Syntactic Processing Using the Generalized Perceptron and Beam Search

Yue Zhang[*]
University of Cambridge

Stephen Clark[**]
University of Cambridge

We study a range of syntactic processing tasks using a general statistical framework that consists of a global linear model, trained by the generalized perceptron together with a generic beam-search decoder. We apply the framework to word segmentation, joint segmentation and POS-tagging, dependency parsing, and phrase-structure parsing. Both components of the framework are conceptually and computationally very simple. The beam-search decoder only requires the syntactic processing task to be broken into a sequence of decisions, such that, at each stage in the process, the decoder is able to consider the top-n candidates and generate all possibilities for the next stage. Once the decoder has been defined, it is applied to the training data, using trivial updates according to the generalized perceptron to induce a model. This simple framework performs surprisingly well, giving accuracy results competitive with the state-of-the-art on all the tasks we consider.

The computational simplicity of the decoder and training algorithm leads to significantly higher test speeds and lower training times than their main alternatives, including log-linear and large-margin training algorithms and dynamic-programming for decoding. Moreover, the framework offers the freedom to define arbitrary features which can make alternative training and decoding algorithms prohibitively slow. We discuss how the general framework is applied to each of the problems studied in this article, making comparisons with alternative learning and decoding algorithms. We also show how the comparability of candidates considered by the beam is an important factor in the performance. We argue that the conceptual and computational simplicity of the framework, together with its language-independent nature, make it a competitive choice for a range of syntactic processing tasks and one that should be considered for comparison by developers of alternative approaches.

1. Introduction

In this article we study a range of syntactic processing tasks using a general framework for structural prediction that consists of the generalized perceptron (Collins 2002) and

[*] University of Cambridge Computer Laboratory, William Gates Building, 15 JJ Thomson Avenue, Cambridge, UK. E-mail: yue.zhang@cl.cam.ac.uk.

[**] University of Cambridge Computer Laboratory, William Gates Building, 15 JJ Thomson Avenue, Cambridge, UK. E-mail: stephen.clark@cl.cam.ac.uk.

Submission received: 10 November 2009; revised submission received: 12 August 2010; accepted for publication: 20 September 2010.

beam-search. We show that the framework, which is conceptually and computationally simple, is practically effective for structural prediction problems that can be turned into an incremental process, allowing accuracies competitive with the state-of-the-art to be achieved for all the problems we consider.

The framework is extremely flexible and easily adapted to each task. One advantage of beam-search is that it does not impose any requirements on the structure of the problem, for example, the optimal sub-problem property required for dynamic-programming, and can easily accommodate non-local features. The generalized perceptron is equally flexible, relying only on a decoder for each problem and using a trivial online update procedure for each training example. An advantage of the linear perceptron models we use is that they are global models, assigning a score to a complete hypothesis for each problem rather than assigning scores to parts which are then combined under statistical independence assumptions. Here we are following a recent line of work applying global discriminative models to tagging and wide-coverage parsing problems (Lafferty, McCallum, and Pereira 2001; Collins 2002; Collins and Roark 2004; McDonald, Crammer, and Pereira 2005; Clark and Curran 2007; Carreras, Collins, and Koo 2008; Finkel, Kleeman, and Manning 2008).

The flexibility of our framework leads to competitive accuracies for each of the tasks we consider. For word segmentation, we show how the framework can accommodate a *word-based* approach, rather than the standard and more restrictive character-based tagging approaches. For POS-tagging, we consider joint segmentation and POS-tagging, showing that a single beam-search decoder can be used to achieve a significant accuracy boost over the pipeline baseline. For Chinese and English dependency parsing, we show how both graph-based and transition-based algorithms can be implemented as beam-search, and then combine the two approaches into a single model which outperforms both in isolation. Finally, for Chinese phrase-structure parsing, we describe a global model for a shift-reduce parsing algorithm, in contrast to current deterministic approaches which use only local models at each step of the parsing process. For all these tasks we present results competitive with the best results in the literature.

In Section 2 we describe our general framework of the generic beam-search algorithm and the generalized perceptron. Then in the subsequent sections we describe each task in turn, based on conference papers including Zhang and Clark (2007, 2008a, 2008b, 2009, 2010), presented in our single coherent framework. We give an updated set of results, plus a number of additional experiments which probe further into the advantages and disadvantages of our framework. For the segmentation task, we also compare our beam-search framework with alternative decoding algorithms including an exact dynamic-programming method, showing that the beam-search method is significantly faster with comparable accuracy. For the joint segmentation and POS-tagging task, we present a novel solution using the framework in this article, and show that it gives comparable accuracies to our previous work (Zhang and Clark 2008a), while being more than an order of magnitude faster.

In Section 7 we provide further discussion of the framework based on the studies of the individual tasks. We present the main advantages of the framework, and give an analysis of the main reasons for the high speeds and accuracies achieved. We also discuss how this framework can be applied to a potential new task, and show that the comparability of candidates in the incremental process is an important factor to consider.

In summary, we study a general framework for incremental structural prediction, showing how the framework can be tailored to a range of syntactic processing problems to produce results competitive with the state-of-the-art. The conceptual and computational simplicity of the framework, together with its language-independent nature, make it a competitive choice that should be considered for comparison by developers of alternative approaches.

2. The Decoding and Training Framework

The framework we study in this article addresses the general structural prediction problem of mapping an input structure $x \in X$ onto an output structure $y \in Y$, where X is the set of possible inputs, and Y is the set of possible outputs. For example, for the problem of Chinese word segmentation, X is the set of raw Chinese sentences and Y is the set of all possible segmented Chinese sentences. For the problem of English dependency parsing, X is the set of all English sentences and Y is the set of all possible English dependency trees.

Given an input sentence x, the output $F(x)$ is defined as the highest scored among the possible output structures for x:

$$F(x) = \underset{y \in \mathrm{GEN}(x)}{\arg\max}\ \mathrm{Score}(y) \tag{1}$$

where $\mathrm{GEN}(x)$ denotes the set of possible outputs for an input sentence x, and $\mathrm{Score}(y)$ is some real-valued function on Y.

To compute $\mathrm{Score}(y)$, the output structure y is mapped into a global feature vector $\Phi(y) \in \mathcal{N}^d$. Here a feature is a count of the occurrences of a certain pattern in an output structure, extracted according to a set of feature templates, and d is the total number of features. The term **global feature vector** is used by Collins (2002) to distinguish between feature counts for whole sequences and the local feature vectors in maximum entropy tagging models, which are boolean-valued vectors containing the indicator features for one element in the sequence (Ratnaparkhi 1998). Having defined the feature vector, $\mathrm{Score}(y)$ is computed using a linear model:

$$\mathrm{Score}(y) = \Phi(y) \cdot \vec{w} \tag{2}$$

where $\vec{w} \in \mathcal{R}^d$ is the parameter vector of the model, the value of which is defined by supervised learning using the generalized perceptron.

For the general framework we study in this article, the output y is required to be built through an incremental process. Suppose that K incremental steps are taken in total to build y. The incremental change at the ith step $(0 < i \leq K)$ can be written as $\delta(y, i)$. For word segmentation, $\delta(y, i)$ can be an additional character added to the output; for shift-reduce parsing, $\delta(y, i)$ can be an additional shift-reduce action. Denoting the change to the global feature vector at the incremental step as $\Phi(\delta(y, i))$, the global feature vector $\Phi(y)$ can be written as $\Phi(y) = \sum_{i=1}^{K} \Phi(\delta(y, i))$. Hence, $\mathrm{Score}(y)$ can be computed incrementally by

$$\mathrm{Score}(y) = \sum_{i=1}^{K} \Phi(\delta(y, i)) \cdot \vec{w} \tag{3}$$

In the following sections, we describe the two major components of the general framework, that is, the general beam-search algorithm for finding $F(x) = \arg\max_{y \in \mathrm{GEN}(x)} \mathrm{Score}(y)$ for a given x, and the generalized perceptron for training $\vec{w}$.

2.1 Beam-Search Decoding

Given an input x, the output structure y is built incrementally. At each step, an incremental sub-structure is added to the partially built output. Due to structural ambiguity, different sub-structures can be built. Taking POS-tagging for example, the incremental sub-structure for each processing step can be a POS-tag assigned to the next input word. Due to structural ambiguity, different POS-tags can be assigned to a word, and the decoding algorithm searches for the particular path of incremental steps which builds the highest scored output.

We present a generic beam-search algorithm for our decoding framework, which uses an agenda to keep the B-best partial outputs at each incremental step. The partially built structures, together with useful additional information, are represented as a set of state items. Additional information in a state item is used by the decoder to organize the current structures or keep a record of the incremental process. For POS-tagging it includes the remaining input words yet to be assigned POS-tags; for a shift-reduce parser, it includes the stack structure for the shift-reduce process and the incoming queue of unanalyzed words.

The agenda is initialized as empty, and the state item that corresponds to the initial structure is put onto it before decoding starts. At each step during decoding, each state item from the agenda is extended with one incremental step. When there are multiple choices to extend one state item, multiple new state items are generated. The new state items generated at a particular step are ranked by their scores, and the B-best are put back onto the agenda. The process iterates until a stopping criterion is met, and the current best item from the agenda is taken as the output.

Pseudo code for the generic beam-search algorithm is given in Figure 1, where the variable *problem* represents a particular task, such as word segmentation or dependency parsing, and the variable *candidate* represents a state item, which has a different definition for each task. For example, for the segmentation task, a *candidate* is a pair, consisting of the partially segmented sentence and the remaining character sequence yet to be segmented. The *agenda* is an ordered list, used to keep all the state items generated at each stage, ordered by score. The variable *candidates* is the set of state items that can be used to generate new state items, that is, the B-best state items from the previous stage. B is the number of state items retained at each stage.

```
function BEAM-SEARCH(problem, agenda, candidates, B)

  candidates ← {STARTITEM(problem)}
  agenda ← CLEAR(agenda)
  loop do
    for each candidate in candidates
      agenda ← INSERT(EXPAND(candidate, problem), agenda)
    best ← TOP(agenda)
    if GOALTEST(problem, best)
      then return best
    candidates ← TOP-B(agenda, B)
    agenda ← CLEAR(agenda)
```

Figure 1
The generic beam-search algorithm.

STARTITEM initializes the start state item according to the problem; for example, for the segmentation task, the start state item is a pair consisting of an empty segmented sentence and the complete sequence of characters waiting to be segmented. CLEAR removes all items from the agenda. INSERT puts one or more state items onto the agenda. EXPAND represents an incremental processing step, which takes a state item and generates new state items from it in all possible ways; for example, for the segmentation task, EXPAND takes the partially segmented sentence in a state item, and extends it in all possible ways using the first character in the remaining character sequence in the state item. TOP returns the highest scoring state item on the agenda. GOALTEST checks whether the incremental decoding process is completed; for example, for the segmentation task, the process is completed if the state item consists of a fully segmented sentence and an empty remaining character sequence. TOP-B returns the B-highest scoring state items on the agenda, which are used for the next incremental step.

State items in the agenda are ranked by their scores. Suppose that K incremental steps are taken in total to build an output y. At the ith step ($0 < i \leq K$), a state item in the agenda can be written as $candidate^i$, and we have

$$\text{Score}(candidate^i) = \sum_{n=1}^{i} \Phi(\delta(candidate^i, n)) \cdot \vec{w}$$

Features for a state item can be based on both the partially built structure and the additional information we mentioned earlier.

The score of a state item can be computed incrementally as the item is built. The score of the start item is 0. At the ith step ($0 < i \leq K$), a state item $candidate^i$ is generated by extending an existing state item $candidate^{i-1}$ on the agenda with $\delta(candidate^i, i)$. In this case, we have

$$\text{Score}(candidate^i) = \text{Score}(candidate^{i-1}) + \Phi(\delta(candidate^i, i)) \cdot \vec{w}$$

Therefore, when a state item is extended, its score can be updated by adding the incremental score of the step $\Phi(\delta(candidate^i, i)) \cdot \vec{w}$. The nature of the scoring function means that, given appropriately defined features, it can be computed efficiently for both the incremental decoding and training processes.

Because the correct item can fall out of the agenda during the decoding process, the general beam-search framework is an approximate decoding algorithm. Nevertheless, empirically this algorithm gives competitive results on all the problems in this article.

2.2 The Generalized Perceptron

The perceptron learning algorithm is a supervised training algorithm. It initializes the parameter vector as all zeros, and updates the vector by decoding the training examples. For each example, the output structure produced by the decoder is compared with the correct structure. If the output is correct, no update is performed. If the output is incorrect, the parameter vector is updated by adding the global feature vector of the training example and subtracting the global feature vector of the decoder output. Intuitively, the training process is effectively coercing the decoder to produce the correct output for each training example. The algorithm can perform multiple passes over the same training sentences. In all experiments, we decide the number of training iterations using a set of development test data, by choosing the number that gives the highest

```
Inputs: training examples $(x_i, y_i)$
Initialization: set $\vec{w} = 0$
Algorithm:
  for $r = 1..P, i = 1..N$
    calculate $z_i$ = decode($x_i$)
    if $z_i \neq y_i$
      $\vec{w} = \vec{w} + \Phi(y_i) - \Phi(z_i)$
Outputs: $\vec{w}$
```

Figure 2
The generalized perceptron algorithm, adapted from Collins (2002).

development test accuracy as the final number in testing. Figure 2 gives the algorithm, where N is the number of training sentences and P is the number of passes over the data.

The averaged perceptron algorithm (Collins 2002) is a standard way of reducing overfitting on the training data. It was motivated by the voted-perceptron algorithm (Freund and Schapire 1999) and has been shown to give improved accuracy over the non-averaged perceptron on a number of tasks. Let N be the number of training sentences, P the number of training iterations, and $\vec{w}^{i,r}$ the parameter vector immediately after the ith sentence in the rth iteration. The averaged parameter vector $\vec{\gamma} \in \mathcal{R}^d$ is defined as

$$\vec{\gamma} = \frac{1}{PN} \sum_{i=1..N, r=1..P} \vec{w}^{i,r}$$

and it is used instead of $\vec{w}$ as the model parameters. We use the averaged perceptron for all the tasks we consider.

We also use the early-update strategy of Collins and Roark (2004), which is a modified version of the perceptron algorithm specifically for incremental decoding using beam search. At any step during the decoding process to calculate z_i, if all partial candidates in the agenda are incorrect, decoding is stopped and the parameter vector is updated according to the current best candidate in the agenda and the corresponding gold-standard partial output. To perform early-update, the decoder needs to keep a version of the correct partial output for each incremental step, so that the parameter values are adjusted as soon as the beam loses track of the correct state item. The intuition is to force the beam to keep the correct state item at every incremental step, rather than learning only the correct overall structure. This strategy has been shown to improve the accuracy over the original perceptron for beam-search decoding.

In summary, our general framework consists of a global linear model, which is trained by the averaged perceptron, and a beam-search decoder. When applied to a particular task, the structure of a state item as well as some of the functions in the decoder need to be instantiated. In the following sections, we show how the general framework can be applied to Chinese word segmentation, joint segmentation and POS-tagging, Chinese and English dependency parsing, and Chinese constituent parsing.

3. Word Segmentation

Chinese word segmentation (CWS) is the problem of finding word boundaries for Chinese sentences, which are written as continuous character sequences. Other languages, including Japanese and Thai, also have the problem of word segmentation, and typical statistical models for CWS can also be applied to them.

Word segmentation is a problem of ambiguity resolution, often requiring knowledge from a variety of sources. Out-of-vocabulary (OOV) words are a major source of ambiguity. For example, a difficult case occurs when an OOV word consists of characters which have themselves been seen as words; here an automatic segmentor may split the OOV word into individual single-character words. Typical examples of unseen words include Chinese names, translated foreign names, and idioms.

