A Compression-based Algorithm for Chinese Word Segmentation

W. J. Teahan*
The Robert Gordon University

Yingying Wen†
University of Waikato

Rodger McNab†
University of Waikato

Ian H. Witten†
University of Waikato

Chinese is written without using spaces or other word delimiters. Although a text may be thought of as a corresponding sequence of words, there is considerable ambiguity in the placement of boundaries. Interpreting a text as a sequence of words is beneficial for some information retrieval and storage tasks: for example, full-text search, word-based compression, and keyphrase extraction. We describe a scheme that infers appropriate positions for word boundaries using an adaptive language model that is standard in text compression. It is trained on a corpus of presegmented text, and when applied to new text, interpolates word boundaries so as to maximize the compression obtained. This simple and general method performs well with respect to specialized schemes for Chinese language segmentation.

1. Introduction

Languages such as Chinese and Japanese are written without using any spaces or other word delimiters (except for punctuation marks)—indeed, the Western notion of a word boundary is literally alien (Wu 1998). Nevertheless, words are present in these languages, and Chinese words often comprise several characters, typically two, three, or four—five-character words also exist, but they are rare. Many characters can stand alone as words in themselves, while on other occasions the same character is the first or second character of a two-character word, and on still others it participates as a component of a three- or four-character word. This phenomenon causes obvious ambiguities in word segmentation.

Readers unfamiliar with Chinese can gain an appreciation of the problem of multiple interpretations from Figure 1, which shows two alternative interpretations of the same Chinese character sequence. The text is a joke that relies on the ambiguity of phrasing. Once upon a time, the story goes, a man set out on a long journey. Before he could return home the rainy season began, and he had to take shelter at a friend's house. But he overstayed his welcome, and one day his friend wrote him a note: the first line in Figure 1. The intended interpretation is shown in the second line, which means "It is raining, the god would like the guest to stay. Although the god wants you to stay, I do not!" On seeing the note, the visitor took the hint and prepared to leave. As a joke he amended the note with the punctuation shown in the third line, which leaves three sentences whose meaning is totally different—"The rainy day, the staying day. Would you like me to stay? Sure!"

* School of Computing and Mathematical Sciences, The Robert Gordon University, Aberdeen, Scotland
† Computer Science, University of Waikato, Hamilton, New Zealand

A sentence in Chinese	下雨天留客天留我不留
Interpretation 1	下雨，天留客。天留，我不留！
Interpretation 2	下雨天，留客天。留我不？留！

Figure 1
A Chinese sentence with ambiguity of phrasing.

A sentence in Chinese	我喜欢新西兰花
Interpretation 1	我 喜欢 新西兰 花
Interpretation 2	我 喜欢 新 西兰花

Figure 2
An example that can be segmented in two different ways.

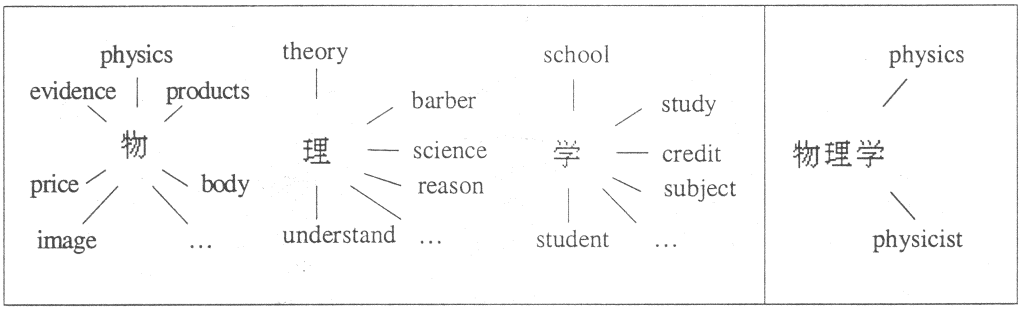

Figure 3
Example of treating each character in a query as a word.

This example relies on ambiguity of phrasing, but the same kind of problem can arise with word segmentation. Figure 2 shows a more prosaic example. For the ordinary sentence of the first line, there are two different interpretations depending on the context of the sentence: "I like New Zealand flowers" and "I like fresh broccoli" respectively.

The fact that machine-readable Chinese text is invariably stored in unsegmented form causes difficulty in applications that use the word as the basic unit. For example, search engines index documents by storing a list of the words they contain, and allow the user to retrieve all documents that contain a specified combination of query terms. This presupposes that the documents are segmented into words. Failure to do so, and treating every character as a word in itself, greatly decreases the precision of retrieval since large numbers of extraneous documents are returned that contain characters, but not words, from the query.

Figure 3 illustrates what happens when each character in a query is treated as a single-character word. The intended query is "physics" or "physicist." The first character returns documents about such things as "evidence," "products," "body," "image," "prices"; while the second returns documents about "theory," "barber," and so on. Thus many documents that are completely irrelevant to the query will be returned, causing the precision of information retrieval to decrease greatly. Similar problems occur in word-based compression, speech recognition, and so on.

It is true that most search engines allow the user to search for multiword phrases by enclosing them in quotation marks, and this facility could be used to search for multicharacter words in Chinese. This, however, runs the risk of retrieving irrelevant documents in which the same characters occur in sequence but with a different intended segmentation. More importantly, it imposes on the user an artificial requirement to perform manual segmentation on each full-text query.

Word segmentation is an important prerequisite for such applications. However, it is a difficult and ill-defined task. According to Sproat et al. (1996) and Wu and Fung (1994), experiments show that only about 75% agreement between native speakers is to be expected on the "correct" segmentation, and the figure reduces as more people become involved.

This paper describes a general scheme for segmenting text by inferring the position of word boundaries, thus supplying a necessary preprocessing step for applications like those mentioned above. Unlike other approaches, which involve a dictionary of legal words and are therefore language-specific, it works by using a corpus of already-segmented text for training and thus can easily be retargeted for any language for which a suitable corpus of segmented material is available. To infer word boundaries, a general adaptive text compression technique is used that predicts upcoming characters on the basis of their preceding context. Spaces are inserted into positions where their presence enables the text to be compressed more effectively. This approach means that we can capitalize on existing research in text compression to create good models for word segmentation. To build a segmenter for a new language, the only resource required is a corpus of segmented text to train the compression model.

The structure of this paper is as follows: The next section reviews previous work on the Chinese segmentation problem. Then we explain the operation of the adaptive text compression technique that will be used to predict word boundaries. Next we show how space insertion can be viewed as a problem of hidden Markov modeling, and how higher-order models, such as the ones used in text compression, can be employed in this way. The following section describes several experiments designed to evaluate the success of the new word segmenter. Finally we discuss the application of language segmentation in digital libraries.

