Incremental Joint POS Tagging and Dependency Parsing in Chinese

Jun Hatori[1] **Takuya Matsuzaki**[1] **Yusuke Miyao**[3] **Jun'ichi Tsujii**[4]
[1]University of Tokyo / 7-3-1 Hongo, Bunkyo, Tokyo, Japan
[2]National Institute of Informatics / 2-1-2 Hitotsubashi, Chiyoda, Tokyo, Japan
[3]Microsoft Research Asia / 5 Danling Street, Haidian District, Beijing, P.R. China
{hatori,matuzaki}@is.s.u-tokyo.ac.jp
yusuke@nii.ac.jp jtsujii@microsoft.com

Abstract

We address the problem of joint part-of-speech (POS) tagging and dependency parsing in Chinese. In Chinese, some POS tags are often hard to disambiguate without considering long-range syntactic information. Also, the traditional pipeline approach to POS tagging and dependency parsing may suffer from the problem of error propagation. In this paper, we propose the first incremental approach to the task of joint POS tagging and dependency parsing, which is built upon a shift-reduce parsing framework with dynamic programming. Although the incremental approach encounters difficulties with underspecified POS tags of look-ahead words, we overcome this issue by introducing so-called *delayed features*. Our joint approach achieved substantial improvements over the pipeline and baseline systems in both POS tagging and dependency parsing task, achieving the new state-of-the-art performance on this joint task.

1 Introduction

The tasks of part-of-speech (POS) tagging and dependency parsing have been widely investigated since the early stages of NLP research. Among mainstream approaches to dependency parsing, an incremental parsing framework is commonly used (e.g. Nivre (2008); Huang and Sagae (2010)), mainly because it achieves state-of-the-art accuracy while retaining linear-time computational complexity, and is also considered to reflect how humans process natural language sentences (Frazier and Rayner, 1982).

However, although some of the Chinese POS tags require long-range syntactic information in order to be disambiguated, to the extent of our knowledge, none of the previous approaches have addressed the joint modeling of these two tasks in an incremental framework. Also, since POS tagging is a preliminary step for dependency parsing, the traditional pipeline approach may suffer from the problem of error propagation.

In the example sentence in Figure 1, 的 has POS ambiguity between DEG (a genitive marker), which connect two noun phrases as "'s" in English, and DEC (a complementizer), which introduces a relative clause. Since both can take the form of NP-的-NP (NP: noun phrase), it is hard to distinguish these two tags only by considering local context. Based only on local context, a standard n-gram tagger is likely to assign the wrong tag DEG to 的, which inevitably makes the following parsing step fail to process the sentence correctly. However, knowing that the NP preceding 的 is the object of 沟通/VV (VV: verb), we can assume that 的 is a complementizer because 的/DEG is unlikely to follow a verb phrase.

In this paper, we propose the first incremental approach to the task of joint POS tagging and dependency parsing. Given a segmented sentence, our model simultaneously considers possible POS tags and dependency relations within the given beam, and outputs the best parse along with POS tags. However, the combined model raises two challenges: First, since the combined search space is huge, efficient decoding is difficult while the naïve use of beam is likely to degrade the search quality. Second, since the proposed model performs joint POS tagging and dependency parsing in a left-to-right manner, the model cannot exploit look-ahead POS tags to determine the next action.

To deal with the increased search space, we adopt a recently-proposed dynamic programming (DP) extension to shift-reduce parsing (Huang and Sagae, 2010), which enables the model to pack equivalent parser states, improving both speed and accuracy. Also, we overcome the lack of look-ahead POS information by introducing a concept of *delayed features*. The delayed features are those features that include underspecified POS tags, and shall be evaluated at the step when the look-ahead tags are determined. Based on experiments on the Chinese Penn Treebank (CTB) 5, we show that our joint models substantially improve over the

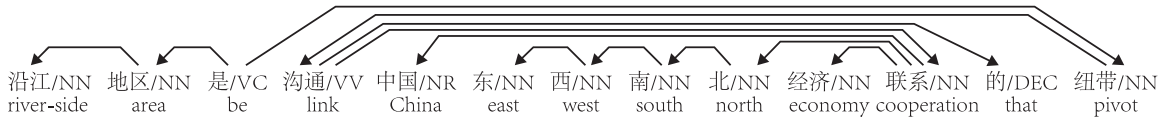

Figure 1: An example sentence from the Chinese Penn Treebank (CTB) 5.
("*The river-side area is the pivot that links China's across-the-country economic cooperation*.")

w_j	t_{j-1} $t_{j-1} \circ t_{j-2}$ w_{j+1} [1)]
$w_j \circ E(w_{j-1})$ [2)]	$w_j \circ B(w_{j+1})$ [2)]
$E(w_{j-1}) \circ w_j \circ B(w_{j+1})$ [3)]	
$B(w_j)$ $E(w_j)$	$P(B(w_j))$ $P(E(w_j))$
$C_n(w_j)$	$(n \in \{2, \ldots, len(w_j) - 1\})$
$B(w_j) \circ C_n(w_j)$	$(n \in \{2, \ldots, len(w_j)\})$
$E(w_j) \circ C_n(w_j)$	$(n \in \{1, \ldots, len(w_j) - 1\})$
$C_n(w_j)$	*(if $C_n(w_j)$ equals to $C_{n+1}(w_j)$)*

1) if $len(w_{j+1}) < 3$; 2) if $len(w_j) < 3$; 3) if $len(w_j) = 1$.