The segmentation of known words can also be ambiguous. For example, 这里面 should be 这里 (here) 面 (flour) in the sentence 这里面和米很贵 (flour and rice are expensive here) or 这 (here) 里面 (inside) in the sentence 这里面很冷 (it's cold inside here). The ambiguity can be resolved with information about the neighboring words. In comparison, for the sentence 洽谈会很成功, possible segmentations include 洽谈 (the discussion) 会 (will) 很 (very) 成功 (be successful) and 洽谈会 (the discussion meeting) 很 (very) 成功 (be successful). The ambiguity can only be resolved with contextual information outside the sentence. Human readers often use semantics, contextual information about the document, and world knowledge to resolve segmentation ambiguities.

There is no fixed standard for Chinese word segmentation. Experiments have shown that there is only about 75% agreement among native speakers regarding the correct word segmentation (Sproat et al. 1996). Also, specific NLP tasks may require different segmentation criteria. For example, 北京银行 could be treated as a single word (Bank of Beijing) for machine translation, although it is more naturally segmented into 北京 (Beijing) 银行 (bank) for tasks such as text-to-speech synthesis. Therefore, supervised learning with specifically defined training data has become the dominant approach.

Following Xue (2003), the standard approach for building a statistical CWS model is to treat CWS as a sequence labeling task. A tag is assigned to each character in the input sentence, indicating whether the character is a single-character word or the start, middle, or end of a multi-character word. The context for disambiguation is normally a five-character window with the current character in the middle. We call these methods **character-based** word segmentation. The advantage of character-based segmentation is that well-known tagging approaches can be applied directly to the CWS problem.

There are various character-based models in the literature. They differ mainly in the learning algorithm and the features used. Several discriminative learning algorithms have been applied to the character-based systems. Examples include Xue (2003), Peng, Feng, and McCallum (2004), and Wang et al. (2006), which use maximum entropy and conditional random field models, and Jiang et al. (2008), which uses the perceptron model. The standard feature set is that defined by Ng and Low (2004), though other feature sets are reported to improve the accuracy (Zhao, Huang, and Li 2006). Zhao, Huang, and Li (2006) also showed that the best accuracy for conditional random field (CRF) models is given by using a set of six character segmentation tags, rather than the standard set {*beginning, middle, end, single*} shown previously. Standard search algorithms for sequence tagging have been applied to the decoding process, such as the dynamic-programming algorithm and beam-search.

A disadvantage of character-based models is the use of limited contextual information. For these methods, context is confined to the neighboring characters. Other contextual information, in particular the surrounding words, is not included. Consider the sentence 中国外企业, which can be from 其中 (among which) 国外 (foreign) 企业 (companies), or 中国 (in China) 外企 (foreign companies) 业务 (business). Note that the five-character window surrounding 外 is the same in both cases, making the tagging decision for that character difficult given the local window. The correct decision can be made, however, by comparing the two three-word windows containing this character.

In Zhang and Clark (2007) we proposed a *word-based approach* to segmentation, which provides a direct solution to the problem. In comparison with the character-based approach, our segmentor does not map the CWS problem into sequence labeling. By using a global linear model, it addresses the segmentation problem directly, extracting word-based features from the output segmented structure. Hence we call our word segmentation model the **word-based approach**. In fact, word-based segmentors can be seen as a generalization of character-based segmentors, because any character-based features can be defined in a word-based model.

In the following sections, we describe a word-based segmentor using the general framework of this article, which is slightly different from the original system we proposed in Zhang and Clark (2007). We compare the accuracies of this segmentor and the 2007 segmentor, and report a set of improved results for our 2007 segmentor using a better method to optimize the number of training iterations. We then study alternative decoders to the general framework, including a Viterbi inference algorithm and a multiple-beam search algorithm, and provide discussion on the general framework and word-based segmentation.

3.1 Instantiating the General Framework

In this section we formulate our word-based segmentor as an instance of the general framework of this article. Our segmentor builds a candidate segmentation incrementally, one character at a time. When each character is processed, it is either combined with the last word of the partial candidate that has been built so far, or added to the candidate as the start of a new word. The same process repeats for each input character, and therefore runs in linear time.

For ambiguity resolution, we use a beam-search decoding algorithm to explore the search space. Initially containing only an empty sentence, an agenda is used to keep a set of candidate items for each processing step. When an input character is processed, it is combined with each candidate in the agenda in the two aforementioned ways, and two new candidates are generated. At the end of each step, the B-best newly generated candidates are kept in the agenda for the next processing step. When the input sentence is exhausted, the top candidate from the agenda is taken as the output.

This decoding process can be expressed as an instance of the generic algorithm in Figure 1. For the word segmentation problem, a state item in the algorithm is a pair $\langle S, Q \rangle$, where S contains part of the input that has been segmented, and Q contains the rest of the input sentence as a queue of incoming characters. The initial state item STARTITEM(*word_segmentation*) contains an empty sentence, and an incoming queue of the whole input sentence. EXPAND(*candidate, word_segmentation*) pops the first character from the incoming queue, and adds it to the partial segmented sentence in *candidate* in two different ways to generate two new state items: It either appends the character to the last word in the sentence or joins it as the start of a new word in the sentence. Finally, GOALTEST(*word_segmentation, best*) returns true if *best* contains a fully segmented input sentence, and therefore an empty incoming queue, and false otherwise.

The score of a segmented sentence is computed by the global linear model in Equation (2), where the parameter vector $\vec{w}$ for the model is computed by the early-update version of the perceptron training algorithm described in Section 2.2. Our word segmentor computes the global feature vector $\Phi(y)$ incrementally according to Equation (3), where for the ith character, $\Phi(\delta(y, i))$ is computed using the feature templates in Table 1, according to whether the character is appended to or separated from its previous character.

Table 1
Feature templates for the word segmentor.

	Feature template	When c_0 is
1	w_{-1}	separated
2	$w_{-1}w_{-2}$	separated
3	w_{-1}, where $len(w_{-1}) = 1$	separated
4	$start(w_{-1})len(w_{-1})$	separated
5	$end(w_{-1})len(w_{-1})$	separated
6	$end(w_{-1})c_0$	separated
7	$c_{-1}c_0$	appended
8	$begin(w_{-1})end(w_{-1})$	separated
9	$w_{-1}c_0$	separated
10	$end(w_{-2})w_{-1}$	separated
11	$start(w_{-1})c_0$	separated
12	$end(w_{-2})end(w_{-1})$	separated
13	$w_{-2}len(w_{-1})$	separated
14	$len(w_{-2})w_{-1}$	separated

w = word; c = character. The index of the current character is 0.

3.2 Comparisons with Zhang and Clark (2007)

Both the segmentor of this article and our segmentor of Zhang and Clark (2007) use a global linear model trained discriminatively using the perceptron. However, when comparing state items in the agenda, our 2007 segmentor treated full words in the same way as partial words, scoring them using the same feature templates. This scoring mechanism can potentially have a negative effect on the accuracy. In this article, we take a different strategy and apply full-word feature templates only when the next input character is separated from the word. In fact, most of the feature templates in Table 1 are related to full word information, and are applied when separating the next character. This method thus gives a clear separation of partial word and full word information. We also applied early-update in this article, so that the training process is closely coupled with beam-search decoding. In Zhang and Clark (2007) we performed the standard global discriminative learning.

3.3 Combining Word-Based and Character-Based Segmentation

As stated earlier, a character-based segmentor maps the segmentation problem into a sequence labeling problem, where labels are assigned to each input character to represent its segmentation. Our word-based approach does not map the segmentation problem into a labeling task but solves it directly. In this article, we further show that the flexibility of the word-based approach, enabled by our general framework, allows the combination of a character-based sub-system into our word-based system. The intuition is simple: Both perceptron learning and beam-search decoding allow arbitrary features, and therefore features from a typical character-based system can be incorporated into our segmentor to provide further information. Though character-based segmentors can also leverage word-level features indirectly, the labeling nature prevents them from direct use of word information.

We follow the convention of character-based segmentation, and define the set of segmentation tags as {B, E, M, S}. The tags B, E, M represent the character being the beginning, end, and middle of a multiple-character word, respectively, and the tag S represents the character being a single-character word.

The character-based features that we incorporate into our segmentor are shown in Table 2, which consist of unigram, bigram, and trigram information in the three-character window surrounding the current character, paired with the segmentation tag of the current character. To distinguish this system from our system without combination of character-based information, we call our segmentor in Section 3.1 the **pure word-based segmentor** and the segmentor that uses character-based features the **combined segmentor** in our experimental sections.

3.4 Experiments

We performed two sets of experiments. In the first set of experiments, we used the Chinese Treebank (CTB) data to study the speed/accuracy tradeoff by varying the size of the beam. In the second set of experiments, we used training and testing sets from the first and second international Chinese word segmentation bakeoffs (Sproat and Emerson 2003; Emerson 2005) to compare the accuracies to other models in the literature, including our segmentor of Zhang and Clark (2007).

F-score is used as the accuracy measure: $2pr/(p+r)$, where precision p is the percentage of words in the decoder output that are segmented correctly, and recall r is the percentage of gold-standard output words that are correctly segmented by the decoder.

CWS systems are evaluated by two types of tests. The **closed test**s require that the system is trained only with a designated training corpus. Any extra knowledge is not allowed, including common surnames, Chinese and Arabic numbers, European letters, lexicons, parts-of-speech, semantics, and so on. The **open test**s do not impose such restrictions. Open tests measure a model's capability to utilize extra information and domain knowledge, which can lead to improved performance, but because this extra information is not standardized, direct comparison between open test results is less informative. In this article, we focus only on the closed test.

3.4.1 Speed/Accuracy Tradeoff. We split CTB5 into training, development test, and test sets as shown in Table 3, where the development test data are used to determine the number of training iterations, which are used to obtain the final accuracies on the test data. We measure the accuracies on the test data with various beam-sizes, and plot the speed/accuracy tradeoff graph in Figure 3. Each point in the figure, from right to left, corresponds to beam size $B = 1$, 2, 4, 8, 16, 32, and 64, respectively. Speed is measured in the number of thousand characters per second, and accuracy is calculated using F-score.

As the size of the beam increases, the speed of the segmentor decreases. Because a larger part of the search is explored with an increased beam size, the accuracy of

Table 2
Feature templates of a typical character-based word segmentor.

	Feature template	When c_0 is
1	$c_i s_0, i \in \{-1, 0, 1\}$	separated, appended
2	$c_{i-1} c_i s_0, i \in 0, 1$	separated, appended
3	$c_{-1} c_0 c_1 s_0$	separated, appended

c = character; s = segmentation tag. The index of the current character is 0.

Table 3
Training, development, and test data for word segmentation on CTB5.

	Sections	Sentences	Words
Training	1–270, 400–931, 1001–1151	18,085	493,892
Dev	301–325	350	6,821
Test	271–300	348	8,008

the decoder has the potential to increase. This explains the increased accuracies when B increases from 1 to 16. However, the amount of increase drops when the beam size increases.

3.4.2 Closed Test on the SIGHAN Bakeoffs. Four training and testing corpora were used in the first bakeoff (Sproat and Emerson 2003), including the Academia Sinica Corpus (AS), the Penn Chinese Treebank Corpus (CTB), the Hong Kong City University Corpus (CU), and the Peking University Corpus (PU). However, because the testing data from the Penn Chinese Treebank Corpus is currently unavailable to us, we excluded this corpus from our experiments. The corpora are encoded in GB (PU, CTB) and BIG5 (AS, CU). In order to test them consistently in our system, they are all converted to UTF8 without loss of information.

The results are shown in Table 4. We follow the format from Peng, Feng, and McCallum (2004), where each row represents a CWS model. The first three columns represent tests with the AS, CU, and PU corpora, respectively. The best score in each column is shown in bold. The last two columns represent the average accuracy of each model over the tests it participated in (SAV), and our average over the same tests (OAV), respectively. The first eight rows represent models from Sproat and Emerson (2003) that participated in at least one closed test from the table, row "Peng" represents the CRF model from Peng, Feng, and McCallum (2004), row "Zhang 2007" represents our model as reported in Zhang and Clark (2007), and the last two rows represent our model in this article, using only word-based features in Table 1 and combined features in Tables 1 plus 2, respectively.

In Zhang and Clark (2007) we fixed the number of training iterations to six for all experiments, according to a separate set of development data. An alternative way to decide the number of training iterations is to set apart 10% from the training data

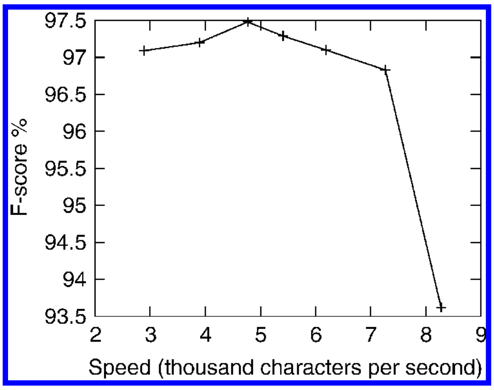

Figure 3
Speed/accuracy tradeoff of the segmentor.

Table 4
The accuracies of various word segmentors over the first SIGHAN bakeoff data.

	AS	CU	PU	SAV	OAV
S01	93.8	90.1	**95.1**	93.0	**95.5**
S04			93.9	93.9	**94.8**
S05	94.2		89.4	91.8	**95.9**
S06	94.5	92.4	92.4	93.1	**95.5**
S08		90.4	93.6	92.0	**94.8**
S09	96.1		94.6	95.4	**95.9**
S10			94.7	94.7	**94.8**
S12	95.9	91.6		93.8	**95.9**
Peng	95.6	92.8	94.1	94.2	**95.5**
Zhang 2007	96.5	94.6	94.0	95.0	**95.5**
Zhang 2007*	96.9	94.6	94.1	95.2	**95.5**
this article pure	**97.0**	94.6	94.6	95.4	**95.5**
this article combined	96.9	**94.8**	**94.8**		

The best score in each column and the best average in each row is in **boldface**.
*Zhang 2007 with the (Carreras, Surdeanu, and Marquez 2006) method applied (see text for details).

as development test data, and use the rest for development training. For testing, all training data are used for training, with the number of training iterations set to be the number which gave the highest accuracy during the development experiments. This method was used by Carreras, Surdeanu, and Marquez (2006) in their parsing model. We apply it to our segmentor model in this article. Moreover, we also use this method to decide the number of training iterations for our system of Zhang and Clark (2007), and show the accuracies in row "Zhang 2007*".

For each row the best average is shown in bold. We achieved the best accuracy in all three corpora, and better overall accuracy than all the other models using the method of this article. Our new method to decide the number of training iterations also gave improved accuracies compared to our 2007 model. The combination of character-based features and our original word-based features gave slight improvement in the overall accuracy.

Four training and testing corpora were used in the second bakeoff (Emerson 2005), including the Academia Sinica corpus (AS), the Hong Kong City University Corpus (CU), the Peking University Corpus (PK), and the Microsoft Research Corpus (MR). Different encodings were provided, and the UTF8 data for all four corpora were used in our experiments.

Following the format of Table 4, the results for this bakeoff are shown in Table 5. We chose the three models that achieved at least one best score in the closed tests from Emerson (2005), as well as the sub-word-based model of Zhang, Kikui, and Sumita (2006) for comparison. Row "Zh-a" and "Zh-b" represent the pure sub-word CRF model and the confidence-based combination of the CRF and rule-based models, respectively. Again, our model achieved better overall accuracy than all the other models. The combination of character-based features improved the accuracy slightly again.

Table 5
The accuracies of various word segmentors over the second SIGHAN bakeoff data.

	AS	CU	PK	MR	SAV	OAV
S14	94.7	94.3	95.0	96.4	95.1	**95.6**
S15b	95.2	94.1	94.1	95.8	94.8	**95.6**
S27	94.5	94.0	95.0	96.0	94.9	**95.6**
Zh-a	94.7	94.6	94.5	96.4	95.1	**95.6**
Zh-b	95.1	95.1	**95.1**	97.1	**95.6**	**95.6**
Zhang 2007	94.6	95.1	94.5	97.2	95.4	**95.6**
Zhang 2007*	95.0	95.1	94.6	**97.3**	95.5	**95.6**
This article pure	95.1	**95.2**	94.4	**97.3**	95.5	**95.6**
This article combined	**95.4**	95.1	94.4	**97.3**		

The best score in each column and the best average in each row is in **boldface**.
*Zhang 2007 with the (Carreras, Surdeanu, and Marquez 2006) method applied (see text for details).