Our system for segmenting Chinese text is available on the World Wide Web at http://www.nzdl.org/cgi-bin/congb. It takes GB-encoded input text, which can be cut from a Chinese document and pasted into the input window.[1] Once the segmenter has been invoked, the result is rewritten into the same window.

2. Previous Methods for Segmenting Chinese

The problem of segmenting Chinese text has been studied by researchers for many years; see Wu and Tseng (1993) for a detailed survey. Several different algorithms have been proposed, which, generally speaking, can be classified into dictionary-based and statistical-based methods, although other techniques that involve more linguistic information, such as syntactic and semantic knowledge, have been reported in the natural language processing literature.

Cheng, Young, and Wong (1999) describe a dictionary-based method. Given a dictionary of frequently used Chinese words, an input string is compared with words in the dictionary to find the one that matches the greatest number of characters of the

1 To enable proper viewing, and input, of GB-encoded characters, an appropriate version of Netscape Communicator or Internet Explorer must be used.

input. This is called the **maximum forward match** heuristic. An alternative is to work backwards through the text, resulting in the **maximum backward match** heuristic. It is easy to find situations where these fail. To use an English example, forward matching fails on the input "the red . . . " (it is misinterpreted as "there d . . . "), while backward matching fails on text ending ". . . his car" (it is misinterpreted as ". . . hi scar"). Analogous failures occur with Chinese text.

Dai, Khoo, and Loh (1999) use statistical methods to perform text segmentation. They concentrate on two-character words, because two characters is the most common word length in Chinese. Several different notions of frequency of characters and bigrams are explored: relative frequency, document frequency, weighted document frequency, and local frequency. They also look at both contextual and positional information. Contextual information is found to be the single most important factor that governs the probability that a bigram forms a word; incorporating the weighted document frequency can improve the model significantly. In contrast, the positional frequency is not found to be helpful in determining words.

Ponte and Croft (1996) introduce two models for word segmentation: word-based and bigram models. Both utilize probabilistic automata. In the word-based method, a suffix tree of words in the lexicon is used to initialize the model. Each node is associated with a probability, which is estimated by segmenting training text using the longest match strategy. This makes the segmenter easy to transplant to new languages. The bigram model uses the lexicon to initialize probability estimates for each bigram, and the probability with which each bigram occurs, and uses the Baum-Welch algorithm (Rabiner 1989) to update the probabilities as the training text is processed.

Hockenmaier and Brew (1998) present an algorithm, based on Palmer's (1997) experiments, that applies a symbolic machine learning technique—transformation-based error-driven learning (Brill 1995)—to the problem of Chinese word segmentation. Using a set of rule templates and four distinct initial-state annotators, Palmer concludes that the learning technique works well. Hockenmaier and Brew investigate how performance is influenced by different rule templates and corpus size. They use three rule templates: simple bigram rules, trigram rules, and more elaborate rules. Their experiments indicate that training data size has the most significant influence on performance. Good performance can be acquired using simple rules only if the training corpus is large enough.

Lee, Ng, and Lu (1999) have recently introduced a new segmentation method for a Chinese spell-checking application. Using a dictionary with single-character word occurrence frequencies, this scheme first divides text into sentences, then into phrases, and finally into words using a small number of word combinations that are conditioned on a heuristic to avoid delay during spell-checking. When compared with forward maximum matching, the new method resolves more than 10% more ambiguities, but enjoys no obvious speed advantage.

The way in which Chinese characters are used in names differs greatly from the way they are used in ordinary text, and some researchers, notably Sproat et al. (1996), have established special-purpose recognizers for Chinese names (and translated foreign names), designed to improve the accuracy of automatic segmenters by treating names specially.[2] Chinese names always take the form *family name* followed by *given name*. Whereas family names are limited to a small group of characters, given names can consist of any characters. They normally comprise one or two characters, but

2 In English there are significant differences between the frequency distribution of letters in names and in words—for example, compare the size of the *T* section of a telephone directory with the size of the *T* section of a dictionary—but such differences are far more pronounced in Chinese.

three-character names have arisen in recent years to ensure uniqueness when the family name is popular—such as Smith or Jones in English. Sproat et al. (1996) implement special recognizers not only for Chinese names and transliterated foreign names, but for components of morphologically obtained words as well. The approach we present is not specially tailored for name recognition, but because it is fully adaptive it is likely that it would yield good performance on names if lists of names were provided as supplementary training text. This has not yet been tested.

3. Language Modeling using PPM

Statistical language models are well developed in the field of text compression. Compression methods are usually divided into symbolwise and dictionary schemes (Bell, Cleary, and Witten, 1990). Symbolwise methods, which generally make use of adaptively generated statistics, give excellent compression—in fact, they include the best known methods. Although dictionary methods such as the Ziv-Lempel schemes perform less well, they are used in practical compression utilities like Unix *compress* and *gzip* because they are fast.

In our work we use the prediction by partial matching (PPM) symbolwise compression scheme (Cleary and Witten 1984), which has become a benchmark in the compression community. It generates "predictions" for each input symbol in turn. Each prediction takes the form of a probability distribution that is provided to an encoder. The encoder is usually an arithmetic coder; the details of coding are of no relevance to this paper.

PPM is an *n*-gram approach that uses finite-context models of characters, where the previous few (say three) characters predict the upcoming one. The conditional probability distribution of characters, conditioned on the preceding few characters, is maintained and updated as each character of input is processed. This distribution, along with the actual value of the preceding few characters, is used to predict each upcoming symbol. Exactly the same distributions are maintained by the decoder, which updates the appropriate distribution as each character is received. This is what we call **adaptive modeling**: both encoder and decoder maintain the same models—not by communicating the models directly, but by updating them in precisely the same way.

Rather than using a fixed context length (three was suggested above), the PPM method chooses a maximum context length and maintains statistics for this and all shorter contexts. The maximum is five in most of the experiments below, and statistics are maintained for models of order 5, 4, 3, 2, 1, and 0. These are not stored separately; they are all kept in a single trie structure.

PPM incorporates a simple and highly effective method to combine the predictions of the models of different order—often called the problem of "backoff." To encode the next symbol, it starts with the maximum-order model (order 5). If that model contains a prediction for the upcoming character, the character is transmitted according to the order 5 distribution. Otherwise, both encoder and decoder **escape** down to order 4. There are two possible situations. If the order 5 **context**—that is, the preceding five-character sequence—has not been encountered before, then escape to order 4 is inevitable, and both encoder and decoder can deduce that fact without requiring any communication. If not, that is, if the preceding five characters have been encountered in sequence before but not followed by the upcoming character, then only the encoder knows that an escape is necessary. In this case, therefore, it must signal this fact to the decoder by transmitting an **escape event**—and space must be reserved for this event in every probability distribution that the encoder and decoder maintain.

