Joint Optimization for Chinese POS Tagging and Dependency Parsing

Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, and Wenliang Chen

***Abstract*—Dependency parsing has gained more and more interest in natural language processing in recent years due to its simplicity and general applicability for diverse languages. Previous work demonstrates that part-of-speech (POS) is an indispensable feature in dependency parsing since pure lexical features suffer from serious data sparseness problem. However, due to little morphological changes, Chinese POS tagging has proven to be much more challenging than morphology-richer languages such as English (94% vs. 97% on POS tagging accuracy). This leads to severe error propagation for Chinese dependency parsing. Our experiments show that parsing accuracy drops by about 6% when replacing manual POS tags of the input sentence with automatic ones generated by a state-of-the-art statistical POS tagger. To address this issue, this paper proposes a solution by jointly optimizing POS tagging and dependency parsing in a unique model. We propose for our joint models several dynamic programming based decoding algorithms which can incorporate rich POS tagging and syntactic features. Then we present an effective pruning strategy to reduce the search space of candidate POS tags, leading to significant improvement of parsing speed. Experimental results on two Chinese data sets, i.e. Penn Chinese Treebank 5.1 and Penn Chinese Treebank 7, demonstrate that our joint models significantly improve both the state-of-the-art tagging and parsing accuracies. Detailed analysis shows that the joint method can help resolve syntax-sensitive POS ambiguities $\{\mathsf{NN}, \mathsf{VV}\}$. In return, the POS tags become more reliable and helpful for parsing since the syntactic features are used in POS tagging. This is the fundamental reason for the performance improvement.**

***Index Terms*—Dependency parsing, dynamic programming, joint models, part-of-speech tagging.**

Manuscript received November 12, 2012; revised March 22, 2013; accepted June 27, 2013. Date of publication November 01, 2013; date of current version December 10, 2013. The work of Z. Li, M. Zhang, and W. Chen was supported by the National Natural Science Foundation of China (Grant No. 61203314, 61373095). The work of W. Che and T. Liu was supported by the National Natural Science Foundation of China (Grant No. 61133012), the National "863" Major Projects (2011AA01A207), and the National "863" Leading Technology Research Project (2012AA011102). The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Gokhan Tur. *(Corresponding author: T. Liu)*.

Z. Li is with the School of Computer Science and Technology, Soochow University, Suzhou 215006, China, and also with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China (e-mail: zhenghualiir@gmail.com).

M. Zhang and W. Chen are with the School of Computer Science and Technology, Soochow University, Suzhou 215006, China (e-mail: zhangminmt@hotmail.com; chenwenliang@gmail.com).

W. Che and T. Liu are with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China (e-mail: car@ir.hit.edu.cn; tliu@ir.hit.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASLP.2013.2288081

I. Introduction

Given an input sentence of n words, denoted by $\mathbf{x} = w_1 \dots w_n$, part-of-speech (POS) tagging aims to find an optimal tag sequence $\mathbf{t} = t_1 \dots t_n$, where $t_i \in \mathcal{T}$ $(1 \leq i \leq n)$ and $\mathcal{T}$ is a predefined tag set. POS tags are designed to represent word classes so that words of the same POS tag play a similar role in syntactic structures. The size of $\mathcal{T}$ is usually much less than the vocabulary size. Typically, POS tagging is treated as a sequence labeling problem, and has been previously addressed by machine learning algorithms such as maximum-entropy (ME) [1], conditional random fields (CRF) [2], and perceptron [3]. Fig. 1 shows an example sentence from Penn Chinese Treebank 5.1 (CTB5). The lowest three rows present the n-best POS tags for each word produced by a state-of-the-art CRF-based tagger. Looking at the 1-best POS tags, we can see that the CRF model makes four errors, i.e. $\mathsf{de/DEC} \rightarrow \mathsf{DEG}$, $\mathsf{ouwen/NR} \rightarrow \mathsf{NN}$, $\mathsf{xiaoli/VV} \rightarrow \mathsf{NN}$, and $\mathsf{liwupudui/NR} \rightarrow \mathsf{NN}$. In fact, syntax-sensitive POS ambiguities, such as $\{\mathsf{NN}, \mathsf{VV}\}$ and $\{\mathsf{DEC}, \mathsf{DEG}\}$, require long-distance syntactic knowledge to resolve and are very difficult for the sequential labeling models.

Dependency parsing maps a natural language sentence into a structural dependency tree conforming to a predefined dependency grammar, as depicted in Fig. 1. A dependency tree is denoted by $\mathbf{d} = \{(h, m, l) : 0 \leq h \leq n, 1 \leq m \leq n, l \in \mathcal{L}\}$, where (h, m, l) means a dependency from the *head word* (also called *father*) w_h to the *modifier* (also called *child* or *dependent*) w_m with a dependency label l, and $\mathcal{L}$ is the label set. Dependency labels are used to indicate the syntactic or semantic relation between the two words. For instance, the dependency (8, 6, SUB) in Fig. 1 means ouwen is the *subject* of xiaoli. w_0 is an artificial token which points to the head word of the sentence and is used to simplify the formalization of the problem. Since this paper focuses on unlabeled dependency parsing, we use (h, m) or $h \curvearrowright m$ interchangeably to represent a dependency arc without considering the label l.