Table 1: Feature templates for the baseline POS tagger, where t_i is the tag assigned to the i-th word w_i, $B(w)$ and $E(w)$ is the beginning and the ending character of word w, $C_n(w)$ is the n-th character of w, $P(c)$ is the set of tags associated with the single-character word c based on the dictionary.

pipeline and baseline systems in both POS tagging and dependency parsing accuracy. We also present some discussion on the results and error analysis. Although we specifically focus on Chinese in this work, our joint model is applicable to any languages for which a projective shift-reduce parser works well.

2 Baseline Models

First of all, we describe our baseline POS tagger and dependency parsers. These models will later be combined into pipelined models, which are then used as the baseline models in Section 4.

2.1 Baseline POS Tagger

We build a baseline POS tagger, which uses the same POS-tagging features as those used in the state-of-the-art joint word segmentation and POS tagging model for Chinese (Zhang and Clark, 2008a). The list of features are shown in Table 1. We train the model with the averaged perceptron (Collins, 2002), and the decoding is performed using the Viterbi algorithm with beam search.

Following Zhang and Clark (2008a), we use a tag dictionary and closed-set tags, which lead to improvement in both speed and accuracy. During training, the model stores all word–tag pairs into a tag dictionary, and for each word occurring more than N times in the training data, the decoder only assigns one of the tags that have been seen in the training data. For words that do not exist in the dictionary, the decoder still considers every possible tag. We also construct a dictionary for the closed-set tags (Xia, 2000), and allow the decoder to assign these tags only to the words listed in the dictionary.

2.2 Baseline Parsers

For the baseline parsers for experiments, we build two dependency parsers: a reimplementation of the parser by Huang and Sagae (2010) (hereinafter *Parser-HS*), which is a shift-reduce dependency parser enhanced with dynamic programming (DP) using graph-structured stack (GSS; Tomita (1991)), and our extension of Parser-HS by incorporating a richer set of features taken from Zhang and Nivre (2011) (hereinafter *Parser-ZN*), which is originally a non-DP arc-eager dependency parser and achieves the current state-of-the-art performance for Chinese dependency parsing. In this section, we briefly describe these models since the features and DP formalism serve as a basis for the joint models described in Section 3.

2.2.1 Shift-reduce parsing

Shift-reduce dependency parsing algorithms incrementally process an input sentence from left to right. In the framework known as "arc-standard" (Nivre, 2008), the parser performs one of the following three actions at each step:

- SHIFT (SH): move the first word in the input queue, q_0, onto the stack
- REDUCE-RIGHT (RR): combine the top two trees on the stack, (s_0, s_1), into a subtree $s_0^{\curvearrowleft} s_1$
- REDUCE-LEFT (RL): combine the top two trees on the stack, (s_0, s_1), into a subtree $s_0^{\curvearrowright} s_1$

where $S = (\ldots, s_1, s_0)$ is a stack of trees and $Q = (q_0, q_1, \ldots, q_{n-j-1}) = (w_j, w_{j+1}, \ldots, w_{n-1})$ is an input queue where j is the index of the first word in the queue Q and n is the number of words in the input sentence. Note that $s_0^{\curvearrowright} s_1$ denotes a combined tree where s_1 is a child of s_0.

To deal with conflicts between more than one of these actions, each action is associated with a score, and the score of a parser state is the total score of the actions that have been applied. To train the model, we adopt the averaged perceptron algorithm (Collins, 2002) with early update (Collins and Roark, 2004), following Huang and Sagae (2010). With the early update, whenever the gold action sequence falls off from the beam, the parameters are immediately updated with the rest of the sentence neglected.

2.2.2 Merging equivalent states

Dynamic programming is enabled by merging *equivalent states*: if two states produce the same feature vector, they are merged into one state. Formally, a parser state (or configuration) ψ is described by $\langle \ell, i, j, S \rangle$, where ℓ is the current step, $[i \dots j]$ is the span of the top tree s_0 in the stack $S = (s_{d-1}, \dots, s_0)$, where d is the depth of the stack. The equivalence of two states $\psi : \langle \ell, i, j, S \rangle$ and $\psi' : \langle \ell', i', j', S' \rangle$ is then defined as

$$\psi \sim \psi' \;\; \textit{iff} \;\; j = j' \wedge \vec{f}(j, S) = \vec{f}(j', S'), \quad (1)$$

where $\vec{f}(j, S)$ is the feature vector of the state $\langle \ell, i, j, S \rangle$. In practice, just remembering a minimal set of features called *kernel features* $\tilde{\vec{f}}(j, S)$ suffices to evaluate the equivalence of states:

$$\tilde{\vec{f}}(j, S) = \tilde{\vec{f}}(j', S') \Rightarrow \langle \ell, i, j, S \rangle \sim \langle \ell', i', j', S' \rangle. \quad (2)$$

By merging equivalent states based on this condition, we only need to remember relevant information from the top d ($d = 3$ in our models) trees on the stack to evaluate the score of the next actions. However, since the stack shrinks when a REDUCE-LEFT/RIGHT action is applied, you often need to recover the last element of the stack from the history. Following Huang and Sagae (2010), we use a concept of *predictor states* $\Pi(\psi)$ to retain the links to multiple different histories.