Table 1
PPM model after processing the string *tobeornottobe*; c = count, p = prediction probability.

Order 2

Prediction			c	p
be	→	o	1	1/2
	→	*esc*	1	1/2
eo	→	r	1	1/2
	→	*esc*	1	1/2
no	→	t	1	1/2
	→	*esc*	1	1/2
ob	→	e	2	3/4
	→	*esc*	1	1/4
or	→	n	1	1/2
	→	*esc*	1	1/2
ot	→	t	1	1/2
	→	*esc*	1	1/2
rn	→	o	1	1/2
	→	*esc*	1	1/2
to	→	b	2	3/4
	→	*esc*	1	1/4
tt	→	o	1	1/2
	→	*esc*	1	1/2

Order 1

Prediction			c	p
b	→	e	2	3/4
	→	*esc*	1	1/4
e	→	o	1	1/2
	→	*esc*	1	1/2
n	→	o	1	1/2
	→	*esc*	1	1/2
o	→	b	2	3/8
	→	r	1	1/8
	→	t	1	1/8
	→	*esc*	3	3/8
r	→	n	1	1/2
	→	*esc*	1	1/2
t	→	o	2	1/2
	→	t	1	1/6
	→	*esc*	2	1/3

Order 0

Prediction			c	p
	→	b	2	3/26
	→	e	2	3/26
	→	n	1	1/26
	→	o	4	7/26
	→	r	1	1/26
	→	t	3	5/26
	→	*esc*	6	3/13

Order −1

Prediction			c	p
	→	A	1	$1/\|A\|$

Once any necessary escape event has been transmitted and received, both encoder and decoder agree that the upcoming character will be coded by the order 4 model. Of course, this may not be possible either, and further escapes may take place. Ultimately, the order 0 model may be reached; in this case the character can be transmitted if it is one that has occurred before. Otherwise, there is one further escape (to an order −1 model), and the standard ASCII representation of the character is sent.

The only remaining question is how to calculate the probabilities from the counts—a simple matter once we have resolved how much space to allocate for the escape probability. There has been much discussion of this question, and several different methods have been proposed. Our experiments calculate the escape probability in a particular context as

$$\frac{\frac{1}{2}d}{n}$$

where n is the number of times that context has appeared and d is the number of different symbols that have directly followed it (Howard 1993). The probability of a character that has occurred c times in that context is

$$\frac{c - \frac{1}{2}}{n}.$$

Since there are d such characters, and their counts sum to n, it is easy to confirm that the probabilities in the distribution (including the escape probability) sum to 1.

To illustrate the PPM modeling technique, Table 1 shows the model after the string *tobeornottobe* has been processed. In this illustration the maximum model order is 2 (not 5 as stated above), and each prediction has a count c and a prediction probability p. The probability is determined from the counts associated with the prediction using the formula that we discuss above. |A| is the size of the alphabet, and it is this that determines the probability for each unseen character.

The model in Table 1 is used as follows: Suppose the character following *tobeornottobe* is *o*. Since the order 2 context is *be*, and the upcoming symbol has already been seen once in this context, the order 2 model is used for encoding in this case, and the encoding probability is 1/2. Thus the symbol *o* would be encoded in 1 bit. If the next character, instead of *o*, were *t*, this has not been seen in the current order 2 context (which is still *be*). Consequently an order 2 escape event is coded (probability 1/2, again in the *be* context), and the context is truncated to *e*. Checking the order 1 model, the upcoming character *t* has not been seen in this context, so an order 1 escape event is coded (probability 1/2 in the *e* context) and the context is truncated to the null context, corresponding to the order 0 model. The character *t* is finally encoded in this model, with probability 5/26. Thus three encodings occur for this one character, with probabilities 1/2, 1/2, and 5/26 respectively, which together amount to just over 5 bits of information. If the upcoming character had been *x* instead of *t*, a final level of escape, this time to order 0, would have occurred (probability 3/13), and the *x* would be encoded with a probability of 1/256 (assuming that the alphabet has 256 characters) for a total of just over 10 bits.

It is clear from Table 1 that, in the context *tobeornottobe*, if the next character is *o* it will be encoded by the order 2 model. Hence if an escape occurs down to order 1, the next character cannot be *o*. This makes it unnecessary to reserve probability space for the occurrence of *o* in the order 1 (or order 0 or order -1) models. This idea, which is called **exclusion**, can be exploited to improve compression. A character that occurs at one level is excluded from all lower-order predictions, allowing a greater share of the probability space to be allocated to the other characters in these lower-order models (Bell, Cleary, and Witten 1990). For example, if the character *b* were to follow *tobeornottobe* it would be encoded with probabilities $(1/2, 1/2, 3/26)$, without exclusion, leading to a coding requirement of 5.1 bits. However, if exclusion was exploited, both encoder and decoder will recognize that escape from order 1 to order 0 is inevitable because the order 1 model adds no characters that were not already predicted by the order 2 model. Thus the coding probabilities will be $(1/2, 1, 3/18)$ with exclusion, reducing the total code space for *b* to 3.6 bits. An important special case of the exclusion policy occurs at the lowest-level model: for example, the *x* at the end of the previous paragraph would finally be encoded with a probability of 1/250 rather than 1/256 because characters that have already occurred can never be predicted in the order -1 context.

One slight further improvement to PPM is incorporated in the experiments: deterministic scaling (Teahan 1998). Although it probably has negligible effect on our overall results, we record it here for completeness. Experiments show that in deterministic contexts, for which $d = 1$, the probability that the single character that has occurred before reappears is greater than the $1 - 1/(2n)$ implied by the above estimator. Consequently, in this case the probability is increased in an ad hoc manner to $1 - 1/(6n)$.

4. Using a Hidden Markov Model to Insert Spaces

Inserting spaces into text can be viewed as a hidden Markov modeling problem. Being entirely adaptive, the method works regardless of what language it is used with. For pedagogical purposes, we will explain it with English text.