Data-driven dependency parsing models make heavy use of POS tags to compose supporting features, since it leads to severe data sparseness problem if only lexical features are used. However, due to the lack of morphological clues, Chinese POS tagging turns out to be much more challenging than other morphology-richer languages such as English. The state-of-the-art POS tagging accuracy is about 94% for Chinese, which is much lower than 97% for English [3]. This causes a serious error propagation problem for Chinese dependency parsing. Our experimental results show that parsing accuracy drops by about 6% when using automatic POS tags instead of gold-standard ones

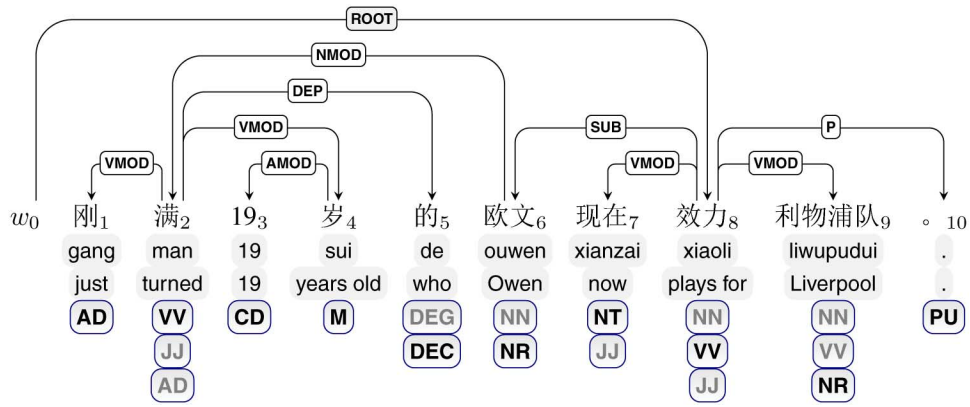

Fig. 1. A labeled dependency tree from CTB5. The Pinyin transcriptions and English translations are presented in the two rows below the Chinese sentence. We prune out the implausible POS tags according to the marginal probabilities (see Section III-C) and list the top three candidate POS tags in the lowest three rows (incorrect POS tags in grey color and correct ones in black color).

(see Table Vand VI in Section IV). For example in Fig. 1, due to the tagging error of xiaoli/VV → NN, our pipelined parsing model fails to recognize xiaoli as the predicate of the sentence and returns a fully unreasonable structure. However, this issue has not been well addressed in previous work. Previous research either simply adopts a pipeline approach by first POS tagging and then dependency parsing, or unrealistically overlooks this problem by using gold-standard POS tags for Chinese [4], [5], [6].

In this paper, we address this issue by jointly optimizing POS tagging and dependency parsing. Intuitively, joint inference of POS tagging and dependency parsing should be helpful to the two closely related individual tasks. On the one hand, syntactic information can help resolve some POS ambiguities which are difficult to handle by the sequential POS tagging models. On the other hand, more accurate POS tags are able to further improve dependency parsing. Take the tagging error of xiaoli/VV → NN in Fig. 1 as an example. Since xiaoli is an unknown word and the word contexts provide little clues, it is difficult for the sequential CRF-based tagger to correctly tag xiaoli as a verb VV instead of a noun NN. Differently, under the joint framework, the model may find that if xiaoli is tagged as a noun, the sentence would have no predicate and an undesirable dependency tree would be built; on the contrary, if xiaoli is tagged as a verb, the model could end up with a much more desirable syntactic structure. With such observations, the joint model may make the correct decision of xiaoli/VV. In summary, we make the following contributions:

- We propose several dynamic programming (DP) based decoding algorithms for our joint models which are capable of effectively incorporating rich POS tagging and parsing features on the one hand and efficiently searching out optimal joint results from the huge combinatorial hypothesis space on the other hand.
- We also present an effective marginal probability based pruning method to constrain the search space of the POS tag candidates of each word. Experiments show that our joint models achieve comparable parsing speeds to their pipeline counterparts.
- We experiment with the joint approach on two Chinese standard dataset, i.e. CTB5 and CTB7, showing that our joint models can significantly improve the state-of-the-art POS tagging and parsing accuracies.
- Detailed error analysis shows that the joint method can help resolve syntax-sensitive POS ambiguities. In return, the POS tags become more reliable and helpful for parsing. This is the fundamental reason for the performance improvements.

The remainder of this paper is organized as follows. Section II describes the pipeline method, including the POS tagging and parsing models. Section III presents the joint models, the decoding algorithms, and the method for POS tag pruning. Section IV reports the experimental results and error analysis. We review previous research closely related to this work in Section V, and conclude this paper in Section VI.

This paper is a substantial extension of our earlier work in [7]. We add new experimental results, a comprehensive description of the models, more details about our method, and in-depth analysis of the results.

II. The Pipeline Approach

The pipeline method treats POS tagging and dependency parsing as two cascaded problems. First, an optimal POS tag sequence $\hat{\mathbf{t}}$ is determined.

$$\hat{\mathbf{t}} = \arg\max_{\mathbf{t}} \mathsf{Score}_{\mathrm{pos}}(\mathbf{x}, \mathbf{t}) \quad (1)$$

where $\mathsf{Score}_{\mathrm{pos}}(\mathbf{x}, \mathbf{t})$ is the score of the POS tag sequence $\mathbf{t}$ for the input sentence $\mathbf{x}$. Then, an optimal dependency tree $\hat{\mathbf{d}}$ is determined based on $\mathbf{x}$ and $\hat{\mathbf{t}}$.

$$\hat{\mathbf{d}} = \arg\max_{\mathbf{d}} \mathsf{Score}_{\mathrm{syn}}(\mathbf{x}, \hat{\mathbf{t}}, \mathbf{d}) \quad (2)$$

where $\mathsf{Score}_{\mathrm{syn}}(\mathbf{x}, \hat{\mathbf{t}}, \mathbf{d})$ is the score of the dependency tree $\mathbf{d}$ given the input sentence $\mathbf{x}$ and the tag sequence $\hat{\mathbf{t}}$.