3.5 Alternative Decoding Algorithms

Besides the general framework of this article, there are various alternative learning and decoding algorithms for a discriminative linear model applied to the word segmentation problem, using the same feature templates we defined. In this section, we study two alternative decoding algorithms to the beam-search decoder, including a multiple-beam search algorithm, which can be viewed as an alternative decoder specifically designed for the word segmentation and joint segmentation and tagging problems, and a dynamic-programming algorithm. Both algorithms explore a larger part of the search space than the single beam-search algorithm, and we compare the accuracy and speed of these algorithms within the generalized perceptron learning framework.

3.5.1 A Multiple-Beam Search Decoder. In Zhang and Clark (2008a) we proposed a multiple-beam decoder for the problem of joint word segmentation and POS-tagging, in which state items only contain complete words. This algorithm can be naturally adapted for word segmentation. Compared with the single-beam decoder, it explores a larger fraction of the search space. Moreover, the multiple-beam decoder does not have the problem of comparing partial words with full words in a single agenda, which our segmentor of Zhang and Clark (2007) has. We implement this decoder for segmentation and compare its accuracies with our single-beam decoder.

Instead of a single agenda, the multiple-beam algorithm keeps an agenda for each character in the input sentence, recording the best partial candidates ending with the character. Like the single beam decoder, the input sentence is processed incrementally. However, at each stage, partial sequence candidates are available at all previous characters. Therefore, the decoder can examine all candidate words ending with the current character. These possible words are combined with the relevant partial candidates from the previous agendas to generate new candidates, which are then inserted into the agenda for the current character. The output of the decoder is the top candidate in the last agenda, representing the best segmentation for the whole sentence. The multiple-beam search decoder explores a larger number of candidate outputs compared to the single-beam search. To improve the running speed, a maximum word length record is kept to limit the length of candidate words.

Because the multiple-beam decoder can also be applied to the joint segmentation and POS-tagging problem in Section 4, we describe this algorithm by extending the generic beam-search algorithm in Figure 1. Three modifications are made to the original algorithm, shown in Figure 4. First, the B-best *candidates* generated in each processing step are kept in *prev_topBs*. Here *prev_topBs* can be seen as a list of the *candidates* in the original search algorithm, with *prev_topBs*[k] containing the current items with size k, and *prev_topBs*[0] containing only the start item. An extra loop is used to enumerate all state items in *prev_topBs* for the generation of new state items. Second, variable k is used to represent the size of the state items to be generated, and Expand generates only state items with size k, taking k as an extra parameter. Third, the B-best newly generated state items are appended to the back of *prev_topBs*, without removing previous state items in *prev_topBs*. The algorithm thereby keeps track of kB state items instead of B at the kth processing stage, and explores a larger subset of the exponential search space.

The rest of the algorithm is the same as the original algorithm in Figure 1. To instantiate this generic algorithm for word segmentation, StartItem(*word_segmentation*) consists of an empty sentence S and a queue Q containing the full input sentence; Expand(*candidate*, k, *word_segmentation*) generates a single new state item by popping characters on the incoming queue from the front until the kth character (k is the index in the input sentence rather than the queue itself), and appending them as a new word to *candidate*; and GoalTest(*word_segmentation*, *best*) returns true if *best* consists of a complete segmented sentence and an empty incoming queue.

As before, the linear model from Section 2 is applied directly to score state items, and the model parameters are trained with the averaged perceptron algorithm. The features for a state item are extracted according to the feature templates in Table 1.

3.5.2 A Dynamic-Programming Decoder. Given the feature templates that we define, a dynamic-programming algorithm can be used to explore the whole search space in cubic time. The idea is to reduce the search task into overlapping sub-problems.

```
function Multiple-Beam-Search(problem, agenda, prev_topBs, B)

  prev_topBs ← {{StartItem(problem)}}
  agenda ← Clear(agenda)
  $k \leftarrow 0$
  loop do
    $k \leftarrow k + 1$
    for each candidates in prev_topBs
      for each candidate in candidates
        agenda ← Insert(Expand(candidate, k, problem), agenda)
    if GoalTest(problem, agenda)
      then return Top(agenda)
    candidates ← Top-B(agenda, B)
    prev_topBs ← Append(prev_topBs, candidates)
    agenda ← Clear(agenda)
```

Figure 4
The extended generic beam-search algorithm with multiple beams.

Suppose that the input has n characters; one way to find the highest-scored segmentation is to first find the highest-scored segmentations with the last word being characters $b..n-1$, where $b \in 0..n-1$, respectively, and then choose the highest-scored one from these segmentations. In order to find the highest-scored segmentation with the last word being characters $b, ..n-1$, the last word needs to be combined with all different segmentations of characters $0..b-1$ so that the highest scored can be selected. However, because the largest-range feature templates span only over two words (see Table 1), the highest scored among the segmentations of characters $0..b-1$ with the last word being characters $b'..b-1$ will also give the highest score when combined with the word $b..n-1$. As a result, the highest-scored segmentation with the last word being characters $b..n-1$ can be found as long as the highest-scored segmentations of $0..b-1$ with the last word being $b'..b-1$ are found, where $b' \in 0..b-1$. With the same reasoning, the highest-scored segmentation of characters $0..b-1$ with the last word being $b'..b-1$ can be found by choosing the highest-scored one among the highest-scored segmentations of $0..b'-1$ with the last word being $b''..b'-1$, where $b'' \in 0..b'-1$. In this way, the search task is reduced recursively into smaller problems, where in the simplest case the highest-scored segmentation of characters $0..e$ with the last word being characters $0..e(e \in 0..n-1)$ are known. And the final highest-scored segmentation can be found by incrementally finding the highest-scored segmentations of characters $0..e(e \in 0..n-1)$ with the last word being $b..e(b \in 0..e)$.

The pseudo code for this algorithm is shown in Figure 5. It works by building an n by n table *chart*, where n is the number of characters in the input sentence *sent*. $chart[b,e]$ records the highest scored segmentation from the beginning to character e, with the last word starting from character b and ending at character e. $chart[0,e]$ can be computed directly for $e = 0..n-1$, whereas $chart[b,e]$ needs to be built by combing the best segmentation on $sent[0, b-1]$ and $sent[b,e]$, for $b > 0$. The final output is the best among $chart[b, n-1]$, with $b = 0..n-1$. The reason for recording partial segmentations with different final words separately (leading to cubic running time) is the word bigram feature template. Note that with a larger feature range, exact inference with dynamic-programming can become prohibitively slow.

```
Inputs: raw sentence sent (length $n$)
Variables: an $n$ by $n$ table chart, where $chart[b,e]$ stores the best scored (partial)
           segmentation of the characters from the begining of the sentence to character
           $e$, with the last word spanning over the characters from $b$ until $e$;
           character index $b$ for the start of word;
           character index $e$ for the end of word;
           character index $p$ for the start of the previous word.
Initialization:
  for $e = 0..n-1$:
    $chart[0,e] \leftarrow$ a single word $sent[0..e]$
Algorithm:
  for $e = 0..n-1$:
    for $b = 1..e$:
      $chart[b,e] \leftarrow$ the highest scored segmentation among those derived by combining
                    $chart[p, b-1]$ with $sent[b,e]$, for $p = 0..b-1$
Outputs: the highest scored segmentation among $chart[b, n-1]$, for $b = 0..n-1$
```

Figure 5
A dynamic-programming algorithm for word segmentation.

Table 6
Comparison between three different decoders for word segmentation.

	Bakeoff 1			Bakeoff 2				
	AS	CU	PU	AS	CU	PU	MS	Average
SB F-measure	96.9	**94.6**	94.1	95.0	**95.1**	**94.6**	**97.3**	95.4
SB sent/sec	212	145	202	358	115	177	105	188
SB char/sec	3,054	6,846	4,808	5,263	5,333	5,870	4,963	5,162
SB # features	4.0M	0.5M	1.5M	3.9M	1.8M	1.5M	2.7M	2.3M
MB F-measure	97.0	94.5	94.1	95.0	95.0	94.4	**97.3**	95.3
MB sent/sec	147	7	13	167	7	11	5	51
MB char/sec	2,118	331	187	2,455	325	365	236	859
MB # features	4.0M	0.6M	1.5M	3.9M	1.8M	1.5M	2.7M	2.3M
DP F-measure	**97.1**	**94.6**	**94.3**	95.0	95.0	94.5	97.2	95.4
DP sent/sec	131	3	6	142	4	4	2	42
DP char/sec	1,887	142	86	2,087	185	133	95	659
DP # features	4.0M	0.5M	1.5M	3.9M	1.7M	1.4M	2.7M	2.3M

3.5.3 Experiments. Table 6 shows the comparison between the single-beam (SB), multiple-beam (MB), and dynamic-programming (DP) decoders by F-score and speed.[1] Speed is measured by the number of sentences (sent) and characters (char) per second (excluding model loading time). We also include the size of the models in each case. The slight difference in model size between different methods is due to different numbers of negative features generated during training. The single-beam search algorithm achieved significantly higher speed than both the multiple-beam and the dynamic-programming algorithms, whereas the multiple-beam search algorithm ran slightly faster than the dynamic-programming algorithm. Though addressing the comparability issue and exploring a larger number of candidate output segmentations, neither multiple-beam search nor dynamic programming gave higher accuracy than the single-beam search algorithm overall. One of the possible reasons is that the perceptron algorithm adjusts its parameters according to the mistakes the decoder makes: Although the single-beam might make more mistakes than the multiple-beam given the same model, it does not necessarily perform worse with a specifically tailored model.

4. Joint Segmentation and Part-of-Speech Tagging

Joint word segmentation and POS-tagging is the problem of solving word segmentation and POS-tagging simultaneously. Traditionally, Chinese word segmentation and POS-tagging are performed in a pipeline. The output from the word segmentor is taken as the input for the POS-tagger. A disadvantage of pipelined segmentation and POS-tagging is that POS-tag information, which is potentially useful for segmentation, is not used during the segmentation step. In addition, word segmentation errors are propagated to the POS-tagger, leading to lower quality of the overall segmented and

[1] The experiments were performed using the Zhang and Clark (2007) feature set and single-beam decoder, and our new way to decide the number of training iterations in this article. The single-beam results correspond to “Zhang 2007*” in Tables 4 and 5.

POS-tagged output. Joint word segmentation and POS-tagging is a method that addresses these problems. In Zhang and Clark (2008a) we proposed a joint word segmentor and POS-tagger using a multiple-beam decoder, and showed that it outperformed a pipelined baseline. We recently showed that comparable accuracies can be achieved by a single-beam decoder, which runs an order of magnitude faster (Zhang and Clark 2010). In this section, we describe our single-beam system using our general framework, and provide a detailed comparison with our multiple-beam and baseline systems of Zhang and Clark (2008a).

4.1 Instantiating the General Framework

Given an input sentence, our joint segmentor and POS-tagger builds an output incrementally, one character at a time. When a character is processed, it is either concatenated with the last word in the partially built output, or taken as a new word. In the latter case, a POS-tag is assigned to the new word. When more characters are concatenated to a word, the POS-tag of the word remains unchanged.

For the decoding problem, an agenda is used to keep B different candidates at each incremental step. Before decoding starts, the agenda is initialized with an empty sentence. When a character is processed, existing candidates are removed from the agenda and extended with the current character in all possible ways, and the B-best newly generated candidates are put back onto the agenda. After all the input characters have been processed, the highest-scored candidate from the agenda is taken as output.

Expressed as an instance of the generic algorithm in Figure 1, a state item is a pair $\langle S, Q\rangle$, with S being a segmented and tagged sentence and Q being a queue of the next incoming characters. STARTITEM(*joint_tagging*) contains an empty sentence and the whole input sentence as incoming characters; EXPAND(*candidate*, *joint_tagging*) pops the first character from the incoming queue, adds it to *candidate*, and assigns POS-tags in the aforementioned way to generate a set of new state items; and GOALTEST(*joint_tagging*, *best*) returns true if *best* contains a complete segmented and POS-tagged output and an empty queue.

The linear model from Section 2 is applied to score state items, differentiating partial words from full words in the aforementioned ways, and the model parameters are trained with the averaged perceptron. The features for a state item are extracted incrementally according to Equation (3), where for the ith character, $\Phi(\delta(y, i))$ is computed according to the feature templates in both Table 1, which are related to word segmentation, and Table 7, which are related to POS-tagging. During training, the early-update method of Collins and Roark (2004), as described in Section 2, is used. It ensures that state items on the beam are highly probable at each incremental step, and is crucial to the high accuracy given by a single-beam.

4.2 Pruning

We use several pruning methods from Zhang and Clark (2008a), most of which serve to improve the accuracy by removing irrelevant candidates from the beam. First, the system records the maximum number of characters that a word with a particular POS-tag can have. For example, from the Chinese Treebank that we used for our experiments, most POS are associated only with one- or two-character words. The only POS-tags that are seen with words over ten characters long are NN (noun), NR (proper noun), and CD (numbers). The maximum word length information is initialized as all ones, and updated according to each training example before it is processed.

Table 7
POS feature templates for the joint segmentor and POS-tagger.

	Feature template	when c_0 is
1	$w_{-1}t_{-1}$	separated
2	$t_{-1}t_0$	separated
3	$t_{-2}t_{-1}t_0$	separated
4	$w_{-1}t_0$	separated
5	$t_{-2}w_{-1}$	separated
6	$w_{-1}t_{-1}end(w_{-2})$	separated
7	$w_{-1}t_{-1}c_0$	separated
8	$c_{-2}c_{-1}c_0t_{-1}$, where $len(w_{-1}) = 1$	separated
9	c_0t_0	separated
10	$t_{-1}start(w_{-1})$	separated
11	t_0c_0	separated or appended
12	$c_0t_0start(w_0)$	appended
13	$ct_{-1}end(w_{-1})$, where $c \in w_{-1}$ and $c \neq end(w_{-1})$	separated
14	$c_0t_0cat(start(w_0))$	separated
15	$ct_{-1}cat(end(w_{-1}))$, where $c \in w_{-1}$ and $c \neq end(w_{-1})$	appended
16	$c_0t_0c_{-1}t_{-1}$	separated
17	$c_0t_0c_{-1}$	appended

w = word; c = character; t = POS-tag. The index of the current character is 0.

Second, a tag dictionary is used to record POS-tags associated with each word. During decoding, frequent words and words with "closed set" tags[2] are only assigned POS-tags according to the tag dictionary, while other words are assigned every POS-tag to make candidate outputs. Whether a word is a frequent word is decided by the number of times it has been seen in the training process. Denoting the number of times the most frequent word has been seen by M, a word is a frequent word if it has been seen more than $M/5,000 + 5$ times. The threshold value is taken from Zhang and Clark (2008a), and we did not adjust it during development. Word frequencies are initialized as zeros and updated according to each training example before it is processed; the tag dictionary is initialized as empty and updated according to each training example before it is processed.

Third, we make an additional record of the initial characters for words with "closed set" tags. During decoding, when the current character is added as the start of a new word, "closed set" tags are only assigned to the word if it is consistent with the record. This type of pruning is used in addition to the tag dictionary to prune invalid partial words, while the tag dictionary is used to prune complete words. The record for initial character and POS is initially empty, and updated according to each training example before it is processed.

Finally, at any decoding step, we group partial candidates that are generated by separating the current character as the start of a new word by the signature $p_0p_{-1}w_{-1}$, and keep only the best among those having the same $p_0p_{-1}w_{-1}$. The signature $p_0p_{-1}w_{-1}$ is decided by the feature templates we use: it can be shown that if two candidates *cand1* and *cand2* generated at the same step have the same signature, and the score of *cand1* is higher than the score of *cand2*, then at any future step, the highest scored candidate

[2] "Closed set" tags are the set of POS-tags which are only associated with a fixed set of words, according to the Penn Chinese Treebank specifications (Xia 2000).

generated from *cand1* will always have a higher score than the highest scored candidate generated from *cand2*.

