A Fast Decoder for Joint Word Segmentation and POS-Tagging Using a Single Discriminative Model

Yue Zhang and **Stephen Clark**
University of Cambridge Computer Laboratory
William Gates Building,
15 JJ Thomson Avenue,
Cambridge CB3 0FD, UK
{yue.zhang, stephen.clark}@cl.cam.ac.uk

Abstract

We show that the standard beam-search algorithm can be used as an efficient decoder for the global linear model of Zhang and Clark (2008) for joint word segmentation and POS-tagging, achieving a significant speed improvement. Such decoding is enabled by: (1) separating full word features from partial word features so that feature templates can be instantiated incrementally, according to whether the current character is separated or appended; (2) deciding the POS-tag of a potential word when its first character is processed. Early-update is used with perceptron training so that the linear model gives a high score to a correct partial candidate as well as a full output. Effective scoring of partial structures allows the decoder to give high accuracy with a small beam-size of 16. In our 10-fold cross-validation experiments with the Chinese Treebank, our system performed over 10 times as fast as Zhang and Clark (2008) with little accuracy loss. The accuracy of our system on the standard CTB 5 test was competitive with the best in the literature.

1 Introduction and Motivation

Several approaches have been proposed to solve word segmentation and POS-tagging jointly, including the reranking approach (Shi and Wang, 2007; Jiang et al., 2008b), the hybrid approach (Nakagawa and Uchimoto, 2007; Jiang et al., 2008a), and the single-model approach (Ng and Low, 2004; Zhang and Clark, 2008; Kruengkrai et al., 2009). These methods led to accuracy improvements over the traditional, pipelined segmentation and POS-tagging baseline by avoiding segmentation error propagation and making use of part-of-speech information to improve segmentation.

The single-model approach to joint segmentation and POS-tagging offers consistent training of all information, concerning words, characters and parts-of-speech. However, exact inference with dynamic programming can be infeasible if features are defined over a large enough range of the output, such as over a two-word history. In our previous work (Zhang and Clark, 2008), which we refer to as Z&C08 from now on, we used an approximate decoding algorithm that keeps track of a set of partially built structures for each character, which can be seen as a dynamic programming chart which is greatly reduced by pruning.

In this paper we follow the line of single-model research, in particular the global linear model of Z&C08. We show that effective decoding can be achieved with standard beam-search, which gives significant speed improvements compared to the decoding algorithm of Z&C08, and achieves accuracies that are competitive with the state-of-the-art. Our research is also in line with recent research on improving the speed of NLP systems with little or no accuracy loss (Charniak et al., 2006; Roark and Hollingshead, 2008).

Our speed improvement is achieved by the use of a single-beam decoder. Given an input sentence, candidate outputs are built incrementally, one character at a time. When each character is processed, it is combined with existing candidates in all possible ways to generate new candidates, and an agenda is used to keep the N-best candidate outputs from the begining of the sentence to the current character. Compared to the multiple-beam search algorithm of Z&C08, the use of a single beam can lead to an order of magnitude faster decoding speed.

1.1 The processing of partial words

An important problem that we solve in this paper is the handling of partial words with a single beam decoder for the global model. As we pointed out in Z&C08, 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 Z&C08 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, since 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 Z&C08 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, since it explores a larger part of the search space than a single-beam decoder, its time complexity is correspondingly higher.

In this paper, 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 above, assigning POS tags to partial words as if they were full words leads to low accuracy.

An obvious solution to the above 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, since the scores of full words are futher supported by POS and POS ngram 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 Z&C08 system.

In this paper, 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”.

As a main contribution of this paper, 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. The 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.[1]

The mechanism of first-character decision of POS also boosts the efficiency, since the enumeration of POS is unecessary when a character is appended to the end of an existing word. As a result, the complexity of each processing step is reduce 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 and its part-of-speech, etc. 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 use the averaged perceptron (Collins, 2002) for training, together with the “early update” mechanism of Collins and Roark (2004). Rather than updating the parameters after decoding is complete, the modified algorithm updates parameters at any processing step if the correct partial candidate falls out of the beam.

In our experiments using the Chinese Treebank data, our system ran an order of magnitude faster than our Z&C08 system with little loss of accuracy. The accuracy of our system was competitive with other recent models.

[1]The next incoming characters are also a useful source of information for predicting the POS. However, our system achieved competitive accuracy with Z&C08 without such character lookahead features.

2 Model and Feature Templates

We use a linear model to score both partial and full candidate outputs. Given an input x, the score of a candidate output y is computed as:

$$\text{Score}(y) = \Phi(y) \cdot \vec{w},$$

where $\Phi(y)$ is the global feature vector extracted from y, and $\vec{w}$ is the parameter vector of the model.