A. CRF-based POS Tagging

POS tagging is a typical sequence labeling problem. We adopt CRF to build our baseline POS tagger for two reasons. Firstly, our preliminary experiments show that the CRF-based tagger

TABLE I
POS TAGGING FEATURES USED IN BOTH TAGGING AND JOINT MODELS

POS Unigram Features: $\mathbf{f}_{\mathrm{pu}}(\mathbf{x}, i, t_i)$
01: $t_i \circ w_i$
02: $t_i \circ w_{i-1}$
03: $t_i \circ w_{i+1}$
04: $t_i \circ w_i \circ c_{i-1,-1}$
05: $t_i \circ w_i \circ c_{i+1,0}$
06: $t_i \circ c_{i,0}$
07: $t_i \circ c_{i,-1}$
08: $t_i \circ c_{i,k}, 0 < k < \#c_i - 1$
09: $t_i \circ c_{i,0} \circ c_{i,k}, 0 < k < \#c_i - 1$
10: $t_i \circ c_{i,-1} \circ c_{i,k}, 0 < k < \#c_i - 1$
11: **if** $\#c_i = 1$ **then** $t_i \circ w_i \circ c_{i-1,-1} \circ c_{i+1,0}$
12: **if** $c_{i,k} = c_{i,k+1}$ **then** $t_i \circ c_{i,k} \circ$ *"consecutive"*
13: $t_i \circ \mathsf{prefix}(w_i, k), 1 \le k \le 4, k \le \#c_i$
14: $t_i \circ \mathsf{suffix}(w_i, k), 1 \le k \le 4, k \le \#c_i$
POS Bigram Features: $\mathbf{f}_{\mathrm{pb}}(\mathbf{x}, i, t_{i-1}, t_i)$
15: $t_i \circ t_{i-1}$

outperforms both perceptron and ME based models. Secondly, the CRF-based tagger can produce marginal probabilities and therefore is used to prune the POS tags for the joint models (see Section III-C).

As a conditional log-linear probabilistic model, CRF defines the probability of a tag sequence as

$$P(\mathbf{t}|\mathbf{x}) = \frac{\exp(\mathsf{Score}_{\mathrm{pos}}(\mathbf{x}, \mathbf{t}))}{\sum_{\mathbf{t}'} \exp(\mathsf{Score}_{\mathrm{pos}}(\mathbf{x}, \mathbf{t}'))} \quad (3)$$

$$\begin{aligned}\mathsf{Score}_{\mathrm{pos}}(\mathbf{x}, \mathbf{t}) &= \mathbf{w}_{\mathrm{pos}} \cdot \mathbf{f}_{\mathrm{pos}}(\mathbf{x}, \mathbf{t}) \\ &= \sum_{1 \le i \le n} (\mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, i, t_i) \\ &\quad + \mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, i, t_{i-1}, t_i)) \quad (4)\end{aligned}$$

$$\mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, i, t_i) = \mathbf{w}_{\mathrm{pu}} \cdot \mathbf{f}_{\mathrm{pu}}(\mathbf{x}, i, t_i)$$

$$\mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, i, t_{i-1}, t_i) = \mathbf{w}_{\mathrm{pb}} \cdot \mathbf{f}_{\mathrm{pb}}(\mathbf{x}, i, t_{i-1}, t_i) \quad (5)$$

where $\mathbf{f}_{\mathrm{pos}}(\mathbf{x}, \mathbf{t})$ refers to the aggregated POS feature vector and $\mathbf{w}_{\mathrm{pos}}$ is the corresponding weight vector. Given the input $\mathbf{x}$, $\mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, i, t_i)$ denotes the score of tagging w_i as t_i, and $\mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, i, t_{i-1}, t_i)$ represents the score of tagging w_{i-1} as t_{i-1} and w_i as t_i. Correspondingly, two sets of POS features are incorporated, i.e. *POS unigram features* $\mathbf{f}_{\mathrm{pu}}(\mathbf{x}, i, t_i)$, and *POS bigram features* $\mathbf{f}_{\mathrm{pb}}(\mathbf{x}, i, t_{i-1}, t_i)$, and $\mathbf{w}_{\mathrm{pu/pb}}$ are their feature weights. We adopt the state-of-the-art features for Chinese POS tagging [1], [8], which are listed in Table I, where $\circ$ means string concatenation; $c_{i,k}$ denotes the k^{th} Chinese character of w_i; $c_{i,0}$ is the first Chinese character; $c_{i,-1}$ is the last Chinese character; $\#c_i$ is the total number of Chinese characters contained in w_i; $\mathsf{prefix/suffix}(w_i, k)$ denote the k-Character prefix/suffix of w_i.

We adopt the *exponentiated gradient* algorithm to learn the weight vector $\mathbf{w}_{\mathrm{pos}}$ [9]. During the test phase, we adopt the Viterbi algorithm to get the optimal tagging sequence for an input sentence.

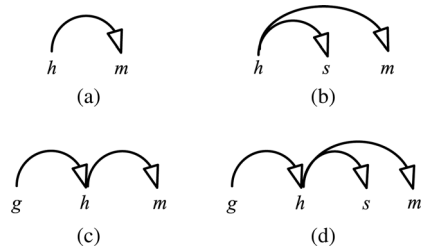

Fig. 2. Four types of scoring parts used in current graph-based models [10] (a) *single dependency* (b) *adjacent sibling* (c) *grandparent* (d) *grandparent-sibling*.