Between every pair of characters lies a potential space. Figure 4(a) illustrates the model for the fragment *tobeornottobe*. It contains one node for each letter in the text and one for each possible intercharacter space (represented as dots • in the figure). Any given assignment of word boundaries to this text fragment will correspond to

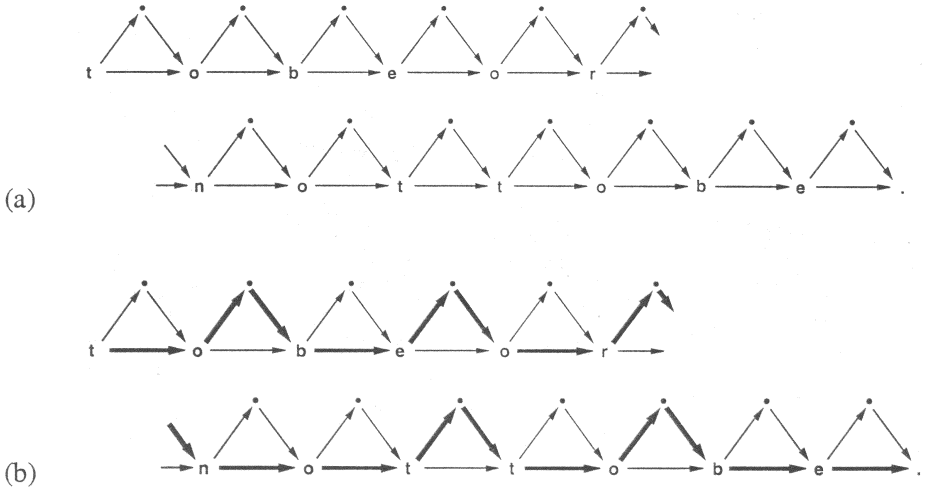

Figure 4
Hidden Markov Model for Space Insertion.

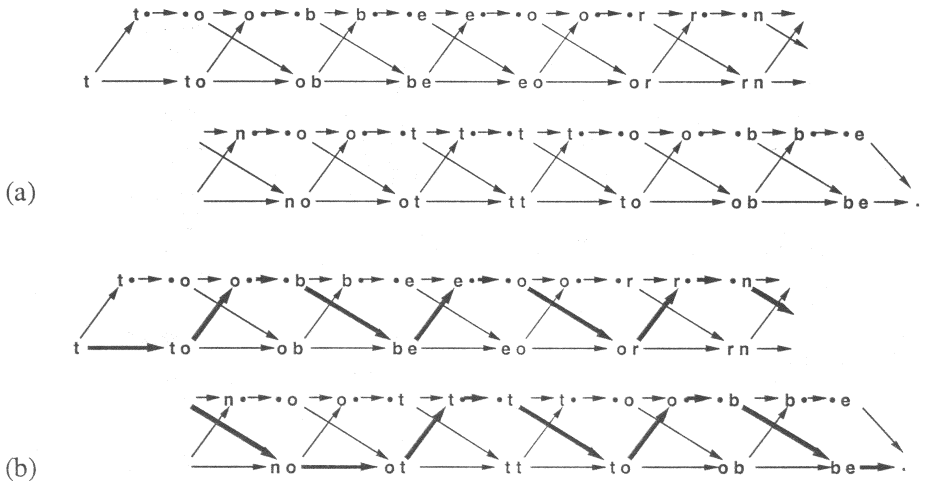

Figure 5
Hidden Markov model for space insertion using an order 1 model.

a path through the model from beginning (at the left) to end (at the right). Of all possible paths, we seek the one that gives the best compression according to the PPM text compression method, suitably primed with English text. This path is the correct path, corresponding to the text *to be or not to be*, shown in bold in Figure 4(b).

4.1 Markov Modeling with Context

Figure 4 can easily be converted into a Markov model for a given order of PPM. Suppose we use order 1: then we rewrite Figure 4(a) so that the states are bigrams, as shown in Figure 5(a). The interpretation of each state is that it corresponds to the *last* character of the string that labels the state. The very first state, labeled *t*, has no prior context—in PPM terms, that character will be transmitted by escaping down to order 0 (or −1). Again, the bold arrows in Figure 5(b) shows the path corresponding to the string with spaces inserted correctly.

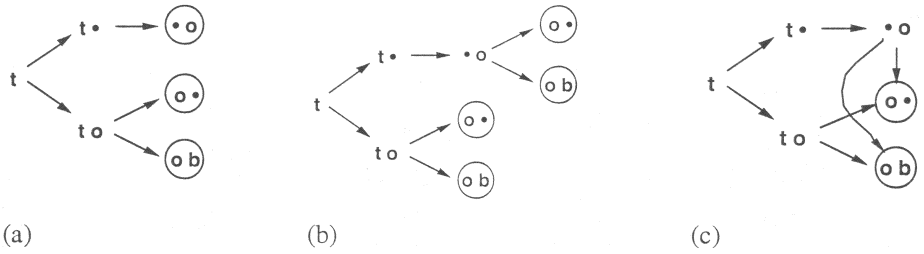

Figure 6
Growing a tree for order 1 modeling of *tobeornottobe*.

Similar models could be written for higher-order versions of PPM. For example, with an order 3 model, states would be labeled by strings of length four (except for the first few states, where the context would be truncated because they occur at the beginning of the string). And each state would have variants corresponding to all different ways of inserting space into the four-character string. For example, the states corresponding to the sixth character of *tobeornottobe* would include *beor* and *e•or*, as well as *•eor*, *eo•r* and *•o•r*. It is not hard to see that the number of states corresponding to a particular character of the input string increases with model order according to the Fibonnacci series. Figure 5(a) shows two states per symbol for order 1, there are three states per symbol for order 2, five for order 3, eight for order 4, thirteen for order 5, and so on.

4.2 The Space Insertion Algorithm

Given a hidden Markov model like the one in Figure 5(a), where probabilities are supplied for each edge according to an order 1 compression model, the space insertion problem is tantamount to finding the sequence of states through the model, from beginning to end, that maximizes the total probability—or, equivalently, that minimizes the number of bits required to represent the text according to that model. The following Viterbi-style algorithm can be used to solve this problem. Beginning at the initial state, the procedure traces through the model, recording at each state the highest probability of reaching that state from the beginning. Thus the two descendants of the start node, nodes *to* and *t•*, are assigned the probability of *o* and •, conditioned in each case on *t* being the prior character, respectively. As more arcs are traversed, the associated probabilities are multiplied: thus the node *•o* receives the product of the probability of • conditioned on *t* and of *o* conditioned on •. When the node *ob* is reached, it is assigned the *greater* of the probabilities associated with the two incoming transitions, and so on throughout the model. This is the standard dynamic programming technique of storing with each state the result of the best way of reaching that state, and using this result to extend the calculation to the next state. To find the optimal state sequence is simply a matter of recording with each state which incoming transition is associated with the greatest probability, and traversing that path in the reverse direction once the final node is reached.

