set,  $A_c$ , of allowable characters beforehand. In this paper,  $A_c$  is defined to contain the ASCII graphical characters and two frequently used control characters. That is,  $A_c$  contains those characters of ASCII values 32–126, 10 and 13, and thus has 97 elements. With respect to  $A_c$ , the number of different double-byte tokens is  $97 * 97 = 9409$ , and the alphabet size is that number plus 256 for different single-byte tokens. A double-byte token,  $B_1 B_2$ , when parsed out, can be re-represented with a value computed as  $(B_1 - 32) * 97 + (B_2 - 32) + 256$ . Here, we re-map the ASCII value 10 to 127 and 13 to 128 before that computation.

In this paper, two parsing methods are studied. The first is called uniform parsing, and the second is called synchronous parsing. In the uniform-parsing method, two bytes,  $B_1$  and  $B_2$ , sliced off successively from the input stream will be parsed out as a double-byte token if the rule

(R1)  $B_1$  belongs to  $A_c$  and  $B_2$  belongs to  $A_c$

is satisfied. Otherwise,  $B_1$  only is parsed out as a single-byte token and  $B_2$  is put back. Next, in the synchronous-parsing method, three bytes,  $B_1$ ,  $B_2$  and  $B_3$ , are sliced off successively from the input stream. Then,  $B_1$ ,  $B_2$  and  $B_3$  are checked to see if any of the three rules

(R2)  $B_1$  is not a letter, and  $B_2$  and  $B_3$  are letters,

(R3)  $B_1 \geq 32$ , and  $B_2 < 32$  and  $B_3 < 32$ ,

(R4)  $B_1 < 32$ , and  $B_2 \geq 32$  and  $B_3 \geq 32$ ,

is satisfied. If any one of the rules, (R2), (R3), and (R4), is satisfied or the rule (R1) is not satisfied, then  $B_1$  only is parsed out as a single-byte token and  $B_2$  and  $B_3$  are put back. Otherwise,  $B_1$  and  $B_2$  are parsed out as a double-byte token and  $B_3$  is put back. With the synchronous-parsing method, the first two letters of a word are guaranteed to be parsed out as a token.

### 3. PROBABILITY ESTIMATION AND BLENDING

In this paper, Markov models of zero, first and second orders are adopted for probability estimation. For the details of Markov modeling refer to Bell *et al.* [3], Sayood [4] or our previous work [1]. Suppose the sequence of tokens parsed out from the input text is  $X_1, X_2, \dots, X_T$ . Let us define the

count variables  $N_i(v)$ ,  $N_i(u, v)$  and  $N_i(s, u, v)$  respectively as

$$N_i(v) = \sum_{j=1}^i C_j(v), \quad C_j(v) = \begin{cases} 1, & \text{if } X_j = v \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$N_i(u, v) = \sum_{j=1}^i C_j(u, v), \quad C_j(u, v) = \begin{cases} 1, & \text{if } X_{j-1} = u \text{ and } X_j = v \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

$$N_i(s, u, v) = \sum_{j=1}^i C_j(s, u, v), \quad C_j(s, u, v) = \begin{cases} 1, & \text{if } X_{j-2} = s, X_{j-1} = u \text{ and } X_j = v \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

That is,  $N_i(s, u, v)$  counts, from time 1 to time  $i$ , the number of times that three consecutive tokens are found to have values  $s, u, v$ , respectively. Then, at time  $i$  to compress the token  $X_{i+1}$ , the probability of the zero-order Markov model (ZOMM) is estimated as

$$P_0(X_{i+1}) = P(X_{i+1}) = \frac{N_i(X_{i+1}) \cdot Nd + 1}{i \cdot Nd + Ns} \quad (4)$$

where  $Ns$  is the size of the alphabet and  $Nd$  is the increment added each time in updating the model.  $Nd$  is usually set to be 1 when a small alphabet is adopted. However, with a large alphabet,  $Nd$  must be set to a larger value, e.g. 16, to have the model, ZOMM, adapted faster. Besides the zero-order model, the probabilities of the first- and second-order Markov models (FOMM and SOMM) are estimated respectively as

$$P_1(X_{i+1}) = P(X_{i+1} | X_i) = \frac{N_i(X_i, X_{i+1})}{N_{i-1}(X_i)} \quad (5)$$

$$P_2(X_{i+1}) = P(X_{i+1} | X_{i-1}, X_i) = \frac{N_i(X_{i-1}, X_i, X_{i+1})}{N_{i-1}(X_{i-1}, X_i)} \quad (6)$$