2.2.3 Features

The feature templates used in the baseline parser Parser-HS are listed in Table 2 (a), where $s.w$ and $s.t$ are the form and tag of the root word of tree s, $s.\mathrm{rc}$ and $s.\mathrm{lc}$ are the right- and left-most children of s, and $\circ$ denotes conjunction of features. Note that these features can be constructed by only using 13 kernel features listed in Table 2 (c). The baseline parser Parser-ZN$^-$ additionally utilizes features in Table 2 (b), where d denotes the distance between the root nodes of s_0 and s_1, $s.v_r$ and $s.v_l$ are the numbers of the right and left modifiers of s, $s.\mathrm{rc}_2$ and $s.\mathrm{lc}_2$ are the second right- and left-most children of s. Note that some of the features described in Zhang and Nivre (2011), which are associated with dependency labels and head information of stack elements, are not included since our framework is based on unlabeled dependencies and the arc-standard strategy. The additional features for Parser-ZN$^-$ require the features in Table 2 (d) to be added into the set of kernel features.

(a)	$s_0.w$	$s_0.t$	$s_0.w \circ s_0.t$		
	$s_1.w$	$s_1.t$	$s_1.w \circ s_1.t$		
	$q_0.w$	$q_0.t$	$q_0.w \circ q_0.t$		
	$s_0.w \circ s_1.w$		$s_0.t \circ s_1.t$		
	$s_0.t \circ q_0.t$		$s_0.w \circ s_0.t \circ s_1.t$		
	$s_0.t \circ s_1.w \circ s_1.t$		$s_0.w \circ s_1.w \circ s_1.t$		
	$s_0.w \circ s_0.t \circ s_1.w$		$s_0.w \circ s_0.t \circ s_1.w \circ s_1.t$		
	$s_0.t \circ q_0.t \circ q_1.t$		$s_1.t \circ s_0.t \circ q_0.t$		
	$s_0.w \circ q_0.t \circ q_1.t$		$s_1.t \circ s_0.w \circ q_0.t$		
	$s_1.t \circ s_1.\mathrm{rc}.t \circ s_0.t$		$s_1.t \circ s_1.\mathrm{lc}.t \circ s_0.t$		
	$s_1.t \circ s_1.\mathrm{rc}.t \circ s_0.w$		$s_1.t \circ s_1.\mathrm{lc}.t \circ s_0.w$		
	$s_1.t \circ s_0.t \circ s_0.\mathrm{rc}.t$		$s_1.t \circ s_0.w \circ s_0.\mathrm{lc}.t$		
	$s_2.t \circ s_1.t \circ s_0.t$				
(b)	$s_0.w \circ d$	$s_0.t \circ d$	$s_1.w \circ d$	$s_1.w \circ d$	
	$s_0.w \circ s_0.v_l$		$s_0.t \circ s_0.v_l$		
	$s_1.w \circ s_1.v_r$		$s_1.t \circ s_1.v_r$		
	$s_1.w \circ s_1.v_l$		$s_1.t \circ s_1.v_l$		
	$s_0.\mathrm{lc}.w$	$s_0.\mathrm{lc}.t$	$s_1.\mathrm{rc}.w$	$s_1.\mathrm{rc}.t$	
	$s_1.\mathrm{lc}.w$	$s_1.\mathrm{lc}.t$	$s_0.\mathrm{lc}_2.w$	$s_0.\mathrm{lc}_2.t$	
	$s_1.\mathrm{rc}_2.w$	$s_1.\mathrm{rc}_2.t$	$s_1.\mathrm{lc}_2.w$	$s_1.\mathrm{lc}_2.t$	
	$s_0.t \circ s_0.\mathrm{lc}.t \circ s_0.\mathrm{lc}_2.t$		$s_1.t \circ s_1.\mathrm{rc}.t \circ s_1.\mathrm{rc}_2.t$		
	$s_1.t \circ s_1.\mathrm{lc}.t \circ s_1.\mathrm{lc}_2.t$				
(c)	j	$s_2.t$	$q_0.w$	$q_0.t$	$q_1.t$
	$s_1.w$	$s_1.t$	$s_1.\mathrm{rc}.t$	$s_1.\mathrm{lc}.t$	
	$s_0.w$	$s_0.t$	$s_0.\mathrm{rc}.t$	$s_0.\mathrm{lc}.t$	
(d)	d	$s_0.v_l$	$s_1.v_l$	$s_1.v_r$	
	$s_0.\mathrm{lc}.w$	$s_1.\mathrm{rc}.w$	$s_1.\mathrm{lc}.w$		
	$s_0.\mathrm{lc}_2.w$	$s_1.\mathrm{rc}_2.w$	$s_1.\mathrm{lc}_2.w$		
	$s_0.\mathrm{lc}_2.t$	$s_1.\mathrm{rc}_2.t$	$s_1.\mathrm{lc}_2.t$		

Table 2: (a) Feature templates for Parser-HS; (b) Additional feature templates for Parser-ZN$^-$; (c) Kernel features for Parser-HS; (d) Additional kernel features for Parser-ZN$^-$.