These models can be generated dynamically by proceeding to predict each character in turn. Figure 6(a) shows the beginning of the tree that results. First, the initial node *t* is expanded into its two children, *t•* and *to*. Then, these are expanded in turn. The first has one child, *•o*, because a space cannot be followed by another space. The second has two, *o•* and *ob*. Figure 6(b) shows the further expansion of the *•o* node. However, the two children that are created already exist in the tree, and so the existing versions of these nodes are used instead, as in Figure 6(c). If this procedure is con-

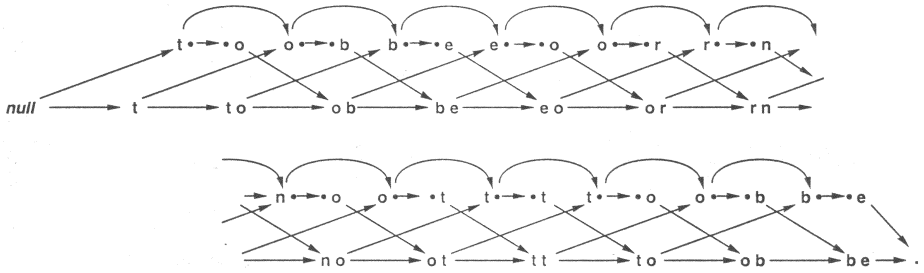

Figure 7
The space insertion procedure as implemented.

tinued, the graph structure of Figure 5(a) will be created. During creation, probability values can be assigned to the nodes, and back pointers inserted to record the best path to each node.

The illustration in Figure 6 is for an order 1 model, but exactly the same procedure applies for higher-order PPM models.

4.3 Implementation of the Space Insertion Algorithm

Our implementation uses a slight variant of the above procedure for finding the optimal place to insert spaces. At each stage, we consider the possibility of adding either the next character, or the next character followed by a space. This generates the structure shown in Figure 7. Starting with the null string, both *t* and *t*• are generated as successor states. From each of these states, either *o* or *o*• can be added, and these yield the next states shown. The procedure continues, growing the trellis structure using an incremental strategy similar to that illustrated in Figure 6, but modified to take into account the new growth strategy of adding either the next character or the next character followed by a space.

The search strategy we use is a variant of the stack algorithm for sequential decoding (Anderson and Mohan 1984). As new nodes are generated, an ordered list is maintained of the best paths generated so far. Only the best path is extended. The metric used to evaluate a path is the number of bits required for the segmentation sequence it represents, when compressed by the PPM model.

It is necessary to delete paths from the list in order to make room for newly generated ones. We remove all paths that were more than *m* nodes shorter than the best path so far, where *m* is the order of the PPM model (5 in our experiments). We reasoned that it is extremely unlikely—at least for natural language sequences—that such a path would ever grow to outperform the current best path, because it already lags behind in code length despite the fact that *m* further letters must be encoded.

5. Experimental Evaluation

Before describing experiments to assess the success of the new word segmentation method, we first discuss measures that are used to evaluate the accuracy of automatic segmentation. We then examine the application of the new segmentation method to English text, and show how it achieves results that significantly outperform the state of the art. Next we describe application to a manually segmented corpus of Chinese text; again, excellent results are achieved. In a further experiment where we apply a model generated from the corpus to a new, independent, test file, performance deteriorates considerably—as one might expect. We then apply the method to a different corpus,

and investigate how well the model transfers from one corpus to another. We end with a discussion of how the results vary with the order of the compression model used to drive the segmenter.

5.1 Measuring the Quality of Segmentation

We use three measures to evaluate the accuracy of automatic segmentation: recall, precision, and error rate. All evaluations use hand-segmentation as the gold standard, which the automatic method strives to attain. To define them, we use the terms

N	Number of words occurring in the hand-segmentation
e	Number of words incorrectly identified by the automatic method
c	Number of words correctly identified by the automatic method
$n = c + e$	Number of words identified by the automatic method

Recall and precision are standard information retrieval measures used to assess the quality of a retrieval system in terms of how many of the relevant documents are retrieved (recall) and how many of the retrieved documents are relevant (precision):

$$\begin{aligned} \text{recall} &= \frac{c}{N}, \\ \text{precision} &= \frac{c}{n}. \end{aligned}$$

The overall error rate can be defined as

$$\text{error rate} = \frac{e}{N}.$$

This in principle can give misleading results—an extreme condition is where the automatic method only identifies a single word, leading to a very small error rate of $1/N$ despite the fact that all words but one are misidentified. However, in all our experiments extreme conditions do not occur because n is always close to N and we find that the error rate is a useful overall indicator of the quality of segmentation. We also used the F-measure to compare our results with others:

$$\text{F-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

If the automatic method produces the same number of words as the hand-segmentation, recall and precision both become equal to one minus the error rate. A perfect segmenter will have an error rate of zero and recall and precision of 100%.

All these measures can be calculated automatically from a machine-segmented text, along with the hand-segmented gold standard. Both texts are identical except for the points where spaces are inserted: thus we record just the start and end positions of each word in both versions. For example, "A BC AED F" in the machine-segmented version is mapped to (1,1) (2,3) (4,6) (7,7), and "A BC A ED F" in the hand-segmented version becomes (1,1) (2,3) (4,4) (5,6) (7,7). The number of correctly and incorrectly segmented words is counted by comparing these two sets of positions, indicated by matched and mismatched pairs, respectively—three correct and two incorrect, in this example.

5.2 Application to English Text

It may be helpful for non-Chinese readers to briefly illustrate the success of the space insertion method by showing its application to English text. The first part of Table 2

Table 2
Segmenting words in English text.

Original text	the unit of New York-based Loews Corp that makes Kent cigarettes stopped using crocidolite in its Micronite cigarette filters in 1956.
Without spaces	TheunitofNewYork-basedLoewsCorpthatmakesKentcigarettesstoppedusing-crocidoliteinitsMicronitecigarettefiltersin1956.
PPM method	the unit of New York-based LoewsCorp that makes Kent cigarettes stopped using croc idolite in its Micronite cigarette filters in 1956.
USeg method	the unit of New York-based Loews Corp that makes Kent cigarettes stopped using c roc id o lite inits Micron it e cigarette filters in 1956.

shows the original text, with spaces in the proper places. The second shows the text with spaces removed, used as input to the segmentation procedure. The third shows the output of the PPM-based method described above, while the fourth shows, for comparison, the output of a word-based method for predicting the position of spaces, USeg (Ponte and Croft 1996).