B. Graph-based Dependency Parsing

Recently, graph-based dependency parsing has gained more and more interest due to both its principled formalization of this problem, which allows the application of graph-related algorithms, and its state-of-the-art accuracy. Graph-based dependency parsing views the problem as finding the highest scoring tree from a directed graph. In order to facilitate an efficient and global search based on DP algorithms, graph-based models usually make strong independence assumptions that only some constrained dependencies are correlated with each other. Therefore, the score of a dependency tree is factored into scores of small parts (subtrees).

$$\begin{aligned}\mathsf{Score}_{\mathrm{syn}}(\mathbf{x}, \mathbf{t}, \mathbf{d}) &= \mathbf{w}_{\mathrm{syn}} \cdot \mathbf{f}_{\mathrm{syn}}(\mathbf{x}, \mathbf{t}, \mathbf{d}) \\ &= \sum_{p \subseteq \mathbf{d}} \mathsf{Score}_{\mathrm{part}}(\mathbf{x}, \mathbf{t}, p) \quad (6)\end{aligned}$$

where p is a scoring part which contains one or more dependencies in the dependency tree $\mathbf{d}$; $\mathsf{Score}_{\mathrm{part}}(\mathbf{x}, \mathbf{t}, p)$ denotes the weight contributed by p; $\mathbf{f}_{\mathrm{syn}}(\mathbf{x}, \mathbf{t}, \mathbf{d})$ is the aggregated syntactic feature vector corresponding to $(\mathbf{x}, \mathbf{t}, \mathbf{d})$ and $\mathbf{w}_{\mathrm{syn}}$ is the weight vector.

In the past few years, researchers try to weaken the independence assumptions made in earlier graph-based models, and incorporate more non-local features by incrementally enlarging the scoring subtrees. Although with the cost of efficiency, this effort is rewarding and the parsing accuracy improves [11], [12], [13], [10]. Fig. 2 shows different types of scoring parts used in current graph-based models. If all the subtrees in Fig. 2 are incorporated, the score of a dependency tree is rewritten as

$$\begin{aligned}\mathsf{Score}_{\mathrm{syn}}(\mathbf{x}, \mathbf{t}, \mathbf{d}) = &\sum_{\{(h,m)\} \subseteq \mathbf{d}} \mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, \mathbf{t}, h, m) \\ &+ \sum_{\{(h,s),(h,m)\} \subseteq \mathbf{d}} \mathsf{Score}_{\mathrm{sib}}(\mathbf{x}, \mathbf{t}, h, s, m) \\ &+ \sum_{\{(g,h),(h,m)\} \subseteq \mathbf{d}} \mathsf{Score}_{\mathrm{grd}}(\mathbf{x}, \mathbf{t}, g, h, m) \\ &+ \sum_{\{(g,h),(h,s),(h,m)\} \subseteq \mathbf{d}} \mathsf{Score}_{\mathrm{gsib}}(\mathbf{x}, \mathbf{t}, g, h, s, m) \quad (7)\end{aligned}$$

where $\mathsf{Score}_{\mathrm{dep/sib/grd/gsib}}(.)$ respectively denote the scores contributed by the four scoring parts in Fig. 2. Under a linear model, these score functions are defined as