2.2.4 Beam search with DP

In the shift-reduce parsing with dynamic programming, we cannot simply apply beam search as in a non-DP shift-reduce parsing, because each state does not have a unique score any more. To decide the ordering of states within the beam, the concept of *prefix score* and *inside score* (Stolcke, 1995) is adopted. The prefix score ξ is the total score of the best action sequence from the initial state to the current state, while the inside score η

is the score of the tree on the top of the stack. With these scores and a set of predictor states $\Pi(\psi)$ of state ψ, the full description of state ψ takes the form $\psi : \langle \ell, i, j, S; \xi, \eta, \Pi \rangle$. The calculation of the prefix and inside scores is described in Huang and Sagae (2010). By using these scores, the ordering of states is defined as

$$\langle \ell, \ldots; \xi, \eta, \_ \rangle \prec \langle \ell, \ldots; \xi', \eta', \_ \rangle$$
$$\textit{iff } \xi < \xi' \vee (\xi = \xi' \wedge \eta < \eta'),$$

where "_" denotes "match anything".

3 Joint POS Tagging and Parsing Model

In this section, we describe our models that jointly solve POS tagging and dependency parsing, which are based on the shift-reduce parsers described in Section 2.2. Corresponding to the two baseline parsers Parser-HS and Parser-ZN$^-$, we investigate two joint models: Joint-HS$^+$ and Joint-ZN$^-$. Although the latter uses a richer set of features, the formers can take more advantage of DP because a compact representation of features results in more frequent state packing.

3.1 POS Tagging with Modified Shift Action

Our joint parsers incorporate POS tagging during the course of shift-reduce parsing, by modifying the SHIFT action so that it assigns a tag to the word when it is shifted:

- SHIFT(t) (SH(t)): move the head of the queue, q_0, onto the stack, and assign tag t to it.

Along with REDUCE-LEFT/RIGHT actions, our joint model utilizes a total of $n+2$ actions, where n is the number of tags in the given dataset ($n = 33$ for the CTB-5 tag set (Xia, 2000)). A trace of an example joint parsing is illustrated in Figure 2.

3.2 Training and Decoding

We formulate the task of POS tagging and dependency parsing in a joint framework: given an input segmented sentence x, the model tries to find the best output y that satisfies:

$$\tilde{y} = \underset{y \in \mathcal{Y}(x)}{\operatorname{argmax}} \vec{w} \cdot \vec{\theta}(y),$$

where $\mathcal{Y}(x)$ is a set of possible outputs for x, $\vec{w}$ is the global feature vector, and $\vec{\theta}(y)$ is the feature vector of y. As in the baseline parsers, we train our models with the averaged perceptron; the beam search and early update strategy is almost the same except that the update is now caused by an error in POS tagging as well as by an error in dependency parsing. Similarly to the baseline tagger, we use the tag dictionary and closed-set tags to prune unlikely tags during decoding.

(a)	$q_0.t$	$q_0.w \circ q_0.t$	$s_0.t \circ q_0.t$	
	$s_0.t \circ q_0.t \circ q_1.t$		$s_1.t \circ s_0.t \circ q_0.t$	
	$s_0.w \circ q_0.t \circ q_1.t$		$s_1.t \circ s_0.w \circ q_0.t$	
(b)	$t \circ s_0.w$		$t \circ s_0.t$	
	$t \circ s_0.w \circ q_0.w$		$t \circ s_0.t \circ q_0.w$	
	$t \circ B(s_0.w) \circ q_0.w$		$t \circ E(s_0.w) \circ q_0.w$	
	$t \circ s_0.t \circ s_0.\text{rc}.t$		$t \circ s_0.t \circ s_0.\text{lc}.t$	
	$t \circ s_0.w \circ s_0.t \circ s_0.\text{rc}.t$		$t \circ s_0.w \circ s_0.t \circ s_0.\text{lc}.t$	
(c)	j	$s_2.t$ $q_0.w$	$q_{-1}.t$	$q_{-2}.t$
	$s_1.w$	$s_1.t$	$s_1.\text{rc}.w$	$s_1.\text{lc}.t$
	$s_0.w$	$s_0.t$	$s_0.\text{rc}.w$	$s_0.\text{lc}.t$

Table 3: (a) List of delayed features for the joint parsers. (b) Syntactic features for the joint parsers, where t is the POS tag to be assigned to q_0. (c) Kernel features for the joint parser Joint-HS$^+$.

3.3 Features

For the features of the models, we incorporate the union of the features in the baseline tagger and the baseline parsers; features from Parser-HS are used for Joint-HS, and features from Parser-ZN$^-$ for Joint-ZN$^-$. Furthermore, we additionally incorporate a set of *syntactic features* for POS tagging that capture dependencies between syntactic elements in the stack and the POS to be tagged (described in Section 3.3.1).

The features for the baseline tagger (shown in Table 1) and Parser-ZN$^-$ (shown in Table 2 (b)) can be used *in situ*, because they do not rely on look-ahead POS tags (i.e. POS tags of the words in the queue). However, it is not straightforward to incorporate the features for Parser-HS: in the joint framework, since the look-ahead POS tags are unavailable when the model tries to determine the next action, we cannot easily incorporate those features that include look-ahead POS (listed in Table 3 (a)). In order to deal with this issue, we introduce a concept of *delayed features*, which enables the model to incorporate the look-ahead information by delayed evaluation of feature scores (described in Section 3.3.2).