In this paper, we decide to adopt probability blending instead of escaping used in PPM schemes. To blend the three probabilities estimated in Equations (4), (5), and (6), we still use escape probabilities estimated with Turing's formula [10,11] to set the weights for the three Markov models. The escape probabilities  $Pe_1$  from first to zero order and  $Pe_2$  from second to first order are estimated respectively as

$$Pe_1 = \frac{1 + M_i(X_i)}{2 + N_{i-1}(X_i)}, \quad M_i(X_i) = \sum_{u=1}^{Ns} D_i(X_i, u), \quad D_i(X_i, u) = \begin{cases} 1, & \text{if } N_i(X_i, u) = 1 \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

$$Pe_2 = \frac{1 + M_i(X_{i-1}, X_i)}{2 + N_{i-1}(X_{i-1}, X_i)}, \quad M_i(X_{i-1}, X_i) = \sum_{u=1}^{Ns} D_i(X_{i-1}, X_i, u)$$

$$D_i(X_{i-1}, X_i, u) = \begin{cases} 1, & \text{if } N_i(X_{i-1}, X_i, u) = 1 \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

Here,  $M_i(X_i)$  counts the number of alphabet elements that occur one time with  $X_i$  as their predecessor, and  $M_i(X_{i-1}, X_i)$  similarly counts those with  $X_{i-1}$  and  $X_i$  as predecessors. The estimation formulae of Equations (7) and (8) are used in compressing Chinese text. For English text compression it is better

to decrease the escape probabilities according to our experimental results. In fact, we add a half of the numerator value to the denominator for both Equations (7) and (8). That is,  $2 + N_{i-1}(X_i)$  is changed to  $2 + N_{i-1}(X_i) + M_i(X_i)/2$  in Equation (7) and a similar adjustment is made in Equation (8). In terms of the two escape probabilities, the probabilities of the three Markov models are blended as

$$P_b(X_{i+1}) = Pe_1 \cdot P_0(X_{i+1}) + (1 - Pe_1)Pe_2 \cdot P_1(X_{i+1}) + (1 - Pe_1)(1 - Pe_2) \cdot P_2(X_{i+1}) \quad (9)$$

The adoption of this way of blending is because practical experimental results show it is slightly better than the usual way of blending as in Equation (10).

$$P_b(X_{i+1}) = (1 - Pe_2) \cdot P_2(X_{i+1}) + Pe_2(1 - Pe_1) \cdot P_1(X_{i+1}) + Pe_2Pe_1 \cdot P_0(X_{i+1}) \quad (10)$$

Although Equations (4), (5), (6), (7), and (8) can be directly used to implement a computer program, a careful selection of the data structure helps to obtain better computation efficiency. In this paper, we use the large-alphabet accumulation-supported binary search tree structure (token values are treated as the keys) proposed by Bell *et al.* [3] to store the counts for computing  $P_0(X_{i+1})$ . In a tree node, besides the ordinary data fields, token value and occurrence count, an additional data field is used to store the above-count of this node's left sub-tree. In our programs, 500 000 tree nodes are allocated and maintained for repeated use. For computing  $P(X_{i+1}|X_i)$  we also adopt such a tree structure, and build a tree for each encountered value of  $X_i$  to store the occurrence counts  $N_i(X_i, v)$ . Also, two arrays are maintained to store the counts  $N_{i-1}(X_i)$  and  $M_i(X_i)$  for different values of  $X_i$ . In addition, consider the computation of  $P(X_{i+1}|X_{i-1}, X_i)$ . The number of possible combinations of the values of  $X_{i-1}$  and  $X_i$  is huge (e.g.,  $14\,199 \times 14\,199 \approx 200$  M) but only a small portion will occur practically. Hence, we allocate a hash table of sufficient size, almost 150 000 entries, to store the encountered value pairs of  $X_{i-1}$  and  $X_i$ , and their corresponding  $N_{i-1}(X_{i-1}, X_i)$  and  $M_i(X_{i-1}, X_i)$ . Also, a tree of the above-mentioned structure is built and associated with each value pair of  $X_{i-1}$  and  $X_i$ . When a new value pair is to be inserted while the load factor of the hash table exceeds a threshold value, we will free a least recently used (LRU) entry and its associated tree entirely. In another condition that no tree node is available, we will free an LRU tree node and adjust the influenced counts' values. Then this tree node can be reused. We need two extra linked lists to implement LRU policy, one for the hash table entries and one for the tree nodes.

#### 4. ARITHMETIC CODING

The alphabet used for Chinese text compression has 14 199 characters and the one used for English has 9665 characters. These alphabet sizes indicate that the denominator in Equation (4),  $i \cdot Nd + Ns$ , will have value of at least 14 199 or 9665. Also, consider that an integer is represented with 32 bits in most PCs. This implies that a count value cannot exceed  $2^{16}$  because it will be multiplied with a value of similar scale in arithmetic coding and the result cannot exceed  $2^{32}$ . However, the range between 14 199 and  $2^{16}$  is not wide enough to accommodate the updating of the count value,  $i \cdot Nd + Ns$ . Furthermore, we need a larger range to represent the values obtained in cumulating the counts from the three Markov models. Here, across-models count accumulation is done in a weighted way according to the probability blending weights in Equation (9). Therefore, we resort to the hardware-supported floating point number processing available in most modern PC CPUs. When declared with the 'long double float' type, a variable will have a precision of more than 64 bits. This implies that a count value

or variable used in arithmetic coding can be as large as  $2^{30}$ . Here 2 (32 – 30) bits are reserved for rescaling processing in arithmetic coding.

Although the range of 30 bits is available now, not all 30 bits can be allocated to represent the value of the ZOMM count,  $i \cdot Nd + Ns$ , because several bits must be reserved to accommodate the representation and accumulation of the weighted FOMM and SOMM counts. Actually, we represent  $i \cdot Nd + Ns$  with a 21-bit range. Hence, when the value of  $i \cdot Nd + Ns$  reaches  $2^{21}$ , all the values of the ZOMM counts,  $N_i(v)$ , will be halved immediately (note that the halved value of 1 is still 1). As to the FOMM counts  $N_{i-1}(X_i)$  in Equation (5), their values are usually much smaller than  $i \cdot Nd + Ns$ . To accumulate the counts  $N_{i-1}(X_i)$  and  $i \cdot Nd + Ns$  for performing arithmetic coding in an easier and faster way, a weighted version instead of the original  $N_{i-1}(X_i)$  needs to be computed. The weighting is made by keeping the value of  $i \cdot Nd + Ns$  as it is and as the reference. Then a weighted version  $N'_{i-1}(X_i)$  of  $N_{i-1}(X_i)$  is derived according to Equation (9) as

$$N'_{i-1}(X_i) = \frac{(1 - Pe_1)Pe_2}{Pe_1} \cdot (i \cdot Nd + Ns) \quad (11)$$

Similarly, a weighted version of the SOMM count  $N_{i-1}(X_{i-1}, X_i)$  is derived according to Equation (9) as

$$N'_{i-1}(X_{i-1}, X_i) = \frac{(1 - Pe_1)(1 - Pe_2)}{Pe_1} \cdot (i \cdot Nd + Ns) \quad (12)$$

If  $N'_{i-1}(X_i)$  plus  $N'_{i-1}(X_{i-1}, X_i)$  is greater than  $2^{30}$  minus  $i \cdot Nd + Ns$ , the values of  $N'_{i-1}(X_i)$  and  $N'_{i-1}(X_{i-1}, X_i)$  are decreased by multiplying the same factor  $(2^{30} - i \cdot Nd - Ns - 1)/(N'_{i-1}(X_i) + N'_{i-1}(X_{i-1}, X_i))$ . From practical experiments, we find that the probability that the weighted count values,  $N'_{i-1}(X_i)$  and  $N'_{i-1}(X_{i-1}, X_i)$ , need to be decreased is very small, on average 3.7% for English text and 2.3% for Chinese text.

Let  $X_{i-1} = s$ ,  $X_i = u$ ,  $X_{i+1} = v$ , and the accumulated counts at time  $i$  for the token value  $v$  in the three Markov models are  $A_0(i, v)$ ,  $A_1(i, u, v)$ ,  $A_2(i, s, u, v)$  respectively. That is,

$$A_0(i, v) = \sum_{z=1}^v (N_i(z) \cdot Nd + 1), \quad A_1(i, u, v) = \sum_{z=1}^v N_i(u, z), \quad A_2(i, s, u, v) = \sum_{z=1}^v N_i(s, u, z) \quad (13)$$

Then, the cumulated probability interval,  $[g_{i+1}, h_{i+1})$ , assigned to the token  $X_{i+1}$  for arithmetic coding is computed as

$$g_{i+1} = \frac{A_b(i, s, u, v - 1)}{(i \cdot Nd + Ns) + N'_{i-1}(u) + N'_{i-1}(s, u)}, \quad h_{i+1} = \frac{A_b(i, s, u, v)}{(i \cdot Nd + Ns) + N'_{i-1}(u) + N'_{i-1}(s, u)} \quad (14)$$

where  $A_b(i, s, u, v)$  is defined as

$$A_b(i, s, u, v) = A_0(i, v) + \frac{N'_{i-1}(u)}{N_{i-1}(u)} A_1(i, u, v) + \frac{N'_{i-1}(s, u)}{N_{i-1}(s, u)} A_2(i, s, u, v) \quad (15)$$

Here, the denominator in Equation (14) is the sum of the three weighted total-counts for the three Markov models.  $A_b(i, s, u, v)$  acts as the blended and cumulated count at time  $i$  for the token value  $v$ , as if the three Markov models are put together and treated as a single model. In a practical implementation, the summation operations in Equation (13) are not directly performed. These cumulated counts can be

computed more efficiently with the large-alphabet accumulation-supported binary search tree structure as mentioned in Section 3. Also, the summations in Equations (7) and (8) need not be computed directly because their values can be maintained in program variables and looked up when needed.

In this paper, the program code for arithmetic coding is not written from scratch, but the code developed for a small alphabet condition by Bell *et al.* [3] has been used as the base. We then check the constant values and declared variables, and change their data types if necessary to provide a 32-bit count-value range and 64-bit-wide precision in multiplication temporarily. In addition, we add program segments for estimating and blending occurrence-probabilities based on three Markov models, and count-variable updating. In the encoding phase of arithmetic coding, the token value of  $X_{i+1}$  is given and we know how to compute its corresponding cumulated probability interval,  $[g_{i+1}, h_{i+1})$ , with the formula listed above. However, in the decoding phase, a tag value is given and we need to find a token value whose cumulated probability interval can contain the given tag value. This is a more difficult and time-consuming task. In this paper, we follow the accumulation-supported binary search tree (token values are treated as the keys) built for ZOMM to check if the token value in a traversed tree node is the desired one. Compute  $[g_{i+1}, h_{i+1})$  according to Equation (14) for the encountered token value first, and check whether the tag value is less than  $g_{i+1}$ , greater than  $h_{i+1}$  or between the two numbers. The left sub-tree is traversed next if the tag value is less than  $g_{i+1}$ , and the right sub-tree is selected to traverse if the tag value is greater than or equal to  $h_{i+1}$ .

## 5. EXPERIMENTS AND RESULTS

To study the compression ability of our scheme, we have programmed it into C language programs. Besides normal versions for Chinese and English text compression, two downgraded versions are also prepared, one using a small alphabet for parsing and the other blending only the zero- and first-order Markov models. For each version of the compression program, we have always programmed its corresponding decompression program in order to verify that each text file is correctly compressed and decompressed. Our programs are developed and tested on a Linux platform running a Pentium III 850 MHz CPU. On this platform, our programs are tested to have average speeds, 289 Kbytes  $s^{-1}$  in compression and 148 Kbytes  $s^{-1}$  in decompression. In run time, the main memory used by our programs may reach 13.8 Mbytes maximally.

For testing our compression programs, we have collected several typical text files from the Internet. The Chinese text files included are (a) 'Hong Lou Mong' (紅樓夢, a classic novel), (b) 'Siau Au Ziang Hu' (笑傲江湖, a famous novel of swordsmen), (c) 'Zyie Dai Suang Ziau' (絕代雙驕, another novel of swordsmen), (d) 'CyiongIau' (瓊瑤, three short love stories merged), and (e) 'Net News' (a sequence of short news reports collected from the Internet). For English text compression, the files included are (a) 'The Lord of the Rings', parts 1, 2, and 3, (b) 'Harry Potter', parts 1, 2, 3, and 4, (c) the 'Bible', (d) 'Little Women', and (e) 'Anne of Avonlea', 'Anne of Green Gables', and 'Anne of the Island'.

### 5.1. Alphabet size and max prediction order

Two alphabet sizes are studied here, large alphabet and small alphabet. 'Large alphabet' means that two consecutive bytes may be parsed out as a token or not according to the parsing rules adopted for Chinese and English. 'Small alphabet' means that each byte itself is parsed out as a token. In this

Table I. Compressed sizes and rates in different combinations of alphabet size and max prediction order for Chinese text files.

Text files	Original size (bytes)	Large alphabet Max order 2		Large alphabet Max order 1		Small alphabet Max order 2		Small alphabet Max order 1	
		Compress. size	Rate (%)	Compress. size	Rate (%)	Compress. size	Rate (%)	Compress. size	Rate (%)
HongLouMong	1 449 821	579 458	40.0	607 798	41.9	662 748	45.7	791 485	54.6
SiauAuZiangHu	2 018 872	745 700	36.9	814 858	40.4	905 854	44.9	1 134 559	56.2
ZyieDaiSuangZ	1 679 292	551 956	32.9	607 704	36.2	672 942	40.1	875 886	52.2
CyiongIau	1 458 592	435 413	29.9	525 076	36.0	589 881	40.4	771 099	52.9
Net News	3 422 576	1 082 753	31.6	1 281 379	37.4	1 393 247	40.7	1 902 049	55.6
Average	10 029 153	3 395 280	33.9	3 836 815	38.3	4 224 672	42.1	5 475 078	54.6

subsection, ‘large alphabet’ also indicates that the synchronous-parsing method is used for English text files. As to max prediction order, max order 2 and max order 1 are studied respectively. Max order 2 means that three Markov models of zero, first, and second order are blended. And max order 1 means that only the zero- and first-order Markov models are blended. In this paper, compression rate is computed as compressed file size divided by original file size.

For Chinese text compression, the compressed file sizes in different combinations of alphabet size and max prediction order are as listed in Table I. Comparing the rates listed within large-alphabet and small-alphabet columns, we find that the average rate improvements are  $42.1 - 33.9 = 8.2\%$  under max order 2, and  $54.6 - 38.3 = 16.3\%$  under max order 1. Therefore, a large-alphabet-oriented approach can get much better compression rates for Chinese text when max prediction order is fixed. Also, comparing the rates within max order 2 and 1, we find that the rate improvement,  $38.3 - 33.9 = 4.4\%$ , from including the order 2 Markov model is significant under the large-alphabet condition, while the improvement  $54.6 - 42.1 = 12.5\%$  is evident under the small-alphabet condition.

For English text compression, the compressed file sizes in different combinations of alphabet size and max prediction order are as listed in Table II. Comparing the rates listed within large-alphabet and small-alphabet columns, we see the average rate improvements are  $32.9 - 23.3 = 9.6\%$  under max order 2 and  $42.2 - 29.2 = 13.0\%$  under max order 1. Therefore, a large-alphabet-oriented approach can indeed get much better compression rates for English text files when max prediction order is fixed. Also, the rate improvement,  $29.2 - 23.3 = 5.9\%$ , from including the second order Markov model is significant under the large-alphabet condition while the improvement  $42.2 - 32.9 = 9.3\%$  is evident under the small-alphabet condition.

In Section 2, two parsing methods are discussed, i.e. uniform parsing and synchronous parsing. Different parsing methods in a large-alphabet-oriented compression scheme may result in a significant difference in compression rates. To investigate this, we program the two parsing methods and conduct experiments for English text file compression. The results, compressed file sizes and rates from uniform parsing are listed in the last column of Table III while the other column is duplicated from Table II.

Table II. Compressed sizes and rates in different combinations of alphabet size and max prediction order for English text files.

Text files	Original size (bytes)	Large alphabet Max order 2		Large alphabet Max order 1		Small alphabet Max order 2		Small alphabet Max order 1	
		Compress. size	Rate (%)	Compress. size	Rate (%)	Compress. size	Rate (%)	Compress. size	Rate (%)
Lord_Ring	2 865 122	691 717	24.1	848 540	29.6	949 446	33.1	1 207,223	42.1
Harry_Potter	2 677 540	641 291	24.0	807 680	30.2	911 654	34.0	1 159 171	43.3
Bible	4 477 958	926 289	20.7	1 207 105	27.0	1 392 054	31.1	1 828 737	40.8
Little_Women	1 039 390	270 516	26.0	321 656	30.9	353 409	34.0	443 159	42.6
Anne_	1 969 393	499 694	25.4	616 146	31.3	678 883	34.5	862 536	43.8
Average	13 029 403	3 029 507	23.3	3 801 127	29.2	4 285 446	32.9	5 500 826	42.2

Table III. Compressed file sizes and rates in synchronous and uniform parsing.

Text files	Original size (bytes)	Synchronous parsing Max order 2		Uniform parsing Max order 2	
		Compress. size	Rate (%)	Compress. size	Rate (%)
Lord_Ring	2 865 122	691 717	24.1	728 869	25.4
Harry_Potter	2 677 540	641 291	24.0	675 657	25.2
Bible	4 477 958	926 289	20.7	967 608	21.6
Little _ Women	1 039 390	270 516	26.0	288 766	27.8
Anne_	1 969 393	499 694	25.4	529 949	26.9
Average	13 029 403	3 029 507	23.3	3 190 849	24.5

Comparing the rates within the two columns, we find the difference in average rate is only 1.2%. Therefore, parsing method is not as important as alphabet size.

## 5.2. Comparison with other compression packages

From Tables I and II, we know the compression rates of our scheme are 33.9% on average for Chinese text and 23.3% on average for English text. To check whether the rates obtained from our large-alphabet-oriented scheme are good or bad we select three popular compression packages and a word-based large-alphabet compression package for comparison. One of the famous packages is PPMd version I, which also adopts Markov modeling and arithmetic coding, but uses a small alphabet and a higher prediction order (4 is the default) [6]. Another package is BZIP2 version 1.0.1, which is based on the Burrows–Wheeler transform (transform from a source character sequence to another

Table IV. Compressed file sizes and rates from different packages for Chinese text files.

Text files	Original size (bytes)	LAMA Compress. size	PPMd Compress. size	BZIP2 Compress. size	GZIP Compress. size	WORD-0 Compress. size
HongLouMong	1 449 821	579 458	590 767	662 841	810 372	962 217
SiauAuZiangHu	2 018 872	745 700	755 852	875 294	1 102 795	1 292 593
ZyieDaiSuangZ	1 679 292	551 956	554 160	631 636	807 787	1 013 209
CyiongIau	1 458 592	435 413	439 046	518 052	750 469	777 533
Net News	3 422 576	1 082 753	1 073 340	1 227 258	1 624 038	2 133 195
Average	10 029 153 100%	3 395 280 33.9%	3 413 165 34.0%	3 915 081 39.0%	5 095 461 50.8%	6 178 747 61.6%

entropy-reduced character sequence) [5]. The third package is GZIP, which is based on LZ77 type of dictionary coding [3,4]. The fourth package is termed WORD-0. It uses a word-based large alphabet and a zero-order Markov model to guide an arithmetic coder [12]. Note that this package supports no higher prediction order than zero. This package is selected because we can find no other package that uses a word-based alphabet and can be tested. Our package is referred to as LAMA, which stands for 'large-alphabet Markov modeling and arithmetic coding'. Although many schemes for Chinese text compression have been proposed and published [13], their executable programs, however, cannot be obtained. Therefore, we cannot compare our scheme with those schemes. The executable programs for our scheme can be retrieved from our web site [14]. 'lamenca.bin' is used for compressing and 'lamdeca.bin' for decompressing.

The same Chinese text files as shown in Table I are used here to test the different packages. The compression rates obtained are listed in Table IV. According to the average rates in Table IV, our scheme, LAMA, is comparable with PPMd, 5% better than BZIP2, 16% better than GZIP, and 27% better than WORD-0. The word-based package, WORD-0, is not good for Chinese text compression because it does not consider particular characteristics of Chinese text, e.g., adjacent words are not delimited by a blank or graphical character. As for English text compression, the same text files as listed in Table II are used. The compression rates obtained are as shown in Table V. According to the rates in Table V, our scheme has a compression rate slightly worse than PPMd, but 1.4% better than BZIP2, 11.2% better than GZIP, and 4.2% better than WORD-0. Therefore, a word-based compression scheme that uses a max prediction order of zero cannot compete with the schemes LAMA, PPMd, and BZIP2 for English and, especially, Chinese text compression.

To find out whether our large-alphabet-oriented scheme will obtain better compression rates only for large text files, we pick the file 'Hong Lou Mong' as an example of Chinese text and the file 'Little Women' as an example of English text. Then compression rates are measured at several points within the two text files. The rates measured in executing the packages LAMA, PPMd, and BZIP2 are collected. Then the rates are used to plot Figure 2. The three curves at the upper side are obtained by compressing the Chinese text file. Apparently the curve of our package is always lower than that of other two, no matter where the measuring point is placed, and is always a gap away from the curve of BZIP2. Therefore, our scheme can obtain a good compression rate even for a small Chinese text file.

Table V. Compressed files size and rates from different packages for English text files.

Text files	Original size (bytes)	LAMA Compress. size	PPMd Compress. size	BZIP2 Compress. size	GZIP Compress. size	WORD-0 Compress. size
Lord_Ring	2 865 122	691 717	692 944	752 723	1 038 272	788 768
Harry_Potter	2 677 540	641 291	641 007	690 588	980 314	739 636
Bible	4 477 958	926 289	891 519	929 712	1 319 995	1 179 560
Little_Women	1 039 390	270 516	267 522	291 641	402 226	301 996
Anne_	1 969 393	499 694	496 864	547 121	754 637	568 054
Average	13 029 403	3 029 507	2 989 856	3 211 785	4 495 444	3 578 014
	100%	23.3%	22.9%	24.7%	34.5%	27.5%

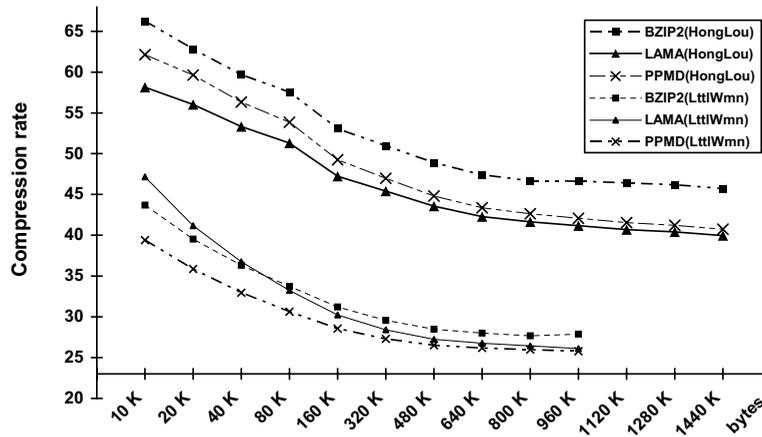


Figure 2. Compression rates plotted against measuring points in two text files.

At the lower side of Figure 2, the three curves are obtained by compressing the English text file. The curve of our scheme will cross the curve of BZIP2. The crossing point is located between the points 40 K and 80 K, i.e. our scheme would be better than BZIP2 after the crossing point (about 60 K). Therefore, our scheme is not good for a small English text file. However, our scheme's curve will go down quickly and approach the curve of PPMd so long as the text file's size is large enough.

## 6. CONCLUSION

The large-alphabet-oriented scheme proposed here not only obtains good compression rates for Chinese text, but can also obtain good rates for English text. Fed with typical text files to compress, our

scheme obtains, on average, rates of 33.9% for Chinese text and 23.3% for English text. These rates are comparable with those obtained by PPMd, significantly better than BZIP2 for Chinese text and slightly better than BZIP2 for English text, much better than GZIP for both Chinese and English text, and much better than WORD-0 for Chinese text and significantly better than WORD-0 for English text. Therefore, our large-alphabet-oriented scheme is as powerful as a small-alphabet-oriented scheme for compressing text files composed in a small-alphabet language.

A type of large alphabet other than a word alphabet is studied in this paper. Two parsing methods are designed and experimented to parse out English text into tokens with more regular lengths and a pre-known alphabet size. The synchronous-parsing method is found to be better than the uniform-parsing method, but the difference in compression rate is small. When the max prediction order is fixed, it is found that the compression rates obtained from using a large alphabet are always much better than those obtained from using a small alphabet. When max prediction order is fixed to 1, the rate improvements are as high as 16.3% for Chinese text and 13.0% for English text. When fixed to 2, the rate improvements are 8.2% for Chinese text and 9.6% for English text.

According to the experimental results, in comparison with PPMd, it seems that the two factors of alphabet size and max prediction order are changeable. This observation is made because the compression rates obtained from our scheme and from PPMd are almost equal, and our scheme uses a large alphabet and a max prediction order of 2, while PPMd uses a small alphabet and a default max prediction order of 4. Since our scheme uses a lower prediction order, the approach, blending several model-predicted probabilities, can be more easily implemented in practice. Otherwise, a suboptimal approach of escaping different model-predicted probabilities should be adopted.

## REFERENCES

1. Gu H-Y. Large-alphabet Chinese text compression using adaptive Markov model and arithmetic coder. *Journal of Computer Processing of Chinese and Oriental Languages* 1995; **9**(2):111–124.
2. Hamming RW. *Coding and Information Theory*. Prentice-Hall: Englewood Cliffs, NJ, 1980.
3. Bell TC, Cleary JG, Witten IH. *Text Compression*. Prentice-Hall: Englewood Cliffs, NJ, 1990.
4. Sayood K. *Introduction to Data Compression* (2nd edn). Morgan Kaufmann: San Francisco, CA, 2000.
5. Seward J. The bzip2 and libbzip2 home page. <http://sources.redhat.com/bzip2/>.
6. Shkarin D. PPM: One step to practicality. *Proceedings of the IEEE Data Compression Conference*, Snowbird, UT, April 2002. IEEE Computer Society Press, 2002; 202–211. Available at: <http://compression.graphicon.ru/ds/>.
7. Salomon D. *Data Compression*. Springer: New York, 1998.
8. Horspool RN, Cormack GV. Constructing word-based text compression algorithms. *Proceedings of the IEEE Data Compression Conference*, Los Alamitos, CA, March 1992. IEEE Computer Society Press, 1992; 67–71.
9. Cleary JG, Witten IH. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications* 1984; **32**(4):396–402.
10. Nadas A. On Turing's formula for word probabilities. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 1985; **33**(6):1414–1416.
11. Witten IH, Bell TC. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory* 1991; **37**(4):1085–1094.
12. Moffat A. The arithmetic coding page. [http://www.cs.mu.oz.au/~alistair/arith\\_coder/](http://www.cs.mu.oz.au/~alistair/arith_coder/).
13. Vines P, Zobel J. Compression techniques for Chinese text. *Software Practice and Experience* 1998; **28**(12):1299–1314.
14. Gu H-Y. Large-alphabet oriented text compression. <http://mail.ntust.edu.tw/~guhy/>.