From these four pruning methods, only the third was not used by our multiple-beam system (Zhang and Clark 2008a). This was designed to help keep likely partial words in the agenda and improve the accuracy, and does not give our system a speed advantage over our multiple-beam system.

4.3 Comparison with Multiple-Beam Search (Zhang and Clark 2008a)

Our system of Zhang and Clark (2008a) was based on the perceptron and a multiple-beam decoder. That decoder can be seen as a slower alternative of our decoder in this article, but one which explores a larger part of the search space. The comparison between our joint segmentation and tagging systems of Zhang and Clark (2008a) and this article is similar to the comparison between our segmentors in sections 3.5.1 and 3.1. In Zhang and Clark (2008a), we argued that the straightforward implementation of the single-beam decoder cannot give competitive accuracies to the multiple-beam decoder, and the main difficulties for a single-beam decoder are in the representing of partial words, and the handling of an exponentially large combined search space using one beam. In this section, we give a description of our system of Zhang and Clark (2008a), and discuss the reason we can achieve competitive accuracies using a single beam in this article.

4.3.1 The Multiple-Beam System of Zhang and Clark (2008a). The decoder of our multiple-beam system can be formulated as an instance of the multiple-beam decoder described in Section 3.5.1.

Similar to the multiple-beam search decoder for word segmentation, the decoder compares candidates only with complete tagged words, and enables the size of the search space to scale with the input size. A set of state items is kept for each character to record possible segmented and POS-tagged sentences ending with the character. Just as with the single-beam decoder, the input sentence is processed incrementally. However, when a character is processed, the number of previously built state items is increased from B to kB, where B is the beam-size and k is the number of characters that have been processed. Moreover, partial candidates ending with any previous character are available. The decoder thus generates all possible tagged words ending with the current character, concatenating each with existing partial sentences ending immediately before the word, and putting the resulting sentence onto the agenda. After the character is processed, the B-best items in the agenda are kept as the corresponding state items for the character, and the agenda is cleared for the next character. All input characters are processed in the same way, and the final output is the best state item for the last character.

To instantiate the generic multiple-beam algorithm in Figure 4 for joint segmentation and POS-tagging, STARTITEM(*joint_tagging*) consists of an empty sentence S and a queue Q containing the full input sentence; EXPAND(*candidate*, k, *joint_tagging*) pops characters on the incoming queue from the front until the kth character (k is the index in the input sentence rather than the queue itself), appending them as a new word to *candidate*, and assigning to the new word all possible POS-tags to generate a set of new items; and GOALTEST(*joint_tagging*, *best*) returns true if *best* consists of a complete segmented and POS-tagged sentence and an empty incoming queue.

The linear model from Section 2 is again applied directly to score state items, and the model parameters are trained with the averaged perceptron algorithm. The features for a state item are extracted according to the union of the feature templates for the baseline segmentor and the baseline POS-tagger.

4.3.2 Discussion. An important problem that we solve for a single-beam decoder for the global model is the handling of partial words. As we pointed out in Zhang and Clark (2008a), it is very difficult to score partial words properly when they are compared with full words, although such comparison is necessary for incremental decoding with a single-beam. To allow comparisons with full words, partial words can either be treated as full words, or handled differently.

We showed in Zhang and Clark (2008a) that a naive single-beam decoder which treats partial words in the same way as full words failed to give a competitive accuracy. An important reason for the low accuracy is over-segmentation during beam-search. Consider the three characters 自来水 (tap water). The first two characters do not make sense when put together as a single word. Rather, when treated as two single-character words, they can make sense in a sentence such as 请 (please) 自 (self) 来 (come) 取 (take). Therefore, when using single-beam search to process 自来水 (tap water), the two-character word candidate 自来 is likely to have been thrown off the agenda before the third character 水 is considered, leading to an unrecoverable segmentation error.

This problem is even more severe for a joint segmentor and POS-tagger than for a pure word segmentor, because the POS-tags and POS-tag bigram of 自 and 来 further supports them being separated when 来 is considered. The multiple-beam search decoder we proposed in Zhang and Clark (2008a) can be seen as a means to ensure that the three characters 自来水 always have a chance to be considered as a single word. It explores candidate segmentations from the beginning of the sentence until each character, and avoids the problem of processing partial words by considering only full words. However, because it explores a larger part of the search space than a single-beam decoder, its time complexity is correspondingly higher.

In our single-beam system, we treat partial words differently from full words, so that in the previous example, the decoder can take the first two characters in 自来水 (tap water) as a partial word, and keep it in the beam before the third character is processed. One challenge is the representation of POS-tags for partial words. The POS of a partial word is undefined without the corresponding full word information. Though a partial word can make sense with a particular POS-tag when it is treated as a complete word, this POS-tag is not necessarily the POS of the full word which contains the partial word. Take the three-character sequence 下雨天 as an example. The first character 下 represents a single-character word 'below', for which the POS can be LC or VV. The first two characters 下雨 represent a two-character word 'rain', for which the POS can be VV. Moreover, all three characters when put together make the word 'rainy day', for which the POS is NN. As discussed earlier, assigning POS tags to partial words as if they were full words leads to low accuracy.

An obvious solution to this problem is not to assign a POS to a partial word until it becomes a full word. However, lack of POS information for partial words makes them less competitive compared to full words in the beam, because the scores of full words are further supported by POS and POS *n*-gram information. Therefore, not assigning POS to partial words potentially leads to over segmentation. In our experiments, this method did not give comparable accuracies to our multiple-beam system.

We take a different approach, and assign a POS-tag to a partial word when its first character is separated from the final character of the previous word. When more characters are appended to a partial word, the POS is not changed. The idea is to use the POS of a partial word as the predicted POS of the full word it will become. Possible predictions are made with the first character of the word, and the likely ones will be kept in the beam for the next processing steps. For example, with the three characters 下雨天, we try to keep two partial words (besides full words) in the beam when the first word 下 is processed, with the POS being VV and NN, respectively. The first POS predicts the two-character word 下雨, and the second the three-character word 下雨天. Now when the second character is processed, we still need to maintain the possible POS NN in the agenda, which predicts the three-character word 下雨天.

We show that the mechanism of predicting the POS at the first character gives competitive accuracy. This mechanism can be justified theoretically. Unlike alphabetical languages, each Chinese character represents some specific meanings. Given a character, it is natural for a human speaker to know immediately what types of words it can start. This allows the knowledge of possible POS-tags of words that a character can start, using information about the character from the training data. Moreover, the POS of the previous words to the current word are also useful in deciding possible POS for the word.[3]

The mechanism of first-character decision of POS also boosts the efficiency, because the enumeration of POS is unnecessary when a character is appended to the end of an existing word. As a result, the complexity of each processing step is reduced by half compared to a method without POS prediction.

Finally, an intuitive way to represent the status of a partial word is using a flag explicitly, which means an early decision of the segmentation of the next incoming character. We take a simpler alternative approach, and treat every word as a partial word until the next incoming character is separated from the last character of this word. Before a word is confirmed as a full word, we only apply to it features that represent its current partial status, such as character bigrams, its starting character, its part-of-speech, and so forth. Full word features, including the first and last characters of a word, are applied immediately after a word is confirmed as complete.

An important component for our proposed system is the training process, which needs to ensure that the model scores a partial word with predicted POS properly. We apply the general framework and use the averaged perceptron for training, together with the "early update" mechanism.

4.4 Experiments

We performed two sets of experiments, using the Chinese Treebank 4 and 5, respectively. In the first set of experiments, CTB4 was separated into two parts: CTB3 (420K characters in 150K words/$10,364$ sentences) was used for the final 10-fold cross validation, and the rest (240K characters in 150K words/$4,798$ sentences) was used as training and test data for development. The second set of experiments was performed to compare with relevant systems on CTB5 data.

The standard F-scores are used to measure both the word segmentation accuracy and the overall segmentation and tagging accuracy, where the overall accuracy is

$$JF = 2pr/(p+r)$$

with the precision p being the percentage of correctly segmented and tagged words in the decoder output, and the recall r being the percentage of gold-standard tagged

[3] The next incoming characters are also a useful source of information for predicting the POS. However, our system achieved competitive accuracy compared to our multiple-beam system without such character lookahead features.

words that are correctly identified by the decoder. For direct comparison with Ng and Low (2004), the POS-tagging accuracy is also calculated by the percentage of correct tags on each character.

4.4.1 Development Experiments. Our development data consists of 150K words in 4,798 sentences. Eighty percent (80%) of the data were randomly chosen as the development training data, and the rest were used as the development test data. Our development tests were mainly used to decide the size of the beam, the number of training iterations, and to observe the effect of early update.

Figure 6 shows the accuracy curves for joint segmentation and POS-tagging by the number of training iterations, using different beam sizes. With the size of the beam increasing from 1 to 32, the accuracies generally increase, although the amount of increase becomes small when the size of the beam becomes 16. After the tenth iteration, a beam size of 32 does not always give better accuracies than a beam size of 16. We therefore chose 16 as the size of the beam for our system.

The testing times for each beam size between 1 and 32 are 7.16 sec, 11.90 sec, 18.42 sec, 27.82 sec, 46.77 sec, and 89.21 sec, respectively. The corresponding speeds in the number of sentences per second are 111.45, 67.06, 43.32, 28.68, 17.06 and 8.95, respectively.

Figure 6 also shows that the accuracy increases with an increased number of training iterations, but the amount of increase becomes small after the 25th iteration. We chose 29 as the number of iterations to train our system.

The effect of early update: We compare the accuracies by early update and normal perceptron training. In the normal perceptron training case, the system reached the best performance at the 22nd iteration, with a segmentation F-score of 90.58% and joint

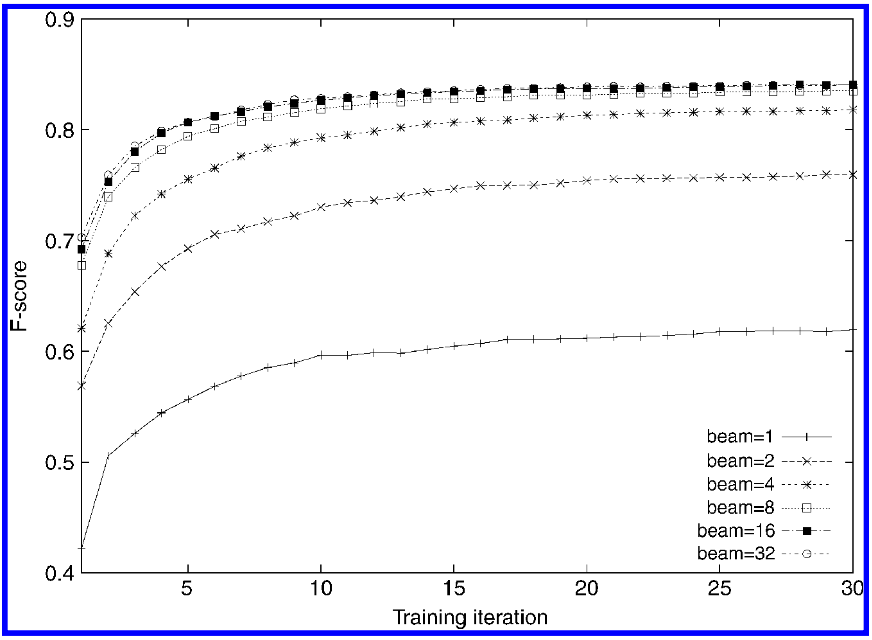

Figure 6
The influence of beam-sizes, and the convergence of the perceptron for the joint segmentor and POS-tagger.

Table 8
The accuracies of joint segmentation and POS-tagging by 10-fold cross validation.

	Baseline (pipeline)			Joint single-beam			Joint multiple-beam		
#	*SF*	*JF*	*JA*	*SF*	*JF*	*JA*	*SF*	*JF*	*JA*
Av.	95.2	90.3	92.2	95.8	91.4	93.0	95.9	91.3	93.0

SF = segmentation F-score; *JF* = overall segmentation and POS-tagging F-score; *JA* = tagging accuracy by character.

F-score of 83.38%. When using early update, the algorithm reached the best accuracy at the 30th training iteration, obtaining a segmentation F-score of 91.14% and a joint F-score of 84.06%.

4.4.2 Cross-Validation Results. Ten-fold cross validation is performed to test the accuracy of the joint word segmentor and POS-tagger, and to make comparisons with existing models in the literature. Following Ng and Low (2004), we partition the sentences in CTB3, ordered by sentence ID, into 10 groups evenly. In the *n*th test, the *n*th group is used as the testing data.

Table 8 shows the cross-validation accuracies of the pipeline baseline system and the joint system using single and multiple beam decoders. SF, JF and JA represent segmentation F-score, tagging F-score, and tagging accuracy, respectively. The joint segmentor and tagger systems outperformed the baseline consistently, while the single beam-search decoder in this article gave comparable accuracies to our multiple-beam algorithm of Zhang and Clark (2008a).

Speed comparisons of the three systems using the same 10-fold cross-validation are shown in Table 9. TE, ML, and SP represents the testing time (seconds), model loading time (seconds), and speed (number of sentences per second), respectively. Speed is calculated as number of test sentences divided by the test time (excluding model loading). For the baseline system, test time and model loading time for both the segmentor and the POS-tagger are recorded. The joint system using a single beam decoder was over 10 times faster than the multiple-beam system, and the baseline system was more than three times as fast as the single-beam joint system. These tests were performed on a Mac OSX platform with a 2.13GHz CPU and a gcc 4.0.1 compiler.

Table 10 shows the overall accuracies of the baseline and joint systems, and compares them to two relevant models in the literature. The accuracy of each model is

Table 9
The speeds of joint word segmentation and POS-tagging by 10-fold cross validation.

	Baseline (pipeline)			Joint single-beam			Joint multiple-beam		
#	*TE (s+p=total)*	*ML (s+p=total)*	*SP*	*TE*	*ML*	*SP*	*TE*	*ML*	*SP*
Av.	8.8+10.4=19.2	2.9+3.7=6.6	82.2	58.6	12.1	22.4	575.0	9.5	1.9

TE = testing time (seconds); *ML* = model loading time (seconds); *SP* = speed by the number of sentences per second, excluding loading time; *(s)* = segmentation in baseline; *(p)* = POS-tagging in baseline.

Table 10
The comparison of overall accuracies of various joint segmentor and POS-taggers by 10-fold cross validation using CTB.

Model	*SF*	*JF*	*JA*
Baseline+ (Ng)	95.1	–	91.7
Joint+ (Ng)	95.2	–	91.9
Baseline+* (Shi)	95.85	91.67	–
Joint+* (Shi)	96.05	91.86	–
Baseline (ours)	95.20	90.33	92.17
Joint (our multiple-beam)	95.90	91.34	93.02
Joint (our single-beam)	95.84	91.37	93.01

SF = segmentation F-score; JF = joint segmentation and POS-tagging F-score; JA = tagging accuracy by character.
+ Knowledge about special characters.
* Knowledge from semantic net outside CTB.

shown in a row, where Ng represents the models from Ng and Low (2004), which applies a character tagging approach to perform word segmentation and POS-tagging simultaneously, and Shi represents the models from Shi and Wang (2007), which is a reranking system for segmentation and POS-tagging. These two models are described in more detail in the related work section. Each accuracy measure is shown in a column, including the segmentation F-score (*SF*), the overall tagging F-score (*JF*), and the tagging accuracy by character (*JA*). As can be seen from the table, our joint models achieved the largest improvement over the baseline, reducing the segmentation error by more than 14% and the overall tagging error by over 12%.

The overall tagging accuracy of our joint model was comparable to but less than the joint model of Shi and Wang (2007). Despite the higher accuracy improvement from the baseline, the joint system did not give higher overall accuracy. One possible reason is that Shi and Wang (2007) included knowledge about special characters and semantic knowledge from Web corpora (which may explain the higher baseline accuracy), whereas our system is completely data-driven. However, the comparison is indirect because our partitions of the CTB corpus are different. Shi and Wang (2007) also chunked the sentences before doing 10-fold cross validation, but used an uneven split. We chose to follow Ng and Low (2004) and split the sentences evenly to facilitate further comparison.