Note that the features from the baseline parsers are used for all actions (i.e. SHIFT(t) and REDUCE-LEFT/RIGHT) while the features from the tagger are only used for SHIFT(t) actions in the joint models. The addition of the tagging features requires a few new elements to be added into the set of kernel features; the new set of kernel features for Joint-HS$^+$ is shown in Table 3 (c).

step	action	stack S	queue Q	translation
0	-	ϕ	我/? 想/? 把/? ...	
1	SH(PN)	我/PN	想/? 把/? 这/? ...	我: "I"
2	SH(VV)	我/PN 想/VV	把/? 这/? 个/? ...	想: "want"
3	SH(BA)	我/PN 想/VV 把/BA	这/? 个/? 句子/? ...	把: *object marker*
4	SH(DT)	我/PN 想/VV 把/BA 这/DT	个/? 句子/? 翻译/? ...	这: "this"
5	SH(M)	我/PN 想/VV 把/BA 这/DT 个/M	句子/? 翻译/? 成/? ...	个: *quantifier*
6	RL	我/PN 想/VV 把/BA 这/DT⌢[个/M]	句子/? 翻译/? 成/? ...	
7	SH(NN)	我/PN 想/VV 把/BA 这/DT⌢[个/M] 句子/NN	翻译/? 成/? 英语/?	句子: "sentence"
8	RR	我/PN 想/VV 把/BA [这/DT⌢[...]]⌢句子/NN	翻译/? 成/? 英语/?	
9	RL	我/PN 想/VV 把/BA⌢[[...]⌢句子/NN]	翻译/? 成/? 英语/?	
10	SH(VV)	我/PN 想/VV 把/BA⌢[[...]⌢句子/NN] 翻译/VV	成/? 英语/?	翻译: "translate"

Figure 2: A trace of joint shift-reduce parsing for "我想把这个句子翻译成英语" ("*I want to translate this sentence into English.*"), where grandchildren of stack elements are omitted.

Specifically, $q_{-1}.t$ and $q_{-2}.t$ are added in order to accommodate some of the tagging features, while $q_0.t$ and $q_1.t$ are removed because the look-ahead POS tags are not available when the equivalence of the states are evaluated. Joint-ZN$^-$ additionally requires kernel features in Table 2 (d).

3.3.1 Syntactic Features

Since our joint framework performs tagging and parsing simultaneously, we can think of incorporating a combined feature of the next tag (to be assigned to q_0) with syntactic information from stack elements, which cannot be used in an n-gram POS tagger. Specifically, we propose to use the features shown in Table 3 (b). Intuitively, these features try to capture dependencies between the POS to be assigned and syntactic structure encoded in the trees being built in the stack. For example, at step 9 in Figure 2, the next word 翻译 can be either a noun or a verb. In determining the tag of this word to be VV, the existence of the preceding phrase "把/BA [...]" on the top of the stack plays an important role, because the phrase headed by 把 represents an object for the following verb; in contrast, in an n-gram POS tagger, capturing this information is not easy because 把/BA is located at a distance of four words. Note that the addition of those syntactic features does not require the addition of any elements to the set of kernel features.

3.3.2 Delayed Features

A challenge in the incremental joint approach is that since the shift-reduce model processes an input sentence in a left-to-right manner, it cannot exploit look-ahead POS tags, which a pipeline shift-reduce parser can consider, to determine the next action. In our experiment with Parser-HS, the ablation of the features including look-ahead POS results in 0.67% decrease in parsing performance on the development set, suggesting that the look-ahead POS information is indispensable to achieve the state-of-the-art performance. In order to relieve this problem, we introduce a concept of *delayed features*, which are a set of features that are evaluated later when certain information becomes available. In our model, the parser features that require look-ahead POS information are defined as the delayed features, and shall later be evaluated at the step when the look-ahead POS are determined.

Let us see an example in Figure 2. At step 2, a parser encounters a shift-reduce conflict: the next action can be any of REDUCE-LEFT/RIGHT and SHIFT(t). If this were a (non-joint) shift-reduce parser, the model can utilize the look-ahead POS information by features such as

$$(s_0.t = \text{VV}) \circ (s_1.t = \text{PN}) \circ (q_0.t = \text{BA}),$$

to determine the next action, because the POS of all words in the sentence are already given. However, in the joint parser, the POS of the first word in the queue, 把, remains undetermined until the word is shifted. To deal with this, we define a *delayed feature* that takes look-ahead POS tag(s) as argument(s), as in

$$(s_0.t = \text{VV}) \circ (s_1.t = \text{PN}) \circ (q_0.t = \lambda_1), \lambda_1 = w_2.t.$$

At step 3, after SHIFT(BA) is performed, the delayed features from the previous step becomes a non-delayed feature

$$(s_0.t = \text{VV}) \circ (s_1.t = \text{PN}) \circ (q_0.t = \text{BA}),$$

which can be evaluated in the same way as normal (non-delayed) features.