TABLE II
SYNTACTIC FEATURES USED IN BOTH PARSING AND JOINT MODELS

Dependency Features: $\mathbf{f}_{\text{dep}}(\mathbf{x}, \mathbf{t}, h, m)$; r and d denote the direction and distance of $h \curvearrowright m$; b is an index between h and m.			
01: $w_h \circ t_h \circ \mathsf{r} \circ \mathsf{d}$	02: $w_h \circ \mathsf{r} \circ \mathsf{d}$	03: $t_h \circ \mathsf{r} \circ \mathsf{d}$	04: $m_h \circ t_m \circ \mathsf{r} \circ \mathsf{d}$
05: $w_m \circ \mathsf{r} \circ \mathsf{d}$	06: $t_m \circ \mathsf{r} \circ \mathsf{d}$	07: $w_h \circ t_h \circ w_m \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	08: $t_h \circ w_m \circ t_m \circ \mathsf{r} \circ \mathsf{d}$
09: $w_h \circ w_m \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	10: $w_h \circ t_h \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	11: $w_h \circ t_h \circ w_m \circ \mathsf{r} \circ \mathsf{d}$	12: $w_h \circ w_m \circ \mathsf{r} \circ \mathsf{d}$
13: $t_h \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	14: $w_h \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	15: $w_h \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	16: $t_h \circ t_{h+1} \circ t_{m-1} \circ t_m \circ \mathsf{r} \circ \mathsf{d}$
17: $t_h \circ t_{h+1} \circ t_m \circ t_{m+1} \circ \mathsf{r} \circ \mathsf{d}$	18: $t_{h-1} \circ t_h \circ t_{m-1} \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	19: $t_{h-1} \circ t_{h+1} \circ t_{m-1} \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	20: $t_{h-1} \circ t_h \circ t_{h+1} \circ t_m \circ \mathsf{r} \circ \mathsf{d}$
21: $t_h \circ t_{m-1} \circ t_m \circ t_{m+1} \circ \mathsf{r} \circ \mathsf{d}$	22: $t_h \circ t_{h+1} \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	23: $t_h \circ t_{m-1} \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	24: $t_h \circ t_m \circ t_{m+1} \circ \mathsf{r} \circ \mathsf{d}$
25: $t_h \circ t_b \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	26: $t_h \circ \#\mathsf{verb}(h, m) \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	27: $t_h \circ \#\mathsf{conj}(h, m) \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	28: $t_h \circ \#\mathsf{punc}(h, m) \circ t_m \circ \mathsf{r} \circ \mathsf{d}$
Sibling Features: $\mathbf{f}_{\text{sib}}(\mathbf{x}, \mathbf{t}, h, s, m)$; r and d denote the direction and distance of $h \curvearrowright m$.			
29: $t_h \circ t_s \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	30: $w_h \circ t_s \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	31: $t_h \circ w_s \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	32: $t_h \circ t_s \circ w_m \circ \mathsf{r} \circ \mathsf{d}$
33: $t_s \circ t_m \circ \mathsf{r} \circ \mathsf{d}$	34: $w_s \circ w_m \circ \mathsf{r} \circ \mathsf{d}$	35: $t_s \circ w_m \circ \mathsf{r} \circ \mathsf{d}$	36: $w_s \circ t_m \circ \mathsf{r} \circ \mathsf{d}$
37: $t_s \circ t_{s+1} \circ t_m \circ \mathsf{r}$	38: $t_{s-1} \circ t_s \circ t_m \circ \mathsf{r}$	39: $t_s \circ t_{m-1} \circ t_m \circ \mathsf{r}$	40: $t_s \circ t_m \circ t_{m+1} \circ \mathsf{r}$
41: $t_s \circ t_{s+1} \circ t_{m-1} \circ t_m \circ \mathsf{r}$	42: $t_{s-1} \circ t_s \circ t_{m-1} \circ t_m \circ \mathsf{r}$	43: $t_s \circ t_{s+1} \circ t_m \circ t_{m+1} \circ \mathsf{r}$	44: $t_{s-1} \circ t_s \circ t_m \circ t_{m+1} \circ \mathsf{r}$
Grandparent Features: $\mathbf{f}_{\text{grd}}(\mathbf{x}, \mathbf{t}, g, h, m)$; r denotes the direction of $h \curvearrowright m$ and r′ denotes the direction of $g \curvearrowright h$.			
45: $t_g \circ t_h \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	46: $w_g \circ t_h \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	47: $t_g \circ w_h \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	48: $t_g \circ t_h \circ w_m \circ \mathsf{r} \circ \mathsf{r}'$
49: $t_g \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	50: $w_g \circ w_m \circ \mathsf{r} \circ \mathsf{r}'$	51: $w_g \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	52: $t_g \circ w_m \circ \mathsf{r} \circ \mathsf{r}'$
53: $t_g \circ t_{m-1} \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	54: $t_g \circ t_m \circ t_{m+1} \circ \mathsf{r} \circ \mathsf{r}'$	55: $t_{g-1} \circ t_g \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	56: $t_g \circ t_{g+1} \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$
57: $t_{g-1} \circ t_g \circ t_{m-1} \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	58: $t_{g-1} \circ t_g \circ t_m \circ t_{m+1} \circ \mathsf{r} \circ \mathsf{r}'$	59: $t_g \circ t_{g+1} \circ t_{m-1} \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	60: $t_g \circ t_{g+1} \circ t_m \circ t_{m+1} \circ \mathsf{r} \circ \mathsf{r}'$
Grandparent-sibling Features: $\mathbf{f}_{\text{gsib}}(\mathbf{x}, \mathbf{t}, g, h, s, m)$; r denotes the direction of $h \curvearrowright m$ and r′ denotes the direction of $g \curvearrowright h$.			
61: $t_g \circ t_h \circ t_s \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	62: $w_g \circ t_h \circ t_s \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	63: $t_g \circ w_h \circ t_s \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	64: $t_g \circ t_h \circ w_s \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$
65: $t_g \circ t_h \circ t_s \circ w_m \circ \mathsf{r} \circ \mathsf{r}'$	66: $w_g \circ w_h \circ t_s \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	67: $t_g \circ t_s \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$	68: $w_g \circ t_s \circ t_m \circ \mathsf{r} \circ \mathsf{r}'$

$$
\begin{aligned}
\mathsf{Score}_{\text{dep}}(\mathbf{x}, \mathbf{t}, h, m) &= \mathbf{w}_{\text{dep}} \cdot \mathbf{f}_{\text{dep}}(\mathbf{x}, \mathbf{t}, h, m) \\
\mathsf{Score}_{\text{sib}}(\mathbf{x}, \mathbf{t}, h, s, m) &= \mathbf{w}_{\text{sib}} \cdot \mathbf{f}_{\text{sib}}(\mathbf{x}, \mathbf{t}, h, s, m) \\
\mathsf{Score}_{\text{grd}}(\mathbf{x}, \mathbf{t}, g, h, m) &= \mathbf{w}_{\text{grd}} \cdot \mathbf{f}_{\text{grd}}(\mathbf{x}, \mathbf{t}, g, h, m) \\
\mathsf{Score}_{\text{gsib}}(\mathbf{x}, \mathbf{t}, g, h, s, m) &= \mathbf{w}_{\text{gsib}} \cdot \mathbf{f}_{\text{gsib}}(\mathbf{x}, \mathbf{t}, g, h, s, m)
\end{aligned} \tag{8}
$$

where $\mathbf{f}_{\text{dep/sib/grd/gsib}}(.)$ denote the four kinds of feature vectors corresponding to the four scoring parts in Fig. 2 and $\mathbf{w}_{\text{dep/sib/grd/gsib}}$ are the feature weights. We follow the state-of-the-art practice to define the feature functions [14], [10]. Table II lists the syntactic feature templates, where $\#\mathsf{verb}(h, m)$ denotes the number of verbs between h and m, $\#\mathsf{conj}(h, m)$ the number of conjunctions, and $\#\mathsf{punc}(h, m)$ the number of punctuation marks. In addition, we use the last Chinese character of each word as its lemma, and duplicate each word-related feature in Table II by replacing words with lemmas following [15]. Compared with [7], this work explores much richer syntactic features, leading to improved parsing accuracy for both pipeline and joint models.

To better study the effect of joint modeling, we implement three pipeline parsing models of different complexities.

The First-order Parsing Model (O1): [16] proposes a general DP based parsing algorithm with time complexity of $O(n^3)$ for bilexical grammars. This algorithm is usually referred to as *Eisner algorithm*. Based on the algorithm, [11] propose the *first-order* model, in which each scoring part contains only one dependency. In other words, the score of a dependency tree in the first-order model only includes $\mathsf{Score}_{\text{dep}}(.)$ (see Eq. (7)).