Compared with Ng and Low (2004), our baseline model gave slightly better accuracy, consistent with our previous observations about the word segmentors in Section 3. Due to the large accuracy gain from the baseline, our joint model performed much better.

In summary, when compared with existing joint word segmentation and POS-tagging systems using cross-validation tests, our proposed model achieved the best accuracy boost from the pipelined baseline, and competitive overall accuracy. Our system based on the general framework of this article gave comparable accuracies to our multiple-beam system in Zhang and Clark (2008a), and a speed that is over an order of magnitude higher than the multiple-beam algorithm.

4.4.3 Test Results Using CTB5. We follow Kruengkrai et al. (2009) and split the CTB5 into training, development testing, and testing sets, as shown in Table 11. The data are used

Table 11
Training, development, and test data from CTB5 for joint word segmentation and POS-tagging.

	Sections	Sentences	Words
Training	1–270, 400–931, 1001–1151	18,085	493,892
Dev	301–325	350	6,821
Test	271–300	348	8,008

to compare the accuracies of our joint system with models in the literature, and to draw the speed/accuracy tradeoff graph.

Kruengkrai et al. (2009) made use of character type knowledge for spaces, numerals, symbols, alphabets, and Chinese and other characters. In the previous experiments, our system did not use any knowledge beyond the training data. To make the comparison fairer, we included knowledge of English letters and Arabic numbers in this experiment. During both training and decoding, English letters and Arabic numbers are segmented using rules, treating consecutive English letters or Arabic numbers as a single word.

The results are shown in Table 12, where row N07 refers to the model of Nakagawa and Uchimoto (2007), rows J08a and J08b refer to the models of Jiang et al. (2008) and Jiang, Mi, and Liu (2008), and row K09 refers to the models of Kruengkrai et al. (2009). Columns SF and JF refer to segmentation and joint segmentation and tagging accuracies, respectively. Our system gave comparable accuracies to these recent works, obtaining the best (same as the error-driven version of K09) joint F-score.

The accuracy/speed tradeoff graphs for the joint segmentor and POS-taggers, together with the baseline pipeline system, are shown in Figure 7. For each point in each curve, the development test data were used to decide the number of training iterations, and then the speed and accuracy were measured using test data. No character knowledge is used in any system. The baseline curve was drawn with $B = 16$ for the baseline segmentor, because the baseline segmentation accuracy did not improve beyond $B = 16$ in our experiments. Each point in this curve corresponds to a different beam size of the baseline POS-tagger, which are 2, 4, 8, 16, 32, 64, 128, and 256, respectively, from right to left.

When the speed is over 2.5 thousand characters per second, the baseline system performed better than the joint single-beam and multiple-beam systems, due to the higher segmentation accuracy brought by the fixed beam segmentor. However, as the

Table 12
Accuracy comparisons between various joint segmentors and POS-taggers on CTB5.

	SF	JF
K09 (error-driven)	97.87	93.67
our system	97.78	93.67
Zhang 2008	97.82	93.62
K09 (baseline)	97.79	93.60
J08a	97.85	93.41
J08b	97.74	93.37
N07	97.83	93.32

SF = segmentation F-score; JF = joint segmentation and POS-tagging F-score.

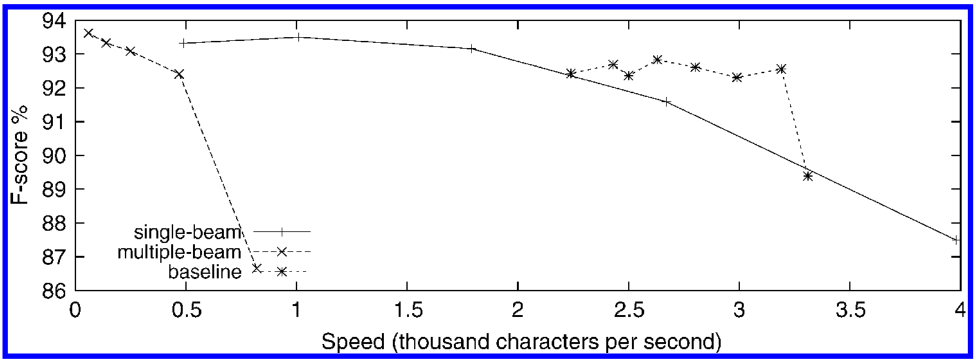

Figure 7
The accuracy/speed tradeoff graph for the joint segmentor and POS-taggers and the two-stage baseline.

tagger beam size further increased, the accuracy of the baseline system did not improve. The highest F-score of the baseline (92.83%) was achieved when the beam size of the baseline POS-tagger was 32. In fact, we have shown in Section 3.5 that the accuracy of the baseline segmentor was similar to using a Viterbi decoder when the beam size was 16. This was also true for the baseline POS-tagger, according to our experiments. Therefore, though being the most accurate when the speed is high, the baseline system reaches the highest F-score at 2.63 thousand characters per second, and cannot further improve the accuracy on this curve.

The points in the single-beam curve correspond to beam sizes of 2, 4, 8, 16, and 32, respectively, from right to left. When the speed is roughly between 0.5 and 2.0 thousand characters per second, the single-beam system gave the best F-score. This is because the multiple-beam system did not reach such high speeds, and the baseline system could not produce a higher accuracy than 92.83%. The single-beam joint system gave the highest accuracy of 93.50% when the beam size was 16 and the speed was 1.01 thousand characters per second, and the F-score dropped slightly when the beam further increased to 32.

The points in the multiple-beam curve correspond to beam sizes of 1, 2, 4, 8, and 16, respectively, from right to left. The multiple-beam system gave the highest F-score of 93.62% when the beam size was 16, but the speed was down to 0.06 thousand sentences per second.

4.5 Related Work

Ng and Low (2004) mapped the joint segmentation and POS-tagging task into a single character sequence tagging problem. Two types of tags are assigned to each character to represent its segmentation and POS. For example, the tag b_NN indicates a character at the beginning of a noun, and the tag e_VV indicates a character at the end of a verb. Using this method, POS features are allowed to interact with segmentation. Because tagging is restricted to characters, the search space is reduced to $O((4T)^n)$, where 4 is the number of segmentation tags and T is the size of the tag set. Beam-search decoding is effective with a small beam-size. However, the disadvantage of this model is the difficulty of incorporating whole word information into POS-tagging. For example, the standard word + POS-tag feature is not applicable.

Shi and Wang (2007) introduced POS information into segmentation by reranking. *B*-best segmentation outputs are passed to a separately-trained POS-tagger, and the best output is selected using the overall POS-segmentation probability score. In this system, the decoding for word segmentation and POS-tagging are still performed separately, and exact inference for both is possible. However, the interaction between POS and segmentation is restricted by reranking: POS information is used to improve segmentation only for the *B* segmentor outputs. In comparison to the two systems described here, our joint system does not impose any hard constraints on the interaction between segmentation and POS information.

Jiang, Mi, and Liu (2008) proposed a reranking system that builds a pruned word lattice instead of generating a *B*-best list by the segmentor. The advantage of this reranking method compared to Shi and Wang's (2007) method is that more ambiguity is kept in the reranking stage. The reranking algorithm used a similar model to our joint segmentor and POS-tagger.

Nakagawa and Uchimoto (2007) proposed a hybrid model for word segmentation and POS tagging using an HMM-based approach. Word information is used to process known words, and character information is used for unknown words in a similar way to Ng and Low (2004). In comparison, our model handles character and word information simultaneously in a single perceptron model. Recently, Kruengkrai et al. (2009) developed this hybrid model further by scoring characters and words in the same model. Their idea is similar to our joint segmentor and POS-tagger in Zhang and Clark (2008a).

5. Dependency Parsing

Dependency parsing is the problem of producing the syntactic structure for an input sentence according to dependency grammar. The output structure of a dependency parser is called a **dependency graph**; in this article, only dependency trees are considered. As can be seen from Figure 8, a dependency tree consists of a set of vertices and directed arcs. Each arc represents the relationship between a pair of words in the sentence; it points from the head word to its modifier. For example, in the arc between the word 我 (I) and the word 喜欢 (like), 喜欢 (like) is the head word and 我 (I) is the subject that modifies 喜欢 (like); in the arc between the word 喜欢 (like) and the word 读书 (reading), 喜欢 (like) is the head word and 读书 (reading) is the object that modifies 喜欢 (like). In a dependency tree, there is only one word that does not modify any other word, and it is called the **head word** of the sentence. The other words each modify exactly one word, and no word can modify itself.

A dependency tree is called **projective** if there are no crossing arcs when the sentence is represented linearly, in word order. Though almost all languages are non-projective to some degree, the majority of sentences in most languages are projective. In

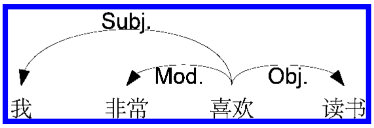

Figure 8
An example Chinese dependency tree.

the CoNLL shared tasks on dependency parsing (Buchholz and Marsi 2006; Nivre et al. 2007) most data sets contain only 1–2% non-projective arcs, and projective dependency parsing models can give reasonable accuracy in these tasks (Carreras, Surdeanu, and Marquez 2006). Although non-projective dependency parsing can be solved directly by using a different model from projective dependency parsing (McDonald et al. 2005), it can also be solved using a projective parsing model, with the help of a reversible transformation procedure between non-projective and projective dependency trees (Nivre et al. 2006). In this article we focus on projective dependency parsing.

An unlabeled dependency tree is a dependency tree without dependency labels such as Subj and Obj in Figure 8. The same techniques used by unlabeled dependency parsers can be applied to labeled dependency parsing. For example, a shift-reduce unlabeled dependency parser can be extended to perform labeled dependency parsing by splitting a single reduce action into a set of reduce actions each associated with a dependency label. Here we focus on unlabeled dependency parsing.

Graph-based (McDonald, Crammer, and Pereira 2005; Carreras, Surdeanu, and Marquez 2006; McDonald and Pereira 2006) and **transition-based** (Yamada and Matsumoto 2003; Nivre et al. 2006) parsing algorithms offer two different approaches to data-driven dependency parsing. Given an input sentence, a graph-based algorithm finds the highest scoring parse tree from all possible outputs, scoring each complete tree, while a transition-based algorithm builds a parse by a sequence of actions, scoring each action individually. Although graph-based and transition-based parsers can be differentiated in various ways, we prefer to think in terms of the features used in the two approaches as the differentiating factor.

In Zhang and Clark (2008b) we proposed a graph-based parser and a transition-based parser using the generalized perceptron and beam-search, showing that beam-search is a competitive choice for both graph-based and transition-based dependency parsing. In the same paper we used a single discriminative model to combine graph-based and transition-based parsing, showing that the combined parser outperforms both graph-based and transition-based parsers individually. All three parsers can be expressed by our general framework. Here we describe the transition-based and combined parsers, which share the same decoding process.

5.1 Instantiating the General Framework

Our dependency parser uses the incremental parsing process of MaltParser (Nivre et al. 2006), which is characterized by the use of a stack and four transition actions: SHIFT, ARCRIGHT, ARCLEFT, and REDUCE. An input sentence is formed into a queue of incoming words and processed from left to right. Initially empty, the stack is used throughout the parsing process to store unfinished words, which are the words before the current word that may still be linked with the current or a future word.

The SHIFT action pops the first word from the queue and pushes it onto the stack. The ARCRIGHT action pops the first word from the queue, adds a dependency link from the stack top to the word (i.e., the stack top becomes the parent of the word), and pushes the word onto the stack. The ARCLEFT action adds a dependency link from the first word on the queue to the stack top, and pops the stack. The REDUCE action pops the stack. Among the four transition actions, SHIFT and ARCRIGHT push a word onto the stack, and ARCLEFT and REDUCE pop the stack; SHIFT and ARCRIGHT read the next input word, and ARCLEFT and ARCRIGHT add a link to the output.

The initial state contains an empty stack and the whole input sentence as incoming words. The final state contains a stack holding only the head word of the sentence and an empty queue, with the input words having all been processed. The incremental parsing process starts from the initial state, and builds a candidate parse tree by repeatedly applying one transition action out of the four, until the final state is reached.

For the decoding problem, an agenda is used to find the output parse tree from different possible candidates. Starting with an initial state item, a set of candidate state items is used to generate new state items for each step. At each step, each existing state item is extended by applying all applicable actions from the four, and the generated items are put onto the agenda. After each step, the best B items are taken from the agenda to generate new state items for the next step, and the agenda is cleared. After all incoming words have been processed, the corresponding parse of the top item from the agenda is taken as the output.

The decoding process is an instance of the generalized algorithm in Figure 1. A state item is a pair $\langle S, Q\rangle$, where S represents the stack with the partial parse, and Q represents the queue of incoming words. The initial state item STARTITEM(*dependency_parsing*) consists of an empty stack, and an incoming queue of the whole input. The function EXPAND(*candidate, dependency_parsing*) applies all possible actions to *candidate* and generates a set of new state items. GOALTEST(*dependency_parsing, best*) returns true if *best* has reached the final state, and false otherwise.

The score of a parse tree is computed by the global linear model in Equation (2), where the parameter vector $\vec{w}$ for the model is computed by the averaged perceptron training algorithm described in Section 2.2. During training of the dependency parser, the early-update strategy of Collins and Roark (2004) is used. The intuition is to improve learning by avoiding irrelevant information, as discussed earlier: When all the items in the current agenda are incorrect, further parsing steps will be irrelevant because the correct partial output no longer exists in the candidate ranking.

Both the transition-based and the combined parsers use this training and decoding framework. The main difference between the two parsers is in the definition of the feature templates. Whereas the transition-based parser uses only transition-based features, the combined parser applies features from the graph-based parser in addition to transition-based features. Table 13 shows the templates for transition-based features. Individual features for a parse are extracted from each transition action that is used to build the parse, by first instantiating the templates according to the local context, and then pairing the instantiated template with the transition action. Shown in Figure 9, the contextual information consists of the top of the stack (ST), the parent (STP) of ST, the leftmost (STLC) and rightmost child (STRC) of ST, the current word (N0), the next three

Table 13
Transition-based feature templates for the dependency parser.

1	stack top	STwt; STw; STt
2	current word	N0wt; N0w; N0t
3	next word	N1wt; N1w; N1t
4	ST and N0	STwtN0wt; STwtN0w; STwN0wt; STwtN0t; STtN0wt; STwN0w; STtN0t
5	POS bigram	N0tN1t
6	POS trigrams	N0tN1tN2t; STtN0tN1t; STPtSTtN0t; STtSTLCtN0t; STtSTRCtN0t; STtN0tN0LCt
7	N0 word	N0wN1tN2t; STtN0wN1t; STPtSTtN0w; STtSTLCtN0w; STtSTRCtN0w; STtN0wN0LCt

w = word; t = POS-tag.

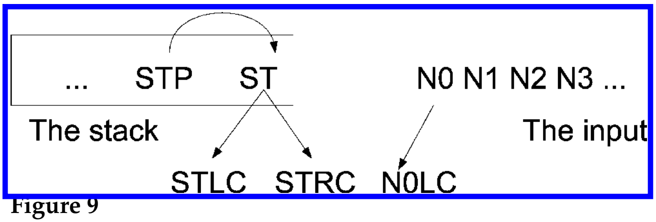

Figure 9
Transition-based feature context for the dependency parser.

words from the input (N1, N2, N3), and the leftmost child of N0 (N0LC). Word and POS information from the context are manually combined, and the combination of feature templates is decided by development tests.