Figure 1 shows the feature templates for the model, where templates 1 – 14 contain only segmentation information and templates 15 – 29 contain both segmentation and POS information. Each template is instantiated according to the current character in the decoding process. Row “For” shows the conditions for template instantiation, where “s” indicates that the corresponding template is instantiated when the current character starts a new word, and “a” indicates that the corresponding template is instantiated when the current character does not start a new word. In the row for feature templates, w, t and c are used to represent a word, a POS-tag and a character, respectively. The subscripts are based on the current character, where w_{-1} represents the first word to the left of the current character, and p_{-2} represents the POS-tag on the second word to the left of the current character, and so on. As an example, feature template 1 is instantiated when the current character starts a new word, and the resulting feature value is the word to the left of this character. $start(w)$, $end(w)$ and $len(w)$ represent the first character, the last character and the length of word w, respectively. The length of a word is normalized to 16 if it is larger than 16. $cat(c)$ represents the POS category of character c, which is the set of POS-tags seen on character c, as we used in Z&C08.

Given a partial or complete candidate y, its global feature vector $\Phi(y)$ is computed by instantiating all applicable feature templates from Table 1 for each character in y, according to whether or not the character is separated from the previous character.

The feature templates are mostly taken from, or inspired by, the feature templates of Z&C08. Templates 1, 2, 3, 4, 5, 8, 10, 12, 13, 14, 15, 19, 20,

	Feature template	For
1	w_{-1}	s
2	$w_{-1}w_{-2}$	s
3	w_{-1}, where $len(w_{-1}) = 1$	s
4	$start(w_{-1})len(w_{-1})$	s
5	$end(w_{-1})len(w_{-1})$	s
6	$end(w_{-1})c_0$	s
7	$c_{-1}c_0$	a
8	$begin(w_{-1})end(w_{-1})$	s
9	$w_{-1}c_0$	s
10	$end(w_{-2})w_{-1}$	s
11	$start(w_{-1})c_0$	s
12	$end(w_{-2})end(w_{-1})$	s
13	$w_{-2}len(w_{-1})$	s
14	$len(w_{-2})w_{-1}$	s
15	$w_{-1}t_{-1}$	s
16	$t_{-1}t_0$	s
17	$t_{-2}t_{-1}t_0$	s
18	$w_{-1}t_0$	s
19	$t_{-2}w_{-1}$	s
20	$w_{-1}t_{-1}end(w_{-2})$	s
21	$w_{-1}t_{-1}c_0$	s
22	$c_{-2}c_{-1}c_0t_{-1}$, where $len(w_{-1}) = 1$	s
23	$start(w_0)t_0$	s
24	$t_{-1}start(w_{-1})$	s
25	t_0c_0	s, a
26	$c_0t_0start(w_0)$	a
27	$ct_{-1}end(w_{-1})$, where $c \in w_{-1}$ and $c \neq end(w_{-1})$	s
28	$c_0t_0cat(start(w_0))$	s
29	$ct_{-1}cat(end(w_{-1}))$, where $c \in w_{-1}$ and $c \neq end(w_{-1})$	s
30	$c_0t_0c_{-1}t_{-1}$	s
31	$c_0t_0c_{-1}$	a

Table 1: Feature templates.

24, 27 and 29 concern complete word information, and they are used in the model to differentiate correct and incorrect output structures in the same way as our Z&C08 model. Templates 6, 7, 9, 16, 17, 18, 21, 22, 23, 25, 26 and 28 concern partial word information, whose role in the model is to indicate the likelihood that the partial word including the current character will become a correct full word. They act as guidance for the action to take for the current character according to the context, and are the crucial reason for the effectiveness of the algorithm with a small beam-size.

```
function DECODE(sent, agenda):
    CLEAR(agenda)
    ADDITEM(agenda, “”)
    for index in [0..LEN(sent)]:
        for cand in agenda:
            new ← APPEND(cand, sent[index])
            ADDITEM(agenda, new)
            for pos in TAGSET():
                new ← SEP(cand, sent[index], pos)
                ADDITEM(agenda, new)
        agenda ← N-BEST(agenda)
    return BEST(agenda)
```

Figure 1: The incremental beam-search decoder.

2.1 Decoding

The decoding algorithm builds an output candidate incrementally, one character at a time. Each character can either be attached to the current word or separated as the start a new word. When the current character starts a new word, a POS-tag is assigned to the new word. An agenda is used by the decoder to keep the N-best candidates during the incremental process. 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 N-best newly generated candidates are put back onto the agenda. After all input characters have been processed, the highest-scored candidate from the agenda is taken as the output.