More formally, each state carries with it a set of delayed feature vectors $\langle \vec{d}_1, \vec{d}_2 \rangle$, where $\vec{d}_n$ is the n-th order delayed feature vector, which has n ar-

guments to be filled in[1]. At each step, a REDUCE-LEFT/RIGHT action a adds a set of delayed features to the delayed feature vectors of state ψ:

$$\langle \vec{d}_1, \vec{d}_2 \rangle \leftarrow \langle \vec{d}_1 + \vec{\Phi}_1(\psi, a), \vec{d}_2 + \vec{\Phi}_2(\psi, a) \rangle,$$

where $\vec{\Phi}_1(\psi, a)$ and $\vec{\Phi}_2(\psi, a)$ are the first-/second-order delayed features generated by action a being applied to ψ. When a SHIFT(t) action is performed, the model fills in the argument in the delayed features with the newly-assigned tag t, as well as adding new delayed features it generates:

$$\langle \vec{d}_1, \vec{d}_2 \rangle \leftarrow \langle \vec{\Phi}_1(\psi, \text{SH}(t)) + \mathcal{T}(t, \vec{d}_2), \vec{\Phi}_2(\psi, \text{SH}(t)) \rangle,$$

where $\mathcal{T}(t, \vec{d}_2)$ is the resulting feature vector after tag t is filled in to the first argument of the features in $\vec{d}_2$. Note that action SH(t) also adds $\vec{d}_0 = \mathcal{T}(t, \vec{d}_1)$ to its (non-delayed) feature vector.

Note that the above formulation with the delayed features is equivalent to the model with full look-ahead features if the exact decoding is performed. Although the approximate search with beam takes the risk of the gold derivation falling off the beam before delayed features are evaluated, we show in Section 4 that the current solution works well in practice.

3.4 Deduction System with DP

With the delayed features, a parser state ψ takes the form of $\langle \ell, i, j, S, \vec{d}_1, \vec{d}_2; \xi, \eta, \Pi \rangle$. Now, if two equivalent states are still merged according to Eq. (1), one state might have multiple sets of delayed feature vectors depending on the previous action sequences. In order to make the joint model still tractable with the DP formalism, we modify the equivalence condition in Eq. (1): in addition to the condition in Eq. (1), two states now need to share the same delayed feature vectors in order for them to be merged:

$$j = j' \wedge \vec{f}(j, S) = \vec{f}(j', S') \wedge \vec{d}_1 = \vec{d}'_1 \wedge \vec{d}_2 = \vec{d}'_2.$$

This guarantees that a parser state has only one unique set of delayed feature vectors.

We can prove by induction that the *correctness* (i.e. the optimality of the deductive system) is still assured (proof omitted due to limited space) even with the delayed features incorporated. However, because any number of REDUCE-LEFT/RIGHT actions can occur between two SHIFT actions, the delayed features might need to refer to unboundedly deep elements from stack trees; therefore, the *boundedness* (see Huang and Sagae (2010)) of the kernel features no longer holds and the worst-case polynomial complexity is not assured. Nonetheless, we show that our models work sufficiently well in practice, with the aid of beam search.

[1] In our joint models, the use of only the first- and second-order delayed vectors suffices, because the feature templates refer to the tags of the first two words in the queue at most.

4 Experiment

4.1 Experimental Settings

We evaluate the performance of our joint parsers and baseline models on the Chinese Penn Treebank (CTB) 5 dataset. We use the standard split of CTB-5 described in Duan et al. (2007) and the head-finding rules in Zhang and Clark (2008b).

We iteratively train each of the models and choose the best model, in terms of the tagging accuracy (for tagger) or word-level dependency accuracy (for parsers and joint parsers) on the development set, to use in the final evaluation. When building a tag dictionary, we discarded instances that appear less than three times (tuned on the development set) in the training data. An Intel Core-i7 950 3.2GHz machine is used for evaluation.

4.2 Baseline Performance

First of all, we evaluate the performance of our baseline tagger and parsers described in Section 2. Based on our preliminary experiments, we set the beam size to 16 for the baseline tagger and Parser-HS, and to 32 for Parser-ZN$^-$. Our baseline tagger achieved a tagging accuracy of 94.15% on the development set, and 93.82% on the test set. Since most recent works on Chinese POS tagging (e.g. Kruengkrai et al. (2009); Sun (2011)) are joint approaches integrating word segmentation, the only directly-comparable work we could find is Li et al. (2011), where they built a perceptron-based POS tagger with the same feature set as we used (Zhang and Clark, 2008a). They reported 93.51% accuracy on test set, which is slightly lower than ours.

The upper part of Table 7 shows the performance of our baseline parsers with a comparison to other state-of-the-art parsers, where (unlabeled) attachment accuracies of word, root, and complete match are shown (with punctuations excluded). Our reimplementation of Huang and Sagae (2010) has reproduced almost the same accuracy. Interestingly, Parser-ZN$^-$ also has comparable performance to that of Zhang and Nivre (2011) even though we could not use some of their features (as described in Section 3.3).

beam	Joint-HS$^+$			Joint-ZN$^-$		
	tag	dep	speed	tag	dep	speed
4	94.37	79.98	37.5	94.26	80.55	25.4
8	94.57	80.56	19.3	94.64	81.47	13.5
16	94.56	80.97	10.1	94.48	81.50	7.0
32	94.66	80.72	4.7	94.40	81.68	3.3
64	94.50	81.09	2.0	94.50	81.88	1.5
128	-	-	-	94.43	81.89	0.69

Table 4: Tagging and word-level dependency accuracies and parsing speed (in sentence/second) on the development set with respect to beam size.