Table 14 shows the templates used to extract graph-based features from partial parses. Templates 1–6 are taken from MSTParser (McDonald and Pereira 2006), a second-order graph-based parser. They are defined in the context of a word, its parent and its sibling. To give more templates, features from templates 1–5 are also conjoined with the arc direction and distance, whereas features from template 6 are also conjoined with the direction and distance between the child and its sibling. Here "distance" refers to the difference between word indexes. Templates 7–8 are two extra feature templates that capture information about grandchildren and arity (i.e., the number of children to the left or right). They are difficult to include in an efficient dynamic-programming decoder, but easy to include in a beam-search decoder. During decoding, the graph-based feature templates are instantiated at the earliest possible situation. For example, the first-order arc and second-order sibling feature templates are instantiated when ARCLEFT or ARCRIGHT is performed, with sibling information for the newly added link. The arity features are added as soon as the left or right arity of a word becomes fixed, the left arity templates being instantiated when ARCLEFT or SHIFT is performed, with the right arity templates being instantiated when ARCLEFT or REDUCE is performed. Similarly, the leftmost grandchild features are instantiated when ARCLEFT or ARCRIGHT is performed, and the rightmost grandchild features are instantiated when ARCLEFT or REDUCE is performed.

Table 14
Graph-based feature templates for the dependency parser.

1	Parent word (P)	Pw; Pt; Pwt
2	Child word (C)	Cw; Ct; Cwt
3	P and C	PwtCwt; PwtCw; PwCwt; PwtCt; PtCwt; PwCw; PtCt
4	Tag Bt between P, C	PtBtCt
5	Neighbor words of P and C, left (L) and right (R)	PtPLtCtCLt; PtPLtCtCRt; PtPRtCtCLt; PtPRtCtCRt; PtPLtCLt; PtPLtCRt; PtPRtCLt; PtPRtCRt; PLtCtCLt; PLtCtCRt; PRtCtCLt; PRtCtCRt; PtCtCLt; PtCtCRt; PtPLtCt; PtPRtCt
6	sibling (S) of C	CwSw; CtSt; CwSt; CtSw; PtCtSt
7	leftmost (CLC) and rightmost (CRC) children of C	PtCtCLCt; PtCtCRCt
8	left (la) and right (ra) arity of P	Ptla; Ptra; Pwtla; Pwtra

w = word; t = POS-tag.

5.1.1 The Comparability of State Items. Our dependency parsers are based on the incremental shift-reduce parsing process. During decoding, state items built with the same number of transition actions are put onto the agenda and compared with each other. To build any full parse tree, each word in the input sentence must be put onto the stack once, and each word except the root of the sentence must be popped off the stack once. Because the four transition actions are either stack-pushing or stack-popping, each full parse must be built with $2n - 1$ transition actions, where n denotes the size of the input. Therefore, the decoding process consists of $2n - 1$ steps, and in each step all state items in the agenda have been built using the same number of actions. Our experiments showed that start-of-the-art accuracy can be achieved by this intuitive method of candidate comparison.

5.2 Experiments for English

We used Penn Treebank 3 for our experiments, which was separated into the training, development, and test sets in the same way as McDonald, Crammer, and Pereira (2005), shown in Table 15. Bracketed sentences from the Treebank were translated into dependency structures using the head-finding rules from Yamada and Matsumoto (2003).

Before parsing, POS-tags are assigned to the input sentence using our baseline POS-tagger of Zhang and Clark (2008a), which can be seen as the perceptron tagger of Collins (2002) with beam-search. Like McDonald, Crammer, and Pereira (2005), we evaluated the parsing accuracy by the precision of lexical heads (the percentage of input words, excluding punctuation, that have been assigned the correct parent) and by the percentage of complete matches, in which all words excluding punctuation have been assigned the correct parent.

A set of development tests, including the convergence of the perceptron, can be found in Zhang and Clark (2008a). In this article, we report only the final test accuracies and a set of additional speed/accuracy tradeoff results. The accuracies of our transition-based and combined parsers on English data are shown together with other systems in Table 16. In the table, each row represents a parsing model. Rows Yamada 2003 and MSTParser represent Yamada and Matsumoto (2003), and MSTParser with templates 1–6 from Table 14 (McDonald and Pereira 2006), respectively. Rows Transition and Combined represent our pure transition-based and combined parsers, respectively. Row Huang 2010 shows the recent work of Huang and Sagae (2010), which applies dynamic-programming packing to transition-based dependency parsing. Rows Koo 2008 and Chen 2009 represent the models of Koo, Carreras, and Collins (2008) and Chen et al. (2009), which perform semi-supervised learning by word-clustering and self-training, respectively. Columns Word and Complete show the precision of lexical

Table 15
The training, development, and test data for English dependency parsing.

	Sections	Sentences	Words
Training	2–21	39,832	950,028
Development	22	1,700	40,117
Test	23	2,416	56,684

Table 16
Accuracy comparisons between various dependency parsers on English data.

	Word	Complete
Yamada 2003	90.3	38.4
Transition	91.4	41.8
MSTParser	91.5	42.1
Combined	92.1	45.4
Huang 2010	92.1	–
Koo 2008	93.2	–
Chen 2009	93.2	47.2

See text for details.

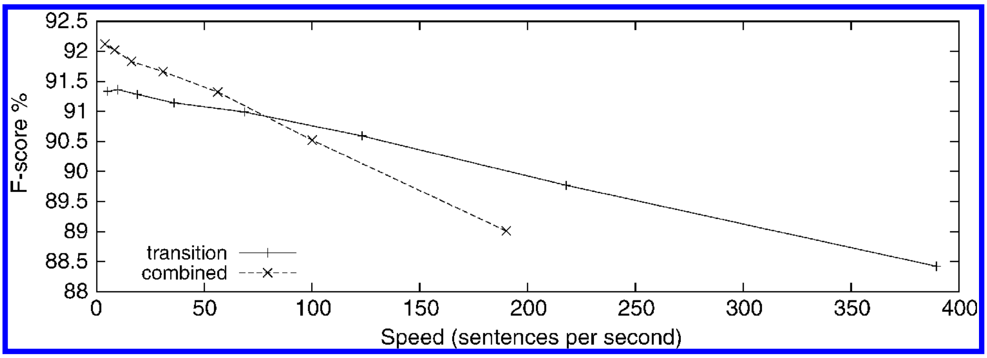

Figure 10
The accuracy/speed tradeoff graph for the transition-based and combined dependency parsers.

heads and complete matches, respectively. The combined parser achieved 92.1% per-word accuracy, which was significantly higher than the pure transition-based parser.[4] These results showed the effectiveness of utilizing both graph-based and transition-based information in a single model. Represented by features that are not available in a pure transition-based system, graph-based information helped the combined parser to achieve higher accuracy.

As in the previous sections, we plot the speed/accuracy tradeoff for the transition-based and combined parsers. For each point in each curve in Figure 10, we run the development test to decide the number of training iterations, and read the speed and accuracy from the final test. Each point in the transition curve corresponds to $B = 1, 2, 4,$ 8, 16, 32, 64, and 128, respectively, and each point in the combined curve corresponds to $B = 1, 2, 4$, 8, 16, 32, and 64, respectively. When the speed was above 100 sentences per second, the pure transition-based parser outperformed the combined parser with the same speed. However, as the size of the beam increases, the accuracy of the combined parser increased more rapidly. The combined parser gave higher accuracies at the same speed when the speed was below 50 sentences per second. The accuracy of the pure

[4] In Zhang and Clark (2008a) we showed that the combined parser also significantly outperformed the graph-based parser.

Table 17
Training, development, and test data for Chinese dependency parsing.

	Sections	Sentences	Words
Training	001–815 1,001–1,136	16,118	437,859
Dev	886–931 1,148–1,151	804	20,453
Test	816–885 1,137–1,147	1,915	50,319

transition parser did not increase beyond $B = 64$. Our experiments were performed on a Linux platform with a 2.0GHz CPU and a gcc 4.0.1 compiler.

When $B = 1$, the transition-based parser was similar to MaltParser, because our transition-based parser is built upon the arc-each transition process of MaltParser, and uses similar feature sets. The main difference is that our transition-based parser performs global training using the perceptron algorithm, whereas MaltParser performs local training using a support vector machine (SVM) with a polynomial kernel. Because global training optimizes the model for full sequences of transition actions, a small beam can potentially have a negative impact on learning, and hence the overall performance.

5.3 Experiments for Chinese

We used the Penn Chinese Treebank 5 for our experiments. Following Duan, Zhao, and Xu (2007), we split the corpus into training, development, and test data as shown in Table 17. We used a set of head-finding rules to turn the bracketed sentences into dependency structures, and they can be found in Zhang and Clark (2008a). Like Duan, Zhao, and Xu (2007), we used gold-standard POS-tags for the input. The parsing accuracy was evaluated by the percentage of non-root words that have been assigned the correct head, the percentage of correctly identified root words, and the percentage of complete matches, all excluding punctuation.

The accuracies are shown in Table 18. Rows Transition and Combined show our models in the same way as for the English experiments from Section 5.2. Row Duan 2007 represents the transition-based model from Duan, Zhao, and Xu (2007), which applied beam-search to the deterministic model from Yamada and Matsumoto (2003). Row

Table 18
Test accuracies of various dependency parsers on CTB5 data.

	Non-root	Root	Complete
Duan 2007	84.36	73.70	32.70
Transition	84.69	76.73	32.79
Huang 2010	85.52	78.32	33.72
Combined	86.21	76.26	34.41

See text for details.

Huang 2010 represents the model of Huang and Sagae (2010), which applies dynamic-programming packing to transition-based parsing. The observations were similar to the English tests. The combined parser outperformed the transition-based parsers. It gave the best accuracy we are aware of for supervised dependency parsing using the CTB.

One last question we investigate for this article is the overall performance when the parser is pipelined with a POS-tagger, or with the joint segmentation and POS-tagging algorithm in Section 4, forming a complete pipeline for Chinese inputs. The results are shown in Table 19. For these experiments, we tune the pipelined POS-tagger and the joint segmentor and POS-tagger on the CTB5 corpus in Table 17, using the development test data to decide the number of training iterations and reporting the final test accuracy. The overall accuracy is calculated in F-score: Defining *nc* as the number of output words that have been correctly segmented and assigned the correctly segmented head word, *no* as the number of words in the output, and *nr* the number of words in the reference, precision is $p = nc/no$ and recall is $r = nc/nr$. When pipelined with a pure POS-tagger using gold-standard segmentation, the pipelined system gave 93.89% POS accuracy and 81.21% joint tagging and parsing F-score for non-root words. When combined with the joint segmentation and POS-tagging system, the segmentation F-score, joint segmentation and POS-tagging F-score were 95.00% and 90.17%, respectively, and the joint segmentation and parsing F-score for non-root words (excluding punctuations) was 75.09%, where the corresponding accuracy with gold-standard segmented and POS-tagged input was 86.21%, as shown in Table 18.

5.4 Related Work

MSTParser (McDonald and Pereira 2006) is a graph-based parser with an exhaustive search decoder, and MaltParser (Nivre et al. 2006) is a transition-based parser with a greedy search decoder. Representative of each method, MSTParser and MaltParser gave comparable accuracies in the CoNLL-X shared task (Buchholz and Marsi 2006). However, they make different types of errors, which can be seen as a reflection of their theoretical differences (McDonald and Nivre 2007). By combining graph-based and transition-based information, our dependency parser achieved higher accuracy than both graph-based and transition-based baselines. The combination of information is enabled by our general framework. Our global model does not impose any limitation on the kind of features that can be used, and therefore allows both graph-based and transition-based features. Moreover, beam-search frees the decoder from locality restrictions such as the "optimal sub-problem" requirement for dynamic-programming, which limits the range of features in order to achieve reasonable decoding speed. Compared to the greedy local search decoder for MaltParser, beam-search can reduce error propagation by keeping track of a set of candidate items.

Table 19
The combined segmentation, POS-tagging, and dependency parsing F-scores using different pipelined systems.

	Seg F	Tag F	Dep F
Gold seg tag	100.00	100.00	86.21
Gold seg auto tag	100.00	93.89	81.21
Auto seg tag	95.00	90.17	75.09

Our transition-based parser is based on the arc-eager parsing process of MaltParser (Nivre et al. 2006), although our parser is different from MaltParser in two aspects. First, we applied beam-search in decoding, which helps to prevent error propagation of local search. Johansson and Nugues (2007) also use beam search. Second, we use the perceptron to train whole sequences of transition actions globally, whereas MaltParser uses SVM to train each transition action locally. Our global training corresponds to beam-search decoding, which searches for a globally optimal sequence of transition actions rather than an optimal action at each step.

An existing method to combine multiple parsing algorithms is the ensemble approach (Sagae and Lavie 2006), which was reported to be useful in improving dependency parsing (Hall et al. 2007). A more recent approach (Nivre and McDonald 2008) combined MSTParser and MaltParser by using the output of one parser for features in the other in a stacking framework. Both Hall et al. (2007) and Nivre and McDonald (2008) can be seen as methods to combine separately defined models. In contrast, our parser combines two components in a single model, in which all parameters are trained consistently.

6. Phrase-Structure Parsing

Phrase-structure parsing is the problem of producing the syntactic structure of an input sentence according to a phrase-structure grammar. An example phrase-structure parse tree is shown in Figure 11. Similar to dependency parsing, dominant approaches to phrase-structure parsing include the transition-based method (Briscoe and Carroll 1993), which builds an output parse tree by choosing a series of transition actions such as SHIFT and REDUCE, and the graph-based method (Collins 1999; Charniak 2000), which explores the search space of possible parse trees to find the best output according to graph-based scores.

For English constituent parsing using the Penn Treebank, the best performing transition-based parser lags behind the current state-of-the-art (Sagae and Lavie 2005). However, for Chinese constituent parsing using the Chinese Treebank, Wang et al. (2006) have shown that a shift-reduce parser can give competitive accuracy scores together with high speeds by using an SVM to make a single decision at each point in the parsing process.

In Zhang and Clark (2009) we proposed a transition-based constituent parser for Chinese, which is based on the transition process of Wang et al. (2006). Rather than making a single decision at each processing step, our parser uses a global linear model

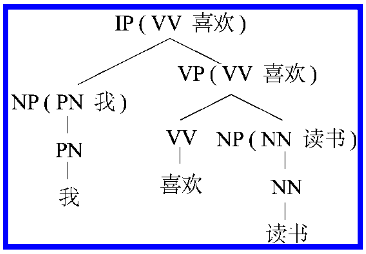

Figure 11
An example Chinese lexicalized phrase-structure parse tree.

and beam-search decoding, and achieved competitive accuracy. This phrase-structure parser can be expressed as an instance of our general framework.

6.1 Instantiating the General Framework

The incremental parsing process of our parser is based on the shift-reduce parsers of Sagae and Lavie (2005) and Wang et al. (2006), with slight modifications. The input is assumed to be segmented and POS-tagged, and the word–POS pairs waiting to be processed are stored in a queue. A stack holds the partial parse trees that are built during the parsing process. The output of this process is a binarized parse tree. The four types of action used to build a parse are:

- SHIFT, which pushes the next word-POS pair in the queue onto the stack.
- REDUCE–unary–X, which makes a new unary-branching node with label X; the stack is popped and the popped node becomes the child of the new node; the new node is pushed onto the stack.
- REDUCE–binary–{L/R}–X, which makes a new binary-branching node with label X; the stack is popped twice, with the first popped node becoming the right child of the new node and the second popped node becoming the left child; the new node is pushed onto the stack. The left (L) and right (R) versions of the rules indicate whether the head of the new node is to be taken from the left or right child.
- TERMINATE, which pops the root node off the stack and ends parsing. This action can only be applied when the input queue is empty, and the stack contains only one item. The popped node is taken as the output.

Defining the start state as the stack being empty and the queue containing the input sentence, and the final state as the state immediately after a TERMINATE action, the incremental process builds a parse tree by repeatedly applying actions from the start state until the final state is reached. Note that not all sequences of actions produce valid binarized trees. In the deterministic parser of Wang et al. (2006), the highest scoring action predicted by the classifier may prevent a valid binary tree from being built. In this case, Wang et al. simply return a partial parse consisting of all the subtrees on the stack. In our parser a set of restrictions is applied which guarantees a valid parse tree. For example, two simple restrictions are that a SHIFT action can only be applied if the queue of incoming words is non-empty, and the binary reduce actions can only be performed if the stack contains at least two nodes. Some of the restrictions are more complex than this; the full set can be found in Zhang and Clark (2009). Wang et al. (2006) give a detailed example showing how a segmented and POS-tagged sentence can be incrementally processed using the shift-reduce actions to produce a binary tree. We show this example in Figure 12.