Fig. 3 and Algorithm 1 illustrate Eisner algorithm in detail. Eisner algorithm uses *spans* as the basic data structures and builds a dependency tree in a bottom-up fashion by iteratively combining two smaller spans into a larger one. The left-side

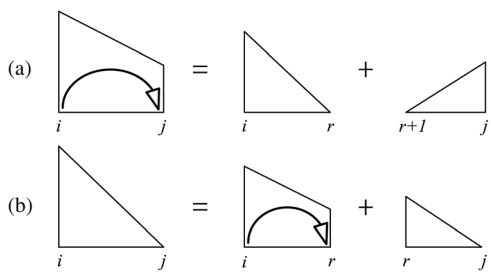

Fig. 3. The DP structures and derivations of Eisner algorithm for the first-order parsing model. Trapezoids denote *incomplete spans*. Triangles denote *complete spans*.

span in Fig. 3(a), denoted by $I_{i \curvearrowright j}$, represents an partial tree spanning $w_i \ldots w_j$ in which w_i and w_j composes a dependency $i \curvearrowright j$. $I_{i \curvearrowright j}$ is called an *incomplete span* in the sense that w_j may append more right-side children in future operations. The left-side span in Fig. 3(b), denoted by $C_{i \rightharpoonup j}$, represents a partial tree spanning $w_i \ldots w_j$ in which w_j is a descendant of w_i. $C_{i \rightharpoonup j}$ is called a *complete span* in the sense that w_j has collected all its children and therefore cannot add new modifiers in future. In both $I_{i \curvearrowright j}$ and $C_{i \rightharpoonup j}$, w_i is called the span head. Analogously, spans headed by w_j are denoted by $I_{i \curvearrowleft j}$ and $C_{i \leftharpoonup j}$. For brevity, Fig 3 omits the symmetric operations for creating these two spans of the opposite direction.

Algorithm 1: Eisner algorithm for the first-order dependency parsing (O1)

1. $\forall 0 \le i \le n, t_i \in \mathcal{T} C_{i \rightharpoonup i} = 0, C_{i \leftharpoonup i} = 0$ // *initialization*
2. **for** $w = 1$ **to** n **do** // *span width*
3. **for** $i = 0$ **to** $(n - w)$ **do** // *span start index*
4. $j = i + w$ // *span end index*
5. $I_{i \curvearrowright j} = \max_{i \le r < j} \{C_{i \rightharpoonup r} + C_{r+1 \leftharpoonup j} + \mathsf{Score}_{\text{dep}}(\mathbf{x}, \mathbf{t}, i, j)\}$ // *corresponding to Fig. 3(a)*
6. $I_{i \curvearrowleft j} = \max_{i \le r < j} \{C_{i \rightharpoonup r} + C_{r+1 \leftharpoonup j} + \mathsf{Score}_{\text{dep}}(\mathbf{x}, \mathbf{t}, j, i)\}$

Algorithm 2: The decoding algorithm of the first-order joint model (JO1)

1. $\forall 0 \le i \le n, t_i \in \mathcal{T} C^{t_i}_{i \rightarrow i} = 0, C^{t_i}_{i \leftarrow i} = 0$ *initialization*
2. **for** $w = 1$ **to** n **do** *//span width*
3. **for** $i = 0$ **to** $(n - w)$ **do** *//span start index*
4. $j = i + w$ *span end index*
5. **for all** $(t_i, t_j) \in \mathcal{T}^2$ **do**
6. $I^{t_{i,j}}_{i \curvearrowright j} = \max_{i \le r < j} \max_{(t_r, t_{r+1}) \in \mathcal{T}^2} \{C^{t_{i,r}}_{i \rightarrow r} + C^{t_{r+1,j}}_{r+1 \leftarrow j} + \mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, [t_i t_j], i, j) + \mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, j, t_j) + \mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, r{+}1, t_r, t_{r+1})\}$ *// corresponding to Fig. 4(a).*
7. $I^{t_{i,j}}_{i \curvearrowleft j} = \max_{i \le r < j} \max_{(t_r, t_{r+1}) \in \mathcal{T}^2} \{C^{t_{i,r}}_{i \rightarrow r} + C^{t_{r+1,j}}_{r+1 \leftarrow j} + \mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, [t_i t_j], j, i) + \mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, i, t_i) + \mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, r{+}1, t_r, t_{r+1})\}$
8. $C^{t_{i,j}}_{i \rightarrow j} = \max_{i < r \le j} \max_{t_r \in \mathcal{T}} \{I^{t_{i,r}}_{i \curvearrowright r} + C^{t_{r,j}}_{r \rightarrow j}\}$ *// corresponding to Fig. 4(b).*
9. $C^{t_{i,j}}_{i \leftarrow j} = \max_{i \le r < j} \max_{t_r \in \mathcal{T}} \{C^{t_{i,r}}_{i \leftarrow r} + I^{t_{r,j}}_{r \curvearrowleft j}\}$
10. **end for**
11. **end for**
12. **end for**
13. **return** $\max_{t_0 = \text{"\#"}, t_n \in \mathcal{T}} \{C^{t_{0,n}}_{0 \rightarrow n}\}$

7. $C_{i \rightarrow j} = \max_{i < r \le j} \{I_{i \curvearrowright r} + C_{r \rightarrow j}\}$ *// corresponding to Fig. 3(b).*
8. $C_{i \leftarrow j} = \max_{i \le r < j} \{C_{i \leftarrow r} + I_{r \curvearrowleft j}\}$
9. **end for**
10. **end for**
11. **return** $C_{0 \rightarrow n}$

Line 5 in Algorithm 1 tries to find the optimal incomplete span $I_{i \curvearrowright j}$. Given r, two spans $C_{i \rightarrow r}$ and $C_{r+1 \leftarrow j}$ are combined, as illustrated in Fig. 3(a). This operation invents a new dependency $i \curvearrowright j$. The score of the resulting span $I_{i \curvearrowright j}$ is composed of three parts, i.e. the scores of the two component spans and the score contributed by the new dependency $\mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, \mathbf{t}, i, j)$ (see Eq. (7)). All possible split points r are tried to find the highest-scoring $I_{i \curvearrowright j}$.