Pseudo code for the decoder is shown in Figure 1. CLEAR removes all items from the agenda, ADDITEM adds a new item onto the agenda, N-BEST returns the N highest-scored items from the agenda, and BEST returns the highest-scored item from the agenda. LEN returns the number of characters in a sentence, and *sent*[i] returns the ith character from the sentence. APPEND appends a character to the last word in a candidate, and SEP joins a character as the start of a new word in a candidate, assigning a POS-tag to the new word.

Both our decoding algorithm and the decoding algorithm of Z&C08 run in linear time. However, in order to generate possible candidates for each character, Z&C08 uses an extra loop to search for possible words that end with the current character. A restriction to the maximum word length is applied to limit the number of iterations in this loop, without which the algorithm would have quadratic time complexity. In contrast, our decoder does not search backword for the possible starting character of any word. Segmentation ambiguities are resolved by binary choices between the actions append or separate for each character, and no POS enumeration is required when the character is appended. This improves the speed by a significant factor.

2.2 Training

The learning algorithm is based on the generalized perceptron (Collins, 2002), but parameter adjustments can be performed at any character during the decoding process, using the "early update" mechanism of Collins and Roark (2004).

The parameter vector of the model is initialized as all zeros before training, and used to decode training examples. Each training example is turned into the raw input format, and processed in the same way as decoding. After each character is processed, partial candidates in the agenda are compared to the corresponding gold-standard output for the same characters. If none of the candidates in the agenda are correct, the decoding is stopped and the parameter vector is updated by adding the global feature vector of the gold-standard partial output and subtracting the global feature vector of the highest-scored partial candidate in the agenda. The training process then moves on to the next example. However, if any item in the agenda is the same as the corresponding gold-standard, the decoding process moves to the next character, without any change to the parameter values. After all characters are processed, the decoder prediction is compared with the training example. If the prediction is correct, the parameter vector is not changed; otherwise it is updated by adding the global feature vector of the training example and subtracting the global feature vector of the decoder prediction, just as the perceptron algorithm does. The same training examples can be used to train the model for multiple iterations. We use the averaged parameter vector (Collins, 2002) as the final model.

Pseudocode for the training algorithm is shown in Figure 2. It is based on the decoding algorithm in Figure 1, and the main differences are: (1) the training algorithm takes the gold-standard output and the parameter vector as two additional arguments; (2) the training algorithm does not return a prediction, but modifies the parameter vector when necessary; (3) lines 11 to 20 are additional lines of code for parameter updates.

Without lines 11 to 16, the training algorithm is exactly the same as the generalized perceptron algorithm. These lines are added to ensure that the agenda contains highly probable candidates during the whole beam-search process, and they are crucial to the high accuracy of the system. As stated earlier, the decoder relies on proper scoring of partial words to maintain a set of high quality candidates in the agenda. Updating the value of the parameter vector for partial outputs can be seen as a means to ensure correct scoring of partial candidates at any character.

2.3 Pruning

We follow Z&C08 and use several pruning methods, most of which serve to 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 with 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.

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 allowed POS-tags according to the tag dictionary, while other words are allowed 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 num-

[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).

```
function TRAIN(sent, agenda, gold-standard, $\vec{w}$):
01:  CLEAR(agenda)
02:  ADDITEM(agenda, “”)
03:  for index in [0..LEN(sent)]:
04:    for cand in agenda:
05:      new ← APPEND(cand, sent[index])
06:      ADDITEM(agenda, new)
07:      for pos in TAGSET():
08:        new ← SEP(cand, sent[index], pos)
09:        ADDITEM(agenda, new)
10:    agenda ← N-BEST(agenda)
11:    for cand in agenda:
12:      if cand = gold-standard[0:index]:
13:        CONTINUE
14:    $\vec{w}$ ← $\vec{w}$ + Φ(gold-standard[0:index])
15:    $\vec{w}$ ← $\vec{w}$ - Φ(BEST(agenda))
16:    return
17:  if BEST(agenda) $\neq$ gold-standard:
18:    $\vec{w}$ ← $\vec{w}$ + Φ(gold-standard)
19:    $\vec{w}$ ← $\vec{w}$ - Φ(BEST(agenda))
20:    return
21:  return
```

Figure 2: The incremental learning function.

ber of times the most frequent word has been seen with M, a word is a frequent word if it has been seen more than $M/5000 + 5$ times. The threshold value is taken from Z&C08, 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 udpated 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 generated from *cand1* will always have a higher score than the highest scored candidate generated from *cand2*.