For the decoding problem, our parser performs beam-search using an agenda to find the output parse from possible candidates. Initially containing a start state item, a set of state items is used to generate new state items for each processing step. At each step, each of the state items is extended using all applicable actions, generating a set of new state items, which are put onto the agenda. After each step, the B-best items from the agenda are taken for the generation of new state items in the next step. The same

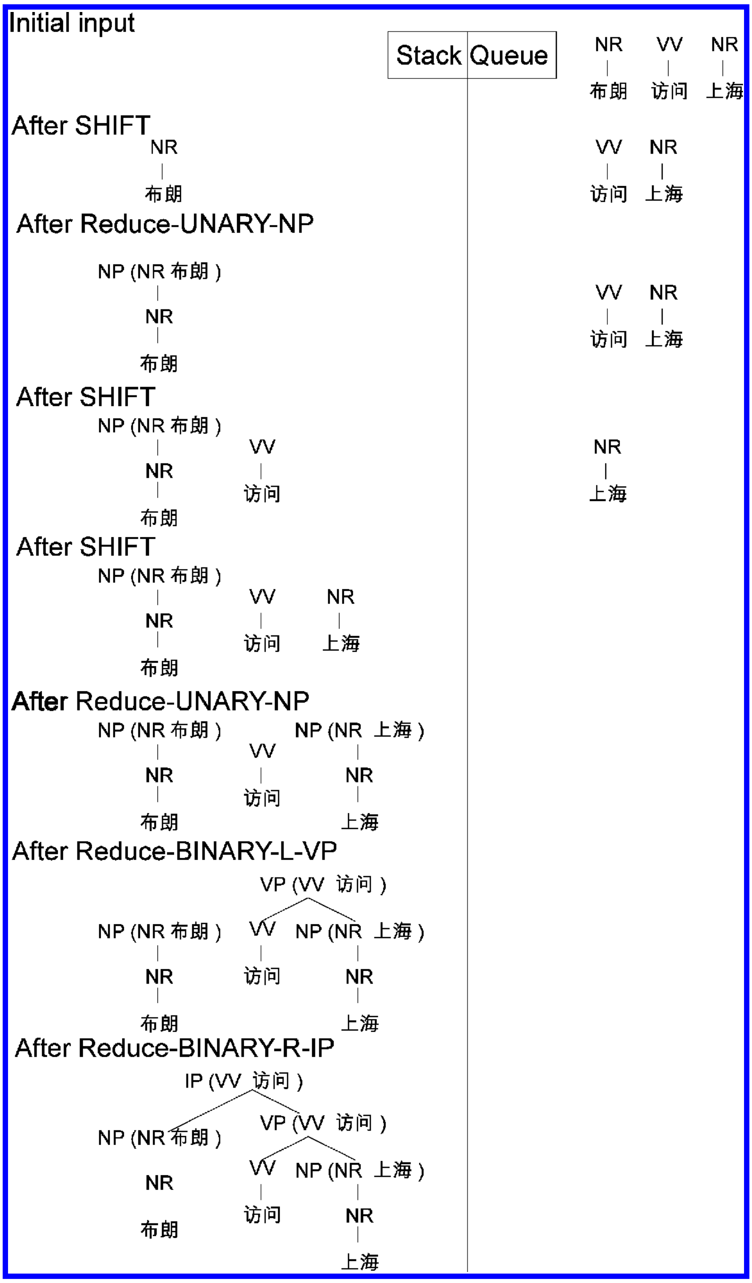

Figure 12
An example shift-reduce parsing process.

process repeats until the highest scored item from the agenda is in the final state, and it is taken as the final parse.

This decoding process can be seen as an instance of the generic algorithm in Figure 1. Here a state item is a pair $\langle S, Q \rangle$, where S represents the stack with partial parses, and Q represents the incoming queue. The initial state item StartItem(*constituent_parsing*) is the start state item, where the stack is empty and the queue contains the whole input sentence, the function EXPAND(*candidate, constituent_parsing*) extends *candidate* by using all applicable actions to generate a set of new state items, and GOALTEST (*constituent_parsing, best*) returns true if *best* reaches the final state, and false otherwise.

The score of a parse tree is computed by the global linear model in Equation (2), where the parameter vector $\vec{w}$ for the model is computed by the averaged perceptron training algorithm described in Section 2.2. As in the training of the dependency parser, the early-update strategy of Collins and Roark (2004) is used.

Table 20 shows the set of feature templates for the model. Individual features are generated from these templates by first instantiating a template with particular labels, words, and tags, and then pairing the instantiated template with a particular action. In the table, the symbols S_0, S_1, S_2, and S_3 represent the top four nodes on the stack, and the symbols N_0, N_1, N_2, and N_3 represent the first four words in the incoming queue. S_0L, S_0R, and S_0U represent the left and right child for binary branching S_0, and the single child for unary branching S_0, respectively; w represents the lexical head token for a node; and c represents the label for a node. When the corresponding node is a terminal, c represents its POS-tag, whereas when the corresponding node is a nonterminal, c represents its constituent label; t represents the POS-tag for a word.

The context $S_0, S_1, S_2, S_3, N_0, N_1, N_2, N_3$ for the feature templates is taken from Wang et al. (2006). However, Wang et al. (2006) used a polynomial kernel function with an SVM and did not manually create feature combinations. Because we used the linear perceptron algorithm we manually combined Unigram features into Bigram and Trigram features.

The Bracket row shows bracket-related features, which were inspired by Wang et al. (2006). Here brackets refer to left brackets including "（", "“", and "《" and right brackets including "）", "”", and "》". In the table, b represents the matching status of the last left bracket (if any) on the stack. It takes three different values: 1 (no matching right bracket has been pushed onto stack), 2 (a matching right bracket has been pushed onto stack), and 3 (a matching right bracket has been pushed onto stack, but then popped off).

Table 20
Feature templates for the phrase-structure parser.

Description	Feature templates
Unigrams	S_0tc, S_0wc, S_1tc, S_1wc, S_2tc, S_2wc, S_3tc, S_3wc,
	N_0wt, N_1wt, N_2wt, N_3wt,
	S_0lwc, S_0rwc, S_0uwc, S_1lwc, S_1rwc, S_1uwc,
Bigrams	S_0wS_1w, S_0wS_1c, S_0cS_1w, S_0cS_1c,
	S_0wN_0w, S_0wN_0t, S_0cN_0w, S_0cN_0t,
	N_0wN_1w, N_0wN_1t, N_0tN_1w, N_0tN_1t
	S_1wN_0w, S_1wN_0t, S_1cN_0w, S_1cN_0t,
Trigrams	$S_0cS_1cS_2c$, $S_0wS_1cS_2c$, $S_0cS_1wS_2c$, $S_0cS_1cS_2w$,
	$S_0cS_1cN_0t$, $S_0wS_1cN_0t$, $S_0cS_1wN_0t$, $S_0cS_1cN_0w$
Bracket	S_0wb, S_0cb
	S_0wS_1cb, S_0cS_1wb, S_0cS_1cb, S_0wN_0tb, S_0cN_0wb, S_0cN_0tb
Separator	S_0wp, S_0wcp, S_0wq, S_0wcq, S_1wp, S_1wcp, S_1wq, S_1wcq
	S_0cS_1cp, S_0cS_1cq

w = word; c = constituent label; t = POS-tag.

Table 21
The standard split of CTB2 data for phrase-structure parsing.

	Sections	Sentences	Words
Training	001–270	3,475	85,058
Development	301–325	355	6,821
Test	271–300	348	8,008

The Separator row shows features that include one of the separator punctuations (i.e., “，”, “。”, “、”, and “；”) between the head words of S_0 and S_1. These templates apply only when the stack contains at least two nodes; p represents a separator punctuation symbol. Each unique separator punctuation between S_0 and S_1 is only counted once when generating the global feature vector. q represents the count of any separator punctuation between S_0 and S_1.

6.1.1 The Comparability of State Items. Just as in our dependency parsers, the phrase-structure parser is based on an incremental shift-reduce parsing process, and state items built with the same number of transition actions are compared with each other during decoding. However, due to the possibility of unary-reduce actions, the number of actions used to build full parse trees can vary given an input sentence. This makes the comparison of state items more difficult than for dependency parsing, because when some state items on the agenda are completed (i.e., in the final state), others on the agenda may still need more actions to complete. We choose to push completed state items back onto the agenda during the decoding process, without changing them. The decoding continues until the highest scored state item on the agenda is completed. When decoding stops, the highest scored state item on the agenda is taken as the final output. In this process, completed state items that do not have the highest score in the agenda are kept unchanged and compared with incomplete state items, even though they may have been built with different numbers of actions. Our experiments showed that this approach gave reasonable accuracies.

6.2 Experiments

The experiments were performed using the Chinese Treebank 2 (Table 21) and Chinese Treebank 5 data. Standard data preparation was performed before the experiments: Empty terminal nodes were removed; any non-terminal nodes with no children were removed; any unary $X \rightarrow X$ nodes resulting from the previous steps were collapsed into one X node.

For all experiments, we used the EVALB tool[5] for evaluation, and used labeled recall (LR), labeled precision (LP) and $F1$ score (which is the harmonic mean of LR and LP) to measure parsing accuracy.

6.2.1 Test Results on CTB2. The following tests were performed using both gold-standard POS-tags and POS-tags automatically assigned by a POS-tagger. We used our baseline POS-tagger in Section 4 for automatic POS-tagging. The results of various models

[5] `http://nlp.cs.nyu.edu/evalb/`.

Table 22
Accuracies of various phrase-structure parsers on CTB2 with gold-standard POS-tags.

Model	*LR*	*LP*	*F*1
Bikel Thesis	80.9%	84.5%	82.7%
Wang 2006 SVM	87.2%	88.3%	87.8%
Wang 2006 Stacked	88.3%	88.1%	88.2%
Our parser	89.4%	90.1%	89.8%

See text for details.

evaluated on sentences with less than 40 words and using gold-standard POS-tags are shown in Table 22. The rows represent the model from Bikel and Chiang (2000) and Bikel (2004), the SVM and ensemble models from Wang et al. (2006), and our parser, respectively. The accuracy of our parser is competitive using this test set.

The results of various models using automatically assigned POS-tags are shown in Table 23. The rows in the table represent the models from Bikel and Chiang (2000), Levy and Manning (2003), Xiong et al. (2005), Bikel (2004), Chiang and Bikel (2002), the SVM model from Wang et al. (2006), the ensemble system from Wang et al. (2006), and the parser of this article, respectively. Our parser gave comparable accuracies to the SVM and ensemble models from Wang et al. (2006). However, comparison with Table 22 shows that our parser is more sensitive to POS-tagging errors than some of the other models. One possible reason is that some of the other parsers (e.g., Bikel 2004) use the parser model itself to resolve tagging ambiguities, whereas we rely on a POS-tagger to accurately assign a single tag to each word. In fact, for the Chinese data, POS-tagging accuracy is not high, with the perceptron-based tagger achieving an accuracy of only 93%. The beam-search decoding framework we use could accommodate joint parsing and tagging, although the use of features based on the tags of incoming words complicates matters somewhat, because these features rely on tags having been assigned to all words in a pre-processing step. One possible solution would be generating multiple POS-tags for each word during tagging, and incorporating tag information into the shift action, so that the parser will resolve the POS-tag ambiguity. We leave this problem for future work.

Table 23
Accuracies of various phrase-structure parsers on CTB2 with automatically assigned tags.

	≤ 40 words				≤ 100 words				Unlimited			
	LR	*LP*	*F*1	*POS*	*LR*	*LP*	*F*1	*POS*	*LR*	*LP*	*F*1	*POS*
Bikel 2000	76.8%	77.8%	77.3%	-	73.3%	74.6%	74.0%	-	-	-	-	-
Levy 2003	79.2%	78.4%	78.8%	-	-	-	-	-	-	-	-	-
Xiong 2005	78.7%	80.1%	79.4%	-	-	-	-	-	-	-	-	-
Bikel Thesis	78.0%	81.2%	79.6%	-	74.4%	78.5%	76.4%	-	-	-	-	-
Chiang 2002	78.8%	81.1%	79.9%	-	75.2%	78.0%	76.6%	-	-	-	-	-
W06 SVM	78.1%	81.1%	79.6%	92.5%	75.5%	78.5%	77.0%	92.2%	75.0%	78.0%	76.5%	92.1%
W06 Stacked	79.2%	81.1%	80.1%	92.5%	76.7%	78.4%	77.5%	92.2%	76.2%	78.0%	77.1%	92.1%
Our parser	80.2%	80.5%	80.4%	93.5%	76.5%	77.7%	77.1%	93.1%	76.1%	77.4%	76.7%	93.0%

See text for details.

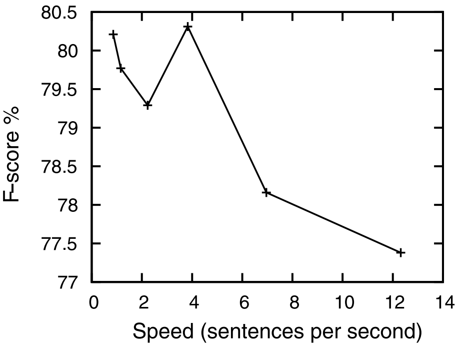

Figure 13
The accuracy/speed tradeoff graph for the phrase-structure parser.

Petrov and Klein (2007) reported *LR* and *LP* of 85.7% and 86.9% for sentences with less than 40 words and 81.9% and 84.8% for all sentences on the CTB2 test set, respectively. These results are significantly better than any model from Table 23. However, we did not include their scores in the table because they used a different training set from CTB5, which is much larger than the CTB2 training set used by all parsers in the table. In order to make a comparison, we split the data in the same way as Petrov and Klein and tested our parser using automatically assigned POS-tags. It gave *LR* and *LP* of 82.0% and 80.9% for sentences with less than 40 words and 77.8% and 77.4% for all sentences, significantly lower than Petrov and Klein. Possible reasons include the sensitivity of our parser to POS-tag errors, and perhaps the use of a latent variable model by Petrov and Klein.

As in the previous sections, we plot the speed/accuracy tradeoff for our phrase-structure parser. For each point in each curve in Figure 13, we run the development test to decide the number of training iterations, and draw the point with speed and accuracy from the final test. Each point in the curve corresponds to $B = 1$, 2, 4, 8, 16, and 32, respectively. The accuracies increased when the beam increased from 1 to 4, but fluctuated when the beam increased beyond 4. In contrast to the development tests, the accuracy reached its maximum when the beam size was 4 rather than 16. However, the general trend of increased accuracy as the speed decreases can still be observed, and the amount of increase diminishes as the speed decreases. These experiments were performed on a Linux platform with a 2.0GHz CPU and a gcc 4.0.1 compiler.

6.2.2 Test Accuracy Using CTB5*.* Table 24 presents the performance of the parser on CTB5. We adopt the data split from the previous section, as shown in Table 17. We used the same parser configurations as Section 6.2.1.

Table 24
Accuracies of our phrase-structure parser on CTB5 using gold-standard and automatically assigned POS-tags.

≤ 40 words				Unlimited			
LR	*LP*	*F*1	*POS*	*LR*	*LP*	*F*1	*POS*
87.9%	87.5%	87.7%	100%	86.9%	86.7%	86.8%	100%
80.2%	79.1%	79.6%	94.1%	78.6%	78.0%	78.3%	93.9%

As an additional evaluation we also produced dependency output from the phrase-structure trees, using the head-finding rules, so that we can compare with dependency parsers. We compare the dependencies read off our constituent parser using CTB5 data with the dependency parser from Section 5, which currently gives the best dependency parsing accuracy on CTB5. The same measures are taken and the accuracies with gold-standard POS-tags are shown in Table 25. Our constituent parser gave higher accuracy than the combined dependency parser. It is interesting that, though the constituent parser uses many fewer feature templates than the dependency parser, the features do include constituent information, which is unavailable to the dependency parser.