Model	tag	word	non-root	root	compl.
Parser-HS	(100)	85.15	85.61	75.76	34.50
Parser-ZN$^-$		85.77	86.18	77.46	34.99
Pipeline-HS	94.15	78.10	78.49	70.03	26.77
Pipeline-ZN$^-$		78.67	78.92	73.32	27.90
Joint-HS$^+$	94.56*	80.97*	81.32	73.64	29.51
Joint-ZN$^-$	94.50*	81.88*	82.21	74.94	30.26

Table 5: Development result of the proposed models. Joint-HS$^+$/ZN$^-$ perform better than Pipeline-HS/ZN$^-$ in terms of both tagging and word-level dependency accuracies, with statistical significance of $p < 0.05$ (denoted by *) by MeNemar's test.

4.3 Development Results

Table 4 shows the tagging and word-level dependency accuracies of the joint models with respect to the beam size, where "tag" and "dep" show the tagging and word-level dependency accuracies, and "speed" is the joint parsing speed (in sentence per second). Based on this experiment, we use the beam size of 16 for Joint-HS$^+$ and 64 for Joint-ZN$^-$ in the following experiments.[2] The best dependency accuracies are achieved after 36-th and 31-st iterations, respectively.

Table 5 shows the performance of the baseline and joint models on the development set, where "Pipeline-HS" and "Pipeline-ZN$^+$" are the pipeline combinations of the baseline tagger with Parser-HS and Parser-ZN$^+$, respectively. Joint-HS$^+$ and JointZN$^-$ have 0.35–0.41% (tagging) and 2.87–3.21% (word-level dependency) higher accuracies than the pipeline models.

Table 6 shows feature ablation results on the development set, where "wo/delay", "wo/dp", and "wo/syn" correspond to the models that do not use the delayed features, dynamic programming, and syntactic features, respectively. Overall, the tagging accuracies are only slightly affected by the ablation of these features (with differences no larger than 0.10%), while the parsing accuracies decreased in most settings. The ablation of the delayed features resulted in 0.26% and 0.16% decreases of word-level dependency accuracies for Joint-HS$^+$ and Joint-ZN$^-$, showing the effectiveness of these features. The contribution of the dynamic programming is clearly shown for Joint-HS$^+$ with 0.22% improvement in dependency accuracy, although no meaningful effect for Joint-ZN$^-$ is confirmed; this is probably because the use of richer features results in less frequent packing of states. Lastly, the ablation of the syntactic features results in as much as 0.60% and 0.23% decreases of dependency accuracies for Joint-HS$^+$ and Joint-ZN$^-$. As opposed to our first expectation, the syntactic features made little effect on the tagging accuracies; on the contrary, the result suggests that capturing the dependencies between the stack elements and the next word's tag is quite effective to improve parsing accuracy.

[2]Due to limited time, the beam size of 32 is used for Joint-ZN$^-$ for the feature ablation experiment shown in Table 6.

Model		default	wo/delay	wo/dp	wo/syn
Joint-HS$^+$	tag	94.56	$\pm$0.00	$-$0.06	$+$0.04
	dep	80.97	$-$0.26	$-$0.22	$-$0.60
Joint-ZN$^-$	tag	94.40	$+$0.10	$+$0.05	$-$0.07
	dep	81.68	$-$0.16	$-$0.01	$-$0.23

Table 6: Feature ablation results for the joint models on the development set.

Model	tag	word	root	compl.	speed
Huang+ '10		85.20	78.32	33.72	-
Zhang+ '11		86.0	-	36.9	-
Li-11-2nd	(100)	86.18	78.58	34.02	5.8
Parser-HS		85.12	78.30	32.77	32.7
Parser-ZN$^-$		85.96	80.87	35.03	9.0
Li-11(v2,3rd)	92.80	80.79	75.84	29.11	0.3
Li-11(v1,3rd)	92.89	80.69	75.90	29.06	0.5
Li-11(v1,2nd)	93.08	80.74	75.80	28.24	1.7
Pipeline-HS	93.82	77.13	72.59	25.13	32.7†
Pipeline-ZN$^-$		78.04	75.55	26.07	9.0†
Joint-HS$^+$	**94.01***	79.83*	73.86	27.85	9.5
Joint-ZN$^-$	93.94	**81.33***	**77.93**	**29.90**	1.5

Table 7: Final result of the proposed model and the baseline. * denotes the statistical significance over the corresponding pipeline model ($p < 0.05$). †Only the parsing speed is shown; the tagging speed was 210.6 sentence/sec.

4.4 Final Results

Table 7 shows the final result of the proposed models compared to the baseline models. "Li-

error pattern	#↓	total	error pattern	#↑	total
NN → VV	61	169	VV → NN	29	128
DEC → DEG	35	65	NN → NR	16	64
DEG → DEC	19	72	JJ → NN	14	62
NN → JJ	11	59	VA → VV	8	12
P → CC	8	13	JJ → NR	6	2
P → VV	8	18	NR → JJ	6	4

Table 8: POS tagging error patterns that decrease (left side) and increase (right side) by joint decoding (on dev. set). The numbers of errors made by the baseline tagger ("total") and the increases and decreases by Joint-ZN$^+$ (#↓ and #↑) are shown.