From the above pruning methods, only the third was not used by Z&C08. It can be seen as an extra mechanism to help keep likely partial words in the agenda and improve the accuracy, but which does not give our system a speed advantage over Z&C08.

3 Experiments

We used the Chinese Treebank (CTB) data to perform one set of development tests and two sets of fi-

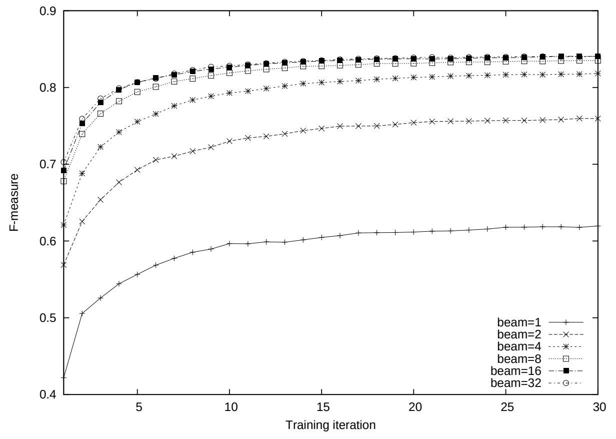

Figure 3: The influence of beam-sizes, and the convergence of the perceptron.

nal tests. The CTB 4 was split into two parts, with the CTB 3 being used for a 10-fold cross validation test to compare speed and accuracies with Z&C08, and the rest being used for development. The CTB 5 was used to perform the additional set of experiments to compare accuracies with other recent work.

We use the standard F-measure to evaluate output accuracies. For word segmentation, precision is defined as the number of correctly segmented words divided by the total number of words in the output, and recall is defined as the number of correctly segmented words divided by the total number of words in the gold-standard output. For joint segmentation and POS-tagging, precision is defined as the number of correctly segmented and POS-tagged words divided by the total number of words from the output, and recall is defined as the correctly segmented and POS-tagged words divided by the total number of words in the gold-standard output.

All our experiments were performed on a Linux platform, and a single 2.66GHz Intel Core 2 CPU.

3.1 Development tests

Our development data consists of 150K words in 4798 sentences. 80% of the data were randomly chosen as the development training data, while 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, the effect of partial features in beam-search decoding, and the effect of incremental learning (i.e. early update).

Figure 3 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, while the amount of increase becomes small when the size of the beam becomes 16. After the 10th 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 7.16s, 11.90s, 18.42s, 27.82s, 46.77s and 89.21s, 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 3 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 incremental training: We compare the accuracies by incremental training using early update and normal perceptron training. In the normal perceptron training case, lines 11 to 16 are taken out of the training algorithm in Figure 2. The algorithm reached the best performance at the 22nd iteration, with the segmentation F-score being 90.58% and joint F-score being 83.38%. In the incremental training case, 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%.

3.2 Final tests using CTB 3

CTB 3 consists of 150K words in 10364 sentences. We follow Z&C08 and split it into 10 equal-sized parts. In each test, one part is taken as the test data and the other nine are combined together as the training data. We compare the speed and accuracy with the joint segmentor and tagger of Z&C08, which is publicly available as the ZPar system, version 0.2[3].

The results are shown in Table 2, where each row shows one cross validation test. The column headings “sf”, “jf”, “time” and “speed” refer to segmentation F-measure, joint F-measure, testing time (in

[3]http://www.sourceforge.net/projects/zpar

	Z&C08				this paper			
#	sf	jf	time	speed	sf	jf	time	speed
1	97.18	93.27	557.97	1.86	97.25	93.51	44.20	23.44
2	97.65	93.81	521.63	1.99	97.66	93.97	42.07	24.26
3	96.08	91.04	444.69	2.33	95.55	90.65	39.23	26.41
4	96.31	91.93	431.04	2.40	96.37	92.15	39.54	26.20
5	96.35	91.94	508.39	2.04	95.84	91.51	43.30	23.93
6	94.48	88.63	482.78	2.15	94.25	88.53	43.77	23.67
7	95.27	90.52	361.95	2.86	95.10	90.42	41.76	24.81
8	94.98	90.01	418.54	2.47	94.87	90.30	39.81	26.02
9	95.23	90.84	471.3	2.20	95.21	90.55	42.03	26.65
10	96.49	92.11	500.72	2.08	96.33	92.12	43.12	24.03
average	96.00	91.41	469.90	2.24	95.84	91.37	41.88	24.94

Table 2: Speed and acccuracy comparisons with Z&C08 by 10-fold cross validation.

seconds) and testing speed (in the number of sentences per second), respectively.