6.3 Related Work

Our parser is based on the shift-reduce parsing process from Sagae and Lavie (2005) and Wang et al. (2006), and therefore it can be classified as a transition-based parser (Nivre et al. 2006). An important difference between our parser and the Wang et al. (2006) parser is that our parser is based on a discriminative learning model with global features, whereas the parser from Wang et al. (2006) is based on a local classifier that optimizes each individual choice. Instead of greedy local decoding, we used beam-search in the decoder.

An early work that applies beam-search to constituent parsing is Ratnaparkhi (1999). The main difference between our parser and Ratnaparkhi's is that we use a global discriminative model, whereas Ratnaparkhi's parser has separate probabilities of actions chained together in a conditional model.

Both our parser and the parser from Collins and Roark (2004) use a global discriminative model and an incremental parsing process. The major difference is that Collins and Roark, like Roark (2001), follow a top–down derivation strategy, whereas we chose to use a shift-reduce process which has been shown to give state-of-the-art accuracies for Chinese (Wang et al. 2006). In addition, we did not include a generative baseline model in the discriminative model, as did Collins and Roark (2004).

7. Discussion

We have demonstrated in the previous sections that accuracies competitive with the state-of-the-art can be achieved by our general framework for Chinese word segmentation, joint word segmentation and POS-tagging, Chinese and English dependency parsing, and Chinese phrase-structure parsing. Besides these tasks, our baseline POS-tagger in Section 4 is also implemented in the framework. When it is applied to English POS-tagging using feature templates from Collins (2002), it gave similar accuracy to the

Table 25
Comparison of dependency accuracies between phrase-structure parsing and dependency parsing using CTB5 data.

	Non-root	Root	Complete
Dependency parser	86.21%	76.26%	34.41%
Constituent parser	86.95%	79.19%	36.08%

dynamic-programming decoder of Collins (2002). All these experiments suggest that the general yet efficient framework provides a competitive solution for structural prediction problems with an incremental output-building process. In this section, we discuss the main reasons for the effectiveness of the general framework, as well as its prerequisites, advantages, and limitations when applied to a general task.

One of the main reasons for the high accuracies achieved by this framework is the freedom in using arbitrary features to capture statistical patterns, including those that lead to impractical time complexity with alternative learning and decoding frameworks and algorithms such as CRFs and dynamic programming. This freedom was exploited by our effort to incorporate larger sources of statistical information for the improvement of accuracy. For example, our word-based segmentor extends the character-based approach by including word information; our joint word segmentor and POS-tagger utilizes POS information for word segmentation; our combined dependency parser includes statistical information from two different methods in a single, consistently trained model. The models gave state-of-the-art accuracies in these problems, demonstrating the advantage of using flexible information covering large parts of the output structure.

Compared to alternative discriminative training algorithms such as structural SVM (Tsochantaridis et al. 2004) and CRFs (Lafferty, McCallum, and Pereira 2001; Clark and Curran 2007), the perceptron has a simple parameter update process, which is often efficient in both memory usage and running time depending on the decoding algorithm. Consider joint word segmentation and POS-tagging for example; we found in our experiments that practical training times can be achieved by the perceptron algorithm, but not with structural SVM. MIRA (Crammer and Singer 2003) and its simplifications (McDonald, Crammer, and Pereira 2005; Crammer et al. 2006) are also commonly used learning algorithms to train a global linear model. They can be treated as slower but potentially more accurate alternatives to the perceptron for the general framework. We experimented with these for joint segmentation and tagging but did not improve upon the perceptron.

Beam-search enables training to be performed efficiently for extremely large complex search spaces, for which dynamic programming algorithms may be impractical. For the joint word segmentation and POS-tagging problem, a dynamic-programming decoder is prohibitively slow but a beam-search decoder runs in reasonable time. A more important advantage of beam-search compared to dynamic-programming is that beam-search allows arbitrary features, whereas the efficiency of a dynamic-programming decoder is restricted by the range of features, due to its requirement for optimal substructure. For our combined dependency parser, the feature set makes a dynamic-programming decoder infeasibly slow. From this perspective, beam-search is in line with other recent research on the improvement of accuracies by incorporating non-local features via approximation, such as belief propagation for dependency parsing (Smith and Eisner 2008), integer linear programming for dependency parsing (Martins, Smith, and Xing 2009), and forest reranking for phrase-structure parsing (Huang 2008).

The only prerequisite of the framework is an incremental process, which consumes the input sentence and builds the output structure using a sequence of actions. All four problems studied in the article were first turned into an incremental process, and then solved by applying the framework. The number of distinct actions for a problem is dependent on the complexity of the output. For word segmentation, there are only two actions (append or separate). For transition-based unlabeled dependency parsing, there are four actions (shift, arc-left, arc-right, and reduce). For joint segmentation and

POS-tagging and constituent parsing, there are many more distinct actions according to the set of labels. For example, the number of distinct unary-reduce actions for constituent parsing is equal to the number of distinct constituent labels that form unary-branching nodes. The incremental processes for all four problems have linear time complexity, and a larger number of distinct actions leads to a slower decoder.

One of the most important issues in using beam-search is the comparability of partially built structures at each incremental step during decoding. For word segmentation and joint segmentation and POS-tagging, we compare partially built sentences that have the same number of characters. For word segmentation, partial words can be treated in the same way as full words, without losing much accuracy. The same approach was not effective when applied to joint segmentation and POS-tagging, for which partial words must be treated differently, or avoided by alternative inference such as using multiple-beams. For dependency parsing, we compared partial outputs that have been built using the same number of transition actions. Because all parses for a sentence with size n are built using exactly $2n - 1$ transition actions, this comparison is consistent throughout the decoding process. For constituent parsing, we initially compare partial outputs that have been built using the same number of actions. However, because different output parse trees can contain a different number of unary-reduce actions, some candidate outputs will be completed earlier than others. When this happens, we choose to retain fully built outputs in the agenda while continuing the decoding process, until the highest scored item in the agenda is a complete parse. Therefore, there are situations when different parses in the agenda have a different number of actions. This did not turn out to be a significant problem. We believe that the most effective way to organize output comparison is largely an empirical question.

Finally, alternative components can be used to replace the learning or decoding algorithms of the framework to give higher accuracies. Huang and Sagae (2010) have recently applied dynamic-programming to dependency parsing to pack items that have the same signature in the beam, and obtained higher accuracy than our transition-based dependency parser in Section 5.

8. Conclusion

We explored word segmentation, joint word segmentation and POS-tagging, dependency parsing, and phrase-structure parsing using a general framework of a global linear model, trained by the perceptron algorithm and decoded with beam-search. We have chosen to focus on Chinese; the framework itself and our algorithms for the specific problems are language-independent, however. In Section 5 we reported results on English dependency parsing. Despite the rather simple nature of the decoding and training processes, the framework achieved accuracies competitive with the state-of-the-art for all the tasks we considered, by making use of a large range of statistical information. As further evidence of the effectiveness of our framework, we have recently adapted our phrase-structure parser in Section 6 to parsing with a lexicalized grammar formalism, Combinatory Categorial Grammar (CCG), and achieved higher F-scores than the state-of-the-art C&C CCG parser (Clark and Curran 2007). The range of problems that we have studied suggests that our framework is a simple yet competitive option for structural prediction problems in general.

Our source code can be found at `www.sourceforge.net/projects/zpar`. It contains the general framework, our implementations of the four tasks addressed in this article, and other tools for syntactic processing.

Acknowledgments

This work was largely carried out while Yue Zhang was a DPhil student at the Oxford University Computing Laboratory, where he was supported by an ORS Scholarship and the Clarendon Fund. Stephen Clark was supported by EPSRC grant EP/E035698/1.

References

Bikel, Daniel M. 2004. *On the Parameter Space of Generative Lexicalized Statistical Parsing Models*. Ph.D. thesis, University of Pennsylvania.

Bikel, Daniel M. and David Chiang. 2000. Two statistical parsing models applied to the Chinese Treebank. In *Proceedings of SIGHAN Workshop*, pages 1–6, Hong Kong.

Briscoe, Ted and John Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

Buchholz, Sabine and Erwin Marsi. 2006. aonll-X shared task on multilingual dependency parsing. In *Proceedings of CoNLL*, pages 149–164, New York, NY.

Carreras, Xavier, Michael Collins, and Terry Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of CoNLL*, pages 9–16, Manchester.

Carreras, Xavier, Mihai Surdeanu, and Lluis Marquez. 2006. Projective dependency parsing with perceptron. In *Proceedings of CoNLL*, pages 181–185, New York, NY.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL*, pages 132–139, Seattle, WA.

Chen, Wenliang, Jun’ichi Kazama, Kiyotaka Uchimoto, and Kentaro Torisawa. 2009. Improving dependency parsing with subtrees from auto-parsed data. In *Proceedings of EMNLP*, pages 570–579, Singapore.

Chiang, David and Daniel M. Bikel. 2002. Recovering latent information in treebanks. In *Proceedings of COLING*, pages 183–198, Taipei.

Clark, Stephen and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Collins, Michael. 1999. *Head-driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Collins, Michael. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8, Philadelphia, PA.

Collins, Michael and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*, pages 111–118, Barcelona.

Crammer, Koby, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

Crammer, Koby and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991.

Duan, Xiangyu, Jun Zhao, and Bo Xu. 2007. Probabilistic models for action-based Chinese dependency parsing. In *Proceedings of ECML/ECPPKDD*, pages 559–566, Warsaw.

Emerson, Thomas. 2005. The second international Chinese word segmentation bakeoff. In *Proceedings of SIGHAN Workshop*, pages 123–133, Jeju.

Finkel, Jenny Rose, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL/HLT*, pages 959–967, Columbus, OH.

Freund, Y. and R. Schapire. 1999. Large margin classification using the perceptron algorithm. In Rob Holte, editor, *Machine Learning*. Kluwer, Boston, MA, pages 277–296.

Hall, Johan, Jens Nilsson, Joakim Nivre, Gülsen Eryigit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP/CoNLL*, pages 933–939, Prague.

Huang, Liang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL/HLT*, pages 586–594, Columbus, OH.

Huang, Liang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL*, pages 1077–1086, Uppsala.

Jiang, Wenbin, Liang Huang, Qun Liu, and Yajuan Lü. 2008. A cascaded linear model for joint Chinese word segmentation and part-of-speech tagging. In *Proceedings of ACL/HLT*, pages 897–904, Columbus, OH.

Jiang, Wenbin, Haitao Mi, and Qun Liu. 2008. Word lattice reranking for Chinese word segmentation and part-of-speech tagging. In *Proceedings of COLING*, pages 385–392, Manchester.

Johansson, Richard and Pierre Nugues. 2007. Incremental dependency parsing using online learning. In *Proceedings of the CoNLL/EMNLP*, pages 1134–1138, Prague.

Koo, Terry, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL/HLT*, pages 595–603, Columbus, OH.

Kruengkrai, Canasai, Kiyotaka Uchimoto, Jun'ichi Kazama, Yiou Wang, Kentaro Torisawa, and Hitoshi Isahara. 2009. An error-driven word-character hybrid model for joint Chinese word segmentation and POS tagging. In *Proceedings of ACL/AFNLP*, pages 513–521, Suntec.

Lafferty, J., A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*, pages 282–289, Williamstown, MA.

Levy, Roger and Christopher D. Manning. 2003. Is it harder to parse Chinese, or the Chinese treebank? In *Proceedings of ACL*, pages 439–446, Sapporo.

Martins, Andre, Noah Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of ACL/AFNLP*, pages 342–350, Suntec.

McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98, Ann Arbor, MI.

McDonald, Ryan and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of EMNLP/CoNLL*, pages 122–131, Prague.

McDonald, Ryan and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, pages 81–88, Trento.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT/EMNLP*, pages 523–530, Vancouver.

Nakagawa, Tetsuji and Kiyotaka Uchimoto. 2007. A hybrid approach to word segmentation and POS tagging. In *Proceedings of ACL Demo and Poster Session*, Prague.

Ng, Hwee Tou and Jin Kiat Low. 2004. Chinese part-of-speech tagging: One-at-a-time or all-at-once? Word-based or character-based? In *Proceedings of EMNLP*, pages 227–284, Barcelona.

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP/CoNLL*, pages 915–932, Prague.

Nivre, Joakim, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of CoNLL*, pages 221–225, New York, NY.

Nivre, Joakim and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL/HLT*, pages 950–958, Columbus, OH.

Peng, F., F. Feng, and A. McCallum. 2004. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of COLING*, pages 562–568, Geneva.

Petrov, Slav and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of HLT/NAACL*, pages 404–411, Rochester, NY.

Ratnaparkhi, Adwait. 1998. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. Ph.D. thesis, University of Pennsylvania.

Ratnaparkhi, Adwait. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175.

Roark, Brian. 2001. Probabilistic top–down parsing and language modeling. *Computational Linguistics*, 27:249–276.

Sagae, Kenji and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of IWPT*, pages 125–132, Vancouver.

Sagae, Kenji and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of HLT/NAACL, Companion Volume: Short Papers*, pages 129–132, New York, NY.

Shi, Yanxin and Mengqiu Wang. 2007. A dual-layer CRF based joint decoding method for cascade segmentation and labelling tasks. In *Proceedings of IJCAI*, pages 1707–1712, Hyderabad.

Smith, David and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 145–156, Honolulu, HI.

Sproat, R., C. Shih, W. Gail, and N. Chang. 1996. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3):377–404.

Sproat, Richard and Thomas Emerson. 2003. The first international Chinese word segmentation bakeoff. In *Proceedings of The Second SIGHAN Workshop*, pages 282–289, Sapporo.

Tsochantaridis, I., T. Hofmann, T. Joachims, and Y. Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of ICML*, pages 102–114, Banff.

Wang, Xinhao, Xiaojun Lin, Dianhai Yu, Hao Tian, and Xihong Wu. 2006. Chinese word segmentation with maximum entropy and n-gram language model. In *Proceedings of SIGHAN Workshop*, pages 138–141, Sydney.

Xia, Fei. 2000. *The Part-of-Speech Tagging Guidelines for the Chinese Treebank (3.0)*, University of Pennsylvania.

Xiong, Deyi, Shuanglong Li, Qun Liu, Shouxun Lin, and Yueliang Qian. 2005. Parsing the Penn Chinese Treebank with semantic knowledge. In *Proceedings of IJCNLP*, pages 70–81, Jeju.

Xue, N. 2003. Chinese word segmentation as character tagging. *International Journal of Computational Linguistics and Chinese Language Processing*, 8(1):29–48.

Yamada, H. and Y. Matsumoto. 2003. Statistical dependency analysis using support vector machines. In *Proceedings of IWPT*, pages 195–206, Nancy.

Zhang, Ruiqiang, Genichiro Kikui, and Eiichiro Sumita. 2006. Subword-based tagging by conditional random fields for Chinese word segmentation. In *Proceedings of the HLT/NAACL, Companion*, volume Short Papers, pages 193–196, New York, NY.

Zhang, Yue and Stephen Clark. 2007. Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of ACL*, pages 840–847, Prague.

Zhang, Yue and Stephen Clark. 2008a. Joint word segmentation and POS tagging using a single perceptron. In *Proceedings of ACL/HLT*, pages 888–896, Columbus, OH.

Zhang, Yue and Stephen Clark. 2008b. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*, pages 562–571, Honolulu, HI.

Zhang, Yue and Stephen Clark. 2009. Transition-based parsing of the Chinese Treebank using a global discriminative model. In *Proceedings of IWPT*, pages 162–171, Paris.

Zhang, Yue and Stephen Clark. 2010. A fast decoder for joint word segmentation and POS-tagging using a single discriminative model. In *Proceedings of EMNLP*, pages 843–852, Cambridge, MA.

Zhao, Hai, Chang-Ning Huang, and Mu Li. 2006. An improved Chinese word segmentation system with conditional random field. In *Proceedings of SIGHAN Workshop*, pages 162–165, Sydney.