Line 7 in Algorithm 1 tries to find the optimal complete span $C_{i \rightarrow j}$. Given r, two spans $I_{i \curvearrowright r}$ and $C_{r \rightarrow j}$ are combined, as illustrated in Fig. 3(b). The score of the resulting span $C_{i \rightarrow j}$ is the summation of the scores of the two component spans. All possible split points r are tried to find the highest-scoring $C_{i \rightarrow j}$.

Analogously, line 6 and line 8 build the spans headed by w_j. We omit the explanation for conciseness. Finally, the returned span $C_{0,n}$ contains the score of the optimal complete dependency tree, and the dependency tree can be gained via backtracking. It can be proved that Algorithm 1 finds the highest scoring tree according to Eq. (7).

The algorithm stores about $2n^2$ complete spans and $2n^2$ incomplete spans. Therefore, the space complexity is $O(n^2)$. The time complexity of line 5-8 in Algorithm 1 is $O(4n)$; the outside loops over w and i has a total complexity of $O(n^2)$; therefore, the time complexity is $O(n^3)$.

The Second- and Third-order Parsing Models (O2 & O3): [10] extend the Eisner algorithm and propose a DP based decoding algorithm which can incorporate all the scoring parts in Fig. 2. The model is referred to as the *third-order* model.

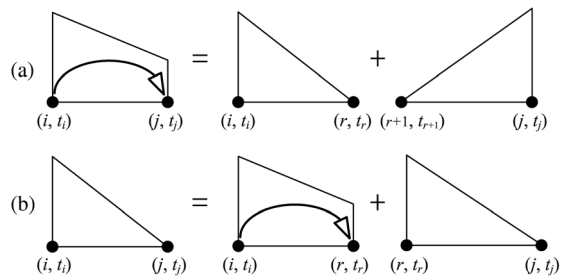

Fig. 4. The DP structures and derivations of the decoding algorithm of the basic first-order joint models (JO1). We omit symmetric right-headed versions for brevity. Solid circles denote POS tags of the corresponding indices.

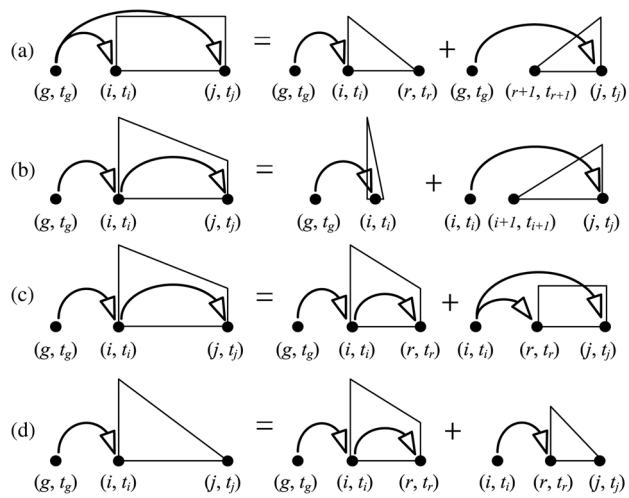

Fig. 5. The DP structures and derivations of the decoding algorithm of the third-order joint model (JO3). For brevity, we elide the right-headed and right-parented versions. Rectangles represent sibling spans.

If the grandparent-sibling features are deactivated, it becomes the *second-order* model. We omit the detailed illustration of the decoding algorithm for brevity, since it can be derived from the more complex decoding algorithm for the third-order joint model (see Fig. 5 and Algorithm 3).

III. JOINT MODELS

In the joint framework, we aim to simultaneously solve the two problems.

$$(\hat{\mathbf{t}}, \hat{\mathbf{d}}) = \arg\max_{\mathbf{t},\mathbf{d}} \mathsf{Score}_{\mathrm{joint}}(\mathbf{x}, \mathbf{t}, \mathbf{d}) \tag{9}$$

where $\mathsf{Score}_{\mathrm{joint}}(\mathbf{x}, \mathbf{t}, \mathbf{d})$ denotes the score of the tag sequence $\mathbf{t}$ and dependency tree $\mathbf{d}$ for the input sentence $\mathbf{x}$, which is the summation of the POS score and syntactic score defined in the pipeline tagging and parsing models.

$$\begin{aligned}\mathsf{Score}_{\mathrm{joint}}(\mathbf{x}, \mathbf{t}, \mathbf{d}) &= \mathsf{Score}_{\mathrm{pos}}(\mathbf{x}, \mathbf{t}) + \mathsf{Score}_{\mathrm{syn}}(\mathbf{x}, \mathbf{t}, \mathbf{d}) \\ &= \mathbf{w}_{\mathrm{pos}\oplus\mathrm{syn}} \cdot \mathbf{f}_{\mathrm{pos}\oplus\mathrm{syn}}(\mathbf{x}, \mathbf{t}, \mathbf{d}) \\ &= \mathbf{w}_{\mathrm{joint}} \cdot \mathbf{f}_{\mathrm{joint}}(\mathbf{x}, \mathbf{t}, \mathbf{d}) \end{aligned} \tag{10}$$

where $\mathbf{f}_{\mathrm{joint}}(.)$ or $\mathbf{f}_{\mathrm{pos}\oplus\mathrm{syn}}(.)$ denotes the aggregated feature vector for the joint result, which is the concatenation of $\mathbf{f}_{\mathrm{pos}}(.)$ and $\mathbf{f}_{\mathrm{syn}}(.)$; $\mathbf{w}_{\mathrm{joint}}$ or $\mathbf{w}_{\mathrm{pos}\oplus\mathrm{syn}}$ is the feature weights. Different from the pipeline method, the joint model simultaneously learns the weights of the POS and syntactic features so that their discriminative power can be well balanced. During the test phase, the POS and syntactic features can interact and negotiate with each other to determine an optimal joint result.