For this experiment (first reported by Teahan et al. [1998]), PPM was trained on the million-word Brown corpus (Kucera and Francis 1967). USeg was trained on a far larger corpus containing 1 Gb of data from the Tipster collection (Broglio, Callan, and Croft 1994). Both were tested on the same 500 Kb extract from the *Wall Street Journal*. The recall and precision for PPM were both 99.52%, while the corresponding figures for Useg were 93.56% and 90.03%, respectively. This result is particularly noteworthy because PPM had been trained on only a small fraction of the amount of text needed for the word-based scheme.

The same example was used by Ponte and Croft (1996), and the improved performance of the character-based method is evident even in this small example. Although the word *Micronite* does not occur in the Brown Corpus, it was correctly segmented using PPM. Likewise, *inits* was correctly split into *in* and *its*. PPM makes just two mistakes. First, a space was not inserted into *LoewsCorp* because the single "word" requires only 54.3 bits to encode, whereas *Loews Corp* requires 55.0 bits. Second, an extra space was added to *crocidolite* because that reduced the number of bits required from 58.7 to 55.3.

5.3 Application to a Corpus of Chinese Text

Our first series of experiments used part of Guo Jin's Mandarin Chinese PH corpus, containing one million words of newspaper stories from the Xinhua news agency of PR China written between January 1990 and March 1991. It is represented in the standard GB coding scheme.

Table 3 shows the distribution of word lengths in the corpus. Single-character words are the most frequent; these and bigrams together constitute almost 94% of words. Nearly half the characters appear as constituents of two-character words. Some published figures for Chinese language statistics indicate that this corpus may overrepresent single-character words and underrepresent bigrams—for example, Liu (1987) gives figures for modern Chinese of 5%, 75%, 14%, and 6% for one-character, two-character, three-character, and longer words, respectively. However, it has been argued that considering the inherent uncertainty in Chinese word segmentation, general-purpose segmentation algorithms should segment aggressively rather than conservatively (Wu 1998); consequently this corpus seems appropriate for our use.

Table 3
Distribution of word length in the corpus.

Length	Words	Characters
1	55.6%	36.2%
2	38.2%	49.9%
3	4.2%	8.2%
4	1.6%	4.0%
5	0.2%	0.8%
Over 5	0.2%	0.9%

Table 4
Results for five 500-word segments from the Chinese corpus (manually checked figures in parentheses).

File	Error rate	Recall	Precision	F-measure
1	1.2% (1.0%)	98.4% (98.6%)	98.8% (99.0%)	98.6% (98.8%)
2	3.6% (3.0%)	96.4% (96.8%)	96.4% (97.0%)	96.4% (96.9%)
3	4.2% (4.0%)	95.0% (95.8%)	95.8% (96.0%)	95.4% (95.9%)
4	6.4% (5.2%)	91.0% (92.2%)	93.4% (94.7%)	92.2% (93.4%)
5	6.6% (5.0%)	86.2% (90.4%)	92.9% (94.8%)	89.4% (92.5%)

Table 4 shows the results for five 500-word test files from the corpus. We took part of the corpus that was not used for training, divided it into 500-word segments, removed all spaces, and randomly chose five segments as test files. The results show an error rate varying from 1.2% to 6.6%. The resulting F-measures indicate that the new algorithm performs better than the one described in Hockenmaier and Brew (1998), who report an F-measure of 87.9 using trigram rules. This is particularly significant because the two algorithms use training and test data from the same source.

The results were also verified by checking them manually. This produces slightly different results, for two reasons. Firstly, human judgment sometimes accepts a segmentation as correct even though it does not correspond exactly with the corpus version. For example, the last word in 货 轮 船长 is counted as correct even though in the corpus it is written 货 轮船 长. Secondly, improper segmentations such as 常 熟市 and 朔 州市 occur in the corpus. When the program makes the same mistakes, it counts as correct in automatic checking, but incorrect in manual checking. These two kinds of error virtually canceled each other: when checked manually, file 3, for example, has five fewer errors for the first reason and six more for the second reason, giving error counts of 21 and 20 for automatic and manual checking, respectively.

5.4 Application to Independent Chinese Text Files

In a second test, models from this corpus were evaluated on completely separate data provided by the Institute of Computational Linguistics of Peking University. This contained 39 sentences (752 characters), some of which are compound sentences. Since no presegmented version was available, all checking was manual.

This test is interesting because it includes several sentences that are easily misunderstood, three of which are shown in Figure 8. In the first, which reads "I have learned a lot from it," the second and third characters combine into 'from it' and the

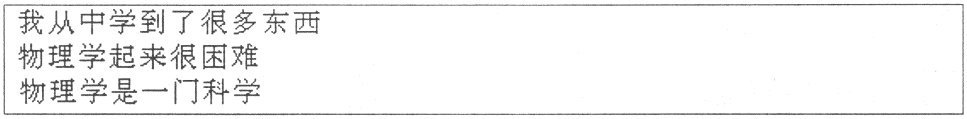

Figure 8
Three examples of easily misunderstood sentences.

Table 5
Error rate (mean and sd) for 1,000-word files from PH and Rocling corpora.

		Training	
		PH corpus	Rocling corpus
Testing	PH files	42 ± 10.23	169.2 ± 19.70
	Rocling files	133.4 ± 19.58	44.8 ± 10.83

fourth and fifth characters combine into 'have learned.' However, the third and fourth characters taken together mean 'middle school,' which does not occur in the meaning of the sentence. In the second and third sentences, the first three characters are the same. In the second, "physics is very hard to learn," the second and third characters should be separated by a space, so that the third character can combine with the following two characters to mean 'to learn.' However, in the third, "physics is one kind of science," the first three characters make a single word meaning 'physics.'

The error rate, recall and precision for this test material are 10.8%, 93.4%, and 89.6%, respectively. Performance is significantly worse than that of Table 4, because of the nature of the test file. Precision is distinctly lower than recall—recall fares better because many relevant words are still retrieved, whereas precision suffers because the automatic segmenter placed too many word boundaries compared with the manual judgment.

Two aspects of the training data have a profound influence on the model's accuracy. First, some errors are obviously caused by deficiencies in the training data, such as improperly segmented common words and names. Second, some errors stem from the topics covered by the corpus. It is not surprising that the error rate increases when the training and testing text represent different topic areas—such as training on news text and testing on medical text.

5.5 Application to the Rocling Corpus

The Rocling Standard Segmentation Corpus contains about two million presegmented words, represented in the Big5 coding scheme. We converted it to GB, used one million words for training, and compared the resulting model to that generated from the PH data, also trained on one million words. Both models were tested on 10 randomly chosen 1,000-word segments from each corpus (none of this material was used in training).