11(. . .)" shows the graph-based models by Li et al. (2011), where v1/2 and 2nd/3rdcorrespond to their version 1/2 and second-/third-order models. The joint models Joint-HS$^+$ and Joint-ZN$^-$ achieve improvements of 0.19% and 0.12% in tagging accuracy over the baseline tagger, and 2.70% and 3.29% improvements in word-level dependency accuracy over the pipeline models, showing the effectiveness of the joint approach. Furthermore, the tagging and parsing accuracies of Joint-ZN$^-$ surpass the graph-based models by Li et al. (2011), achieving the new state-of-the-art performance on this joint task. Since our framework is at least comparable in speed to their models, these results suggest that our incremental framework is suitable to this joint task.

4.5 Discussion and Analysis

Table 8 shows the increase and decrease of error patterns of Joint-ZN$^-$ over the baseline tagger. Notably, the joint model has a clear advantage in the disambiguation of DEC and DEG and the discrimination of NN from VV. While these tags are those that critically influence the overall syntactic structure, the increased error patterns include those tags that are considered less important[3] in deciding the syntactic structure (e.g. NN/NR: general/proper nouns); this observation is largely similar to those reported by Li et al. (2011).

It is noteworthy that we obtained the first positive result that the joint decoding does improve POS tagging, while, in contrast, Li et al. (2011) have reported that the joint decoding has negative effect on the tagging accuracy: their third-order models have 0.6–0.7% lower tagging accuracies than their baseline tagger. When comparing our error patterns with those of their model, although the overall increase and decrease of the error patterns look largely similar, our model has a relatively smaller number of increased error patterns than the decreased ones. Therefore, by selectively improving syntactically-important tags, our joint model is considered to have improved the POS tagging accuracy over the baseline tagger.

[3]although VV → NN errors look like an exceptional case

5 Related Works

In recent years, joint segmentation and tagging have been widely investigated (e.g. Zhang and Clark (2010); Kruengkrai et al. (2009); Zhang and Clark (2008a); Jiang et al. (2008a); Jiang et al. (2008b)). Particularly, our framework of using a single perceptron to solve the joint problem is motivated by Zhang and Clark (2008a). Also, our joint parsing framework is an extension of Huang and Sagae (2010)'s framework, which is described in detail in Section 2.2. In constituency parsing, the parsing naturally involves the POS tagging since the non-terminal symbols are commonly associated with POS tags (e.g. Klein and Manning (2003)). Rush et al. (2010) proposed to use dual composition to combine a constituency parser and a trigram POS tagger, showing the effectiveness of taking advantage of these two systems.

In dependency parsing, Lee et al. (2011) recently proposed a discriminative graphical model that solves morphological disambiguation and dependency parsing jointly. However, their main focus was to capture interaction between morphology and syntax in morphologically-rich, highly-inflected languages (such as Latin and Ancient Greek), which are unlike Chinese. More recently, Li et al. (2011) proposed the first joint model for Chinese POS tagging and dependency parsing in a graph-based parsing framework, which is one of our baseline systems. On the other hand, our work is the first incremental approach to this joint task.

6 Conclusion

In this paper, we have presented the first joint approach that successfully solves POS tagging and dependency parsing on an incremental framework. The proposed joint models outperform the pipeline models in terms of both tagging and dependency parsing accuracies, and our best model achieved the new state-of-the-art performance on this joint task, while retaining competitive parsing speed. Although we mainly focused on Chinese in this work, our framework is generally applicable to other languages including English; for future work, we hope to further investigate the effectiveness of our joint approach in those languages.

References

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*.

Xiangyu Duan, Jun Zhao, and Bo Xu. 2007. Probabilistic parsing action models for multilingual dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*.

Lyn Frazier and Keith Rayner. 1982. Making and correcting errors during sentence comprehension: Eye movements in the analysis of structurally ambiguous sentences. *Cognitive Psychology*, 14:178–210.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL*.

Wenbin Jiang, Liang Huang, Qun Liu, and Yajuan Lu. 2008a. A cascaded linear model for joint Chinese word segmentation and part-of-speech tagging. In *Proceedings of ACL/HLT*.

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

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL*.

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 *ACL*, Proceedings of ACL.

John Lee, Jason Naradowsky, and David A. Smith. 2011. A discriminative model for joint morphological disambiguation and dependency parsing. In *Proceedings of ACL*.

Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Haizhou. 2011. Joint models for Chinese POS tagging and dependency parsing. In *Proceedings of EMNLP*.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Comput. Linguist.*, 34:513–553.

Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of EMNLP*.

Andreas Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21:165–201.

Weiwei Sun. 2011. A stacked sub-word model for joint chineseword segmentation and part-of-speech tagging. In *Proceedings of ACL*.

Masaru Tomita. 1991. *Generalized LR Parsing*. Kluwer Academic Publishers.

Fei Xia. 2000. The part-of-speech tagging guidelines for the penn chinese treebank (3.0). Technical Report IRCS-00-07, University of Pennsylvania Institute for Research in Cognitive Science Technical Report, October.

Yue Zhang and Stephen Clark. 2008a. Joint word segmentation and POS tagging using a single perceptron. In *Proceedings of ACL-08: HLT*.

Yue Zhang 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*.

Yue Zhang and Stephen Clark. 2010. A fast decoder for joint word segmentation and POS-tagging using a single discriminative model. In *Proceedings of EMNLP*.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of ACL-2011 (short papers)*.