The crucial problem for the joint approach is to design effective decoding algorithms to effectively capture rich POS tagging and syntactic features and efficiently search out the optimal results from a huge hypothesis space at the same time. [16] describes a preliminary idea to handle *polysemy* by extending parsing algorithms. Based on this idea, we propose two DP based decoding algorithms for our joint models by extending the decoding algorithms of the parsing models.

To compare with the baseline parsing models, we implement three joint models of different complexities, i.e. the *first-order*, *second-order*, and *third-order* joint models according to the syntactic features incorporated in $\mathsf{Score}_{\mathrm{syn}}(.)$ (see Eq. (7)).

A. The First-order Joint Model (JO1)

Similar to the first-order parsing model, the syntactic score $\mathsf{Score}_{\mathrm{syn}}(.)$ in the first-order joint model only consists of scores contributed by single dependencies $\mathsf{Score}_{\mathrm{dep}}(.)$ (see Eq. (7)). We propose a DP based decoding algorithm for the JO1 model by extending the Eisner algorithm based on the idea in [16]. Fig. 4 and Algorithm 2 illustrate the algorithm in detail. The difference is that the boundary indices of each span are augmented with their POS tags. $I^{t_i,t_j}_{i\curvearrowright j}$ denotes an incomplete span with w_i tagged as t_i and w_j as t_j like the left-side one in Fig. 4(a); whereas $C^{t_i,t_j}_{i\rightarrow j}$ denotes a complete span with boundary POS tags t_i and t_j like the left-side one in Fig. 4(b).

Algorithm 2 also works in a bottom-up fashion. Line 6 tries to find the optimal incomplete span $I^{t_i,t_j}_{i\curvearrowright j}$. Given r, t_r, and t_{r+1}, two spans $C^{t_i,r}_{i\rightarrow r}$ and $C^{t_{r+1},t_j}_{r+1\leftarrow j}$ are combined, as illustrated in Fig. 4(a). This operation invents a new dependency $i \curvearrowright j$. The score of the resulting span $I^{t_i,t_j}_{i\curvearrowright j}$ is composed of five parts, i.e. the scores of the two component spans, the score contributed by the new dependency ($\mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, [t_i t_j], i, j)$), a POS score for tagging w_j as t_j ($\mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, j, t_j)$), and a POS score for tagging w_r as t_r and w_{r+1} as t_{r+1} ($\mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, r+1, t_r, t_{r+1})$). All possible r, t_r, and t_{r+1}, are enumerated to find the highest-scoring $I^{t_i,t_j}_{i\curvearrowright j}$.

Note that $\mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, [t_i t_j], i, j)$ is a collapsed version of $\mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, \mathbf{t}, i, j)$ defined in Eq. (7) with incomplete parameters. $\mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, \mathbf{t}, i, j)$ requires the full tag sequence $\mathbf{t}$ as the input, but $\mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, [t_i t_j], i, j)$ only provides the tags of w_i and w_j as the decoding algorithm is unable to encode other context POS tags. As shown in Table II, the syntactic feature set for a single dependency $\mathbf{f}_{\mathrm{dep}}(\mathbf{x}, \mathbf{t}, i, j)$ requires other context POS tags like $t_{i\pm1}$, $t_{j\pm1}$, and even the number of verbs between i and j ($\#\mathsf{verb}(i, j)$). Such surrounding or even global POS tags are not encoded in the spans. Therefore, we follow the practice in [17] and approximately fix such context POS tags as the most likely ones from the baseline CRF-based tagger. [7] show that this approximation does not decrease the tagging and parsing accuracies.[1]

Line 8 in Algorithm 2 tries to find the optimal complete span $C^{t_i,t_j}_{i\rightarrow j}$. Given r and t_r, two spans $I^{t_i,t_r}_{i\curvearrowright r}$ and $C^{t_r,t_j}_{r\rightarrow j}$ are combined, as illustrated in Fig. 4(b). The score of the resulting span $C^{t_i,t_j}_{i\rightarrow j}$ is the summation of the scores of the two component spans. All possible r and t_r are enumerated to find the highest-scoring $C^{t_i,t_j}_{i\rightarrow j}$.

Line 7 and 9 create spans headed by w_j. Finally, line 13 enumerates all possible tag t_n for w_n to find the highest scoring complete span $C^{t_0,n}_{0\rightarrow n}$. We use "#" as the POS tag of the dummy word w_0. Through backtracking, we can get the highest-scoring joint result. Analogous to Algorithm 1, we can prove that Algorithm 2 correctly compute the syntactic score according to Eq. (7).

Then the question is whether the POS scores are correctly computed according to Eq. (4). In Algorithm 2, we collect the POS scores following two principles. (1) The POS score for tagging w_k as t_k is collected when the father of w_k is determined. For example, the operation in Fig. 4(a) assign w_i as the father of w_j, leading to the accumulation of $\mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, j, t_j)$ into the resulting span $I^{t_i,t_j}_{i\curvearrowright j}$. Since Algorithm 2 assigns one and only one head for each word, we can see that POS scores contributed by the POS unigram features are correctly computed. (2) the POS score contributed by tagging w_k