The results are shown in Table 5, in terms of the mean and standard deviation (sd) of the errors. When the training and testing files come from the same corpus, results are good, with around 42 (for PH) and 45 (for Rocling) errors per thousand words. Not surprisingly, performance deteriorates significantly when the PH model is used to segment the Rocling test files or vice versa.

Several differences between the corpora influence performance. Many English words are included in Rocling, whereas in PH only a few letters are used to rep-

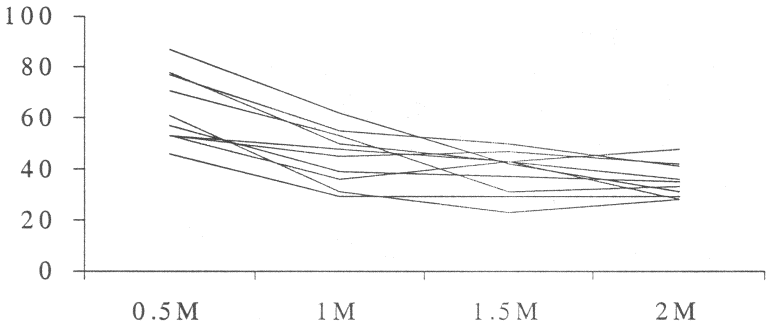

Figure 9
Effect of the amount of training data on the performance for each test file.

Table 6
Error rate (mean and sd) for different amounts of training data.

	0.5M words	1M words	1.5M words	2M words
Error	63.3 ± 13.69	44.8 ± 10.83	38.8 ± 8.60	35.1 ± 6.74

resent certain items. Percentages are represented as 90% or 九十% in Rocling, instead of 百分之九十 in the PH corpus. Quotation marks also differ: 「」 in Rocling but “” in PH. In addition, as is only to be expected in any large collection of natural language, typographical errors occur in both corpora.

The overall result indicates that our algorithm is robust. It performs well so long as the training and testing data come from the same source.

5.6 Effect of the Amount of Training Data

For the Rocling corpus, we experimented with different amounts of training data. Four models were trained with successively larger amounts of data, 0.5M, 1M, 1.5M, and 2M words, each training file being an extension of the text in the preceding training file. The four models were tested on the 10 randomly-chosen 1,000-word Rocling segments used before.

The results for the individual test files, in terms of error rate per thousand words, are shown in Figure 9 and summarized in Table 6. Larger training sets generally give smaller error, which is only to be expected—although the results for some individual test files flatten out and show no further improvement with larger training files, and in some cases more training data actually increases the number of errors. Overall, the error rate is reduced by about 25% for each doubling of the training data.

5.7 Models of Different Order

We have experimented with compression models of different orders on the PH corpus. Generally speaking, compression of text improves as model order increases, up to a point determined by the logarithm of the size of the training text. Typically, little compression is gained by going beyond order 5 models. For segmentation, we observe many errors when a model of order 1 is used. For order 3 models, most words are segmented with the same error rate as for order 5 models, though some words are missed when order 2 models are used.

Figure 10 shows some cases where the order 3 and order 5 models produce different results. Some order 5 errors are corrected by the order 3 model, though others

Order 3 model result	Order 5 model result
教育 之果	教育之 果
张军 首开纪录	张 军首开纪录

Figure 10
Results obtained when using order 3 and order 5 models.

appear even with the lower-order model. For example, both results in the first row are incorrect: no space should be inserted in this case, and the four characters should stand together. However, the order 3 result is to be preferred to the order 5 result because both two-character words do at least make sense individually, whereas the initial three characters in the order 5 version do not represent a word at all. In the second row, the order 5 result is incorrect because the second component does not represent a word. In the order 3 result, the first word, containing two characters, is a person's name. The second word could also be correct as it stands, though it would be equally correct if a space had been inserted between the two bigrams. On the whole, we find that the order 3 model gives the best results overall, although there is little difference between orders 3, 4, and 5.

6. Applications in a Digital Library

Word segmentation forms a valuable component of any Chinese digital library system. It improves full-text retrieval in two ways: higher-precision searching (that is, fewer false matches), and the ability to incorporate relevance ranking. This increases the effectiveness of full-text search and helps to provide users with better feedback. For example, one study concludes that the performance of an unsegmented character-based query is about 10% worse than that of the corresponding segmented query (Broglio, Callan, and Croft 1996). Many emerging digital library technologies also presuppose word segmentation—for example, text summarization, document clustering, and keyphrase extraction all rely on word frequencies. These would not work well on unsegmented text because character frequencies do not generally reflect word frequencies.

Once the source text in a digital library exceeds a few megabytes, full-text indexes are needed to process queries in a reasonable time (Witten, Moffat, and Bell 1999). Full-text indexing was developed using languages where word boundaries are notated (principally English), and the techniques that were developed rely on word-based processing. Although some techniques—for example stemming (Frakes 1992) and casefolding—are not applicable to Chinese information retrieval, many are. Examples include heuristics for relevance ranking, and query expansion using a language thesaurus.

Of course, full-text indexes can be built from individual characters rather than words. However, these will suffer from the problem of low precision—searches will return many irrelevant documents, where the same characters are used in contexts different from that of the query. To reduce false matches to a reasonable level, auxiliary indexes (for example, sentence indexes) will have to be created. These will be much larger than regular word-based indexes of paragraphs or documents, and will still not be as accurate.

Information retrieval systems often rank the results of each search, giving preference to documents that are more relevant to the query by placing them nearer the beginning of the list. Relevance metrics are based on the observation that infrequent

words are more important than common ones and should therefore rate more highly. Word segmentation is essential for this purpose, because the relationship between the frequency of a word and the frequency of the characters that appear within it is often very weak. Without word segmentation, the precision of the result set will be reduced because relevant documents are less likely to be close to the top of the list.

For example, the word 出国 ("to go abroad") is an infrequent word that appears only twenty times in the PH corpus. But its two characters occur frequently: 出 ("to go out") 13,531 times; and 国 ("country") 45,010 times. In fact 国 is the second most frequent character in the corpus, appearing in 443 separate words. Character-based ranking would place little weight on these two characters, even though they are extremely important if the query is 出国. The word 也 ("also") is another frequent character, appearing 4,553 times in the PH corpus. However, in 4,481 of those cases it appears by itself and contributes little to the meaning of the text. If a query contained both of these words, far more weight would be given to 也 than to the individual characters in 出国.

Word counts also give feedback on the effectiveness of a query. They help users judge whether their query was too wide or too narrow, and provide information on which of the terms are most appropriate.