Our system gave a joint segmentation and POS-tagging F-score of 91.37%, which is only 0.04% lower than that of ZPar 0.2. The speed of our system was over 10 times as fast as ZPar 0.2.

3.3 Final tests using CTB 5

We follow Kruengkrai et al. (2009) and split the CTB 5 into training, development testing and testing sets, as shown in Table 3. We ignored the development test data since our system had been developed in previous experiments.

Kruengkrai et al. (2009) made use of character type knowledge for spaces, numerals, symbols, alphabets, 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 simple rules, treating consecutive English letters or Arabic numbers as a single word.

The results are shown in Table 4, where row “N07” refers to the model of Nakagawa and Uchimoto (2007), rows “J08a” and “b” refer to the models of Jiang et al. (2008a) and Jiang et al. (2008b), and row “K09” refers to the models of Kruengkrai et al. (2009). Columns “sf” and “jf” refer to segmentation and joint 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.

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

Table 3: Training, development and test data on CTB 5.

	sf	jf
K09 (error-driven)	97.87	93.67
our system	97.78	93.67
K09 (baseline)	97.79	93.60
J08a	97.85	93.41
J08b	97.74	93.37
N07	97.83	93.32

Table 4: Accuracy comparisons with recent studies on CTB 5.

4 Related Work

The effectiveness of our beam-search decoder showed that the joint segmentation and tagging problem may be less complex than previously perceived (Zhang and Clark, 2008; Jiang et al., 2008a). At the very least, the single model approach with a simple decoder achieved competitive accuracies to what has been achieved so far by the reranking (Shi and Wang, 2007; Jiang et al., 2008b) models and an ensemble model using machine-translation techniques (Jiang et al., 2008a). This may shed new light on joint segmentation and POS-tagging methods.

Kruengkrai et al. (2009) and Zhang and Clark (2008) are the most similar to our system among related work. Both systems use a discriminatively trained linear model to score candidate outputs. The work of Kruengkrai et al. (2009) is based on Nakagawa and Uchimoto (2007), which separates the processing of known words and unknown words, and uses a set of segmentation tags to represent the segmentation of characters. In contrast, our model is conceptually simpler, and does not differentiate known words and unknown words. Moreover, our model is based on our previous work, in line with Zhang and Clark (2007), which does not treat word segmentation as character sequence labeling.

Our learning and decoding algorithms are also different from Kruengkrai et al. (2009). While Kruengkrai et al. (2009) perform dynamic programming and MIRA learning, we use beam-search to perform incremental decoding, and the early-update version of the perceptron algorithm to train the model. Dynamic programming is exact inference, for which the time complexity is decided by the locality of feature templates. In contrast, beam-search is approximate and can run in linear time. The parameter updating for our algorithm is conceptually and computationally simpler than MIRA, though its performance can be slightly lower. However, the early-update mechanism we use is consistent with our incremental approach, and improves the learning of the beam-search process.

5 Conclusion

We showed that a simple beam-search decoding algorithm can be effectively applied to the decoding problem for a global linear model for joint word segmentation and POS-tagging. By guiding search with partial word information and performing learning for partial candidates, our system achieved significantly faster speed with little accuracy loss compared to the system of Z&C08.

The source code of our joint segmentor and POS-tagger can be found at:
www.sourceforge.net/projects/zpar, version 0.4.

Acknowledgements

We thank Canasai Kruengkrai for discussion on efficiency issues, and the anonymous reviewers for their suggestions. Yue Zhang and Stephen Clark are supported by the European Union Seventh Framework Programme (FP7-ICT-2009-4) under grant agreement no. 247762.

References

Eugene Charniak, Mark Johnson, Micha Elsner, Joseph Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R. Shrivaths, Jeremy Moore, Michael Pozar, and Theresa Vu. 2006. Multilevel coarse-to-fine PCFG parsing. In *Proceedings of HLT/NAACL*, pages 168–175, New York City, USA, June. Association for Computational Linguistics.

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

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

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

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

Canasai Kruengkrai, 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, Singapore, August.

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

Hwee Tou Ng 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*, Barcelona, Spain.

Brian Roark and Kristy Hollingshead. 2008. Classifying chart cells for quadratic complexity context-free inference. In *Proceedings of COLING*, pages 745–752, Manchester, UK, August. Coling 2008 Organizing Committee.

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

Fei Xia, 2000. *The part-of-speech tagging guidelines for the Chinese Treebank (3.0)*.

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

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