Word-based processing is essential to a number of emergent new technologies in the digital library field. Statistical approaches are enjoying a resurgence in natural language analysis (Klavans and Resnik 1997): examples include text summarization, document clustering, and keyphrase extraction. All of these statistical approaches are based on words and word frequencies. For instance, keywords and keyphrases for a document can be determined automatically based on features such as the frequency of the phrase in the document relative to its frequency in an independent corpus of like material, and its position of occurrence in the document (Frank et al. 1999). A decomposition of text into its constituent words is an essential prerequisite for the application of such techniques.

7. Conclusions

The problem of word segmentation of Chinese text is important in a variety of contexts, particularly with the burgeoning interest in digital libraries and other systems that store and process text on a massive scale. Existing techniques are either linguistically based, using a dictionary of words, or rely on hand-crafted segmentation rules, or use adaptive models that have been specifically created for the purpose of Chinese word segmentation. We have developed an alternative based on a general-purpose character-level model of text—the kind of models used in the very best text compression schemes. These models are formed adaptively from training text.

The advantage of using character-level models is that they do not rely on a dictionary and therefore do not necessarily fail on unusual words. In effect, they can fall back on general properties of language statistics to process novel text. The advantage of basing models on a corpus of training text is that particular characteristics of the text are automatically taken into account in language statistics—as exemplified by the significant differences between the models formed for the PH and Rocling corpora.

Encouraging results have been obtained using the new scheme. Our results compare very favorably with the results of Hockenmaier and Brew (1998) on the PH corpus; unfortunately no other researchers have published quantitative results on a

standard corpus. Further work is needed to analyze the results of the Rocling corpus in more detail.

The next step is to use automatically segmented text to investigate the digital library applications we have described: information retrieval, text summarization, document clustering, and keyphrase extraction.

Acknowledgments

We are grateful to Stuart Inglis, Hong Chen, and John Cleary, who provided advice and assistance. The corrected version of Guo Jin's PH corpus and the Rocling corpus were provided by Julia Hockenmaier and Chris Brew at the University of Edinburgh and the Chinese Knowledge Information Processing Group of Academia Sinica, respectively. The Institute of Computational Linguistics of Peking University also provided some test material. Bill Teahan acknowledges the generous support of the Department of Information Technology, Lund University, Sweden. Thanks also to the anonymous referees who have helped us to improve the paper significantly.

References

Anderson, John B. and Seshadri Mohan. 1984. Sequential coding algorithms: A survey and cost analysis. *IEEE Transactions on Communications*, 32(2):169–176.

Bell, Timothy C., John G. Cleary, and Ian H. Witten. 1990. *Text Compression*. Prentice Hall, Englewood Cliffs, NJ.

Brill, Eric. 1995. Transformation-based error-driven learning and natural processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565.

Broglio, John, Jamie P. Callan, and W. Bruce Croft. 1994. Inquery system overview. In *Proceedings TIPSTER Text Program (Phase I)*. Morgan Kaufmann, San Francisco, CA, pages 47–67.

Broglio, John, Jamie P. Callan, and W. Bruce Croft. 1996. Technical issues in building an information system for Chinese. *CIIR Technical Report IR-86*, Center for Intelligent Information Retrieval, University of Massachusetts, Amherst.

Cheng, Kwok-Shing, Gilbert H. Young, and Kam-Fai Wong. 1999. A study on word-based and integral-bit Chinese text compression algorithms. *Journal of the American Society for Information Science*, 50(3):218–228.

Cleary, John G. and Ian H. Witten. 1984. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402.

Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. 1990. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

Dai, Yubin, Christopher S. G. Khoo, and Teck Ee Loh. 1999. A new statistical formula for Chinese text segmentation incorporating contextual information. In *Proceedings of ACM SIGIR99*, pages 82–89.

Frakes, William B. 1992. Stemming algorithms. In William B. Frakes and Ricardo Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, Englewood Cliffs, NJ, pages 131–160.

Frank, E., Gordon W. Paynter, Ian H. Witten, Carl Gutwin, and Craig G. Nevill-Manning. 1999. Domain-specific keyphrase extraction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 668–673. Stockholm, Sweden. Morgan Kaufmann, San Francisco, CA.

Hockenmaier, Julia and Chris Brew. 1998. Error driven segmentation of Chinese. *Communications of COLIPS*, volume 1, number 1, pages 69–84.

Howard, Paul Glor. 1993. *The Design and Analysis of Efficient Lossless Data Compression Systems*. Ph.D thesis, Brown University, Providence, RI.

Klavans, Judith L. and Philip Resnik, editors. 1997. The Balancing Act: Combining Symbolic and Statistical Approaches to Language. MIT Press, Cambridge, MA.

Kucera, Henry and W. Nelson Francis. 1967. *Computational Analysis of Present-Day American English*. Brown University Press, Providence, RI.

Liu, Yongquan. 1987. New advances in computers and natural language processing in China. *Information Science*, 8:64–70.

Lee, Kin Hong, Mau Kit Michael Ng, and Qin Lu. 1999. Text segmentation for Chinese spell checking. *Journal of the American Society for Information Science*, 50(9):751–759.

Palmer, David. 1997. A trainable rule-based algorithm for word segmentation. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*

(ACL '97), Madrid, 1997.
Ponte, Jay M. and W. Bruce Croft. 1996. USeg: A retargetable word segmentation procedure for information retrieval. Presented at the Symposium on Document Analysis and Information Retrieval '96 (SDAIR). UMass Technical Report TR96–2, University of Massachusetts, Amherst, MA.
Rabiner, Lawrence R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 77(2):257–286.
Sproat, Richard, Chilin Shih, William Gail, and Nancy Chang. 1996. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3):377–404.
Teahan, W. J. 1998. *Modelling English Text*. Ph.D thesis, University of Waikato, NZ.
Teahan, W. J., S. Inglis, John G. Cleary, and G. Holmes. 1998. Correcting English text using PPM models. In J. A. Storer and J. H. Reif, editors, *Proceeding Data Compression Conference*, pages 289–298, Los Alamitos, CA. IEEE Computer Society Press.
Witten, Ian H., Alistar Moffat, and Timothy C. Bell. 1999. *Managing gigabytes: compressing and indexing documents and images*. Second edition. Morgan Kaufmann, San Francisco, CA.
Wu, Dekai. 1998. A position statement on Chinese segmentation. Presented at the *Chinese Language Processing Workshop*, University of Pennsylvania, PA.
Wu, Dekai and Pascale Fung. 1994. Improving Chinese tokenization with linguistic filters on statistical lexical acquisition. In *ANLP-94, Fourth Conference on Applied Natural Language Processing*, pages 180–181, Stuttgart, October 94.
Wu, Zimin and Gwyneth Tseng. 1993. Chinese text segmentation for text retrieval achievements and problems. *JASIS*, 44(9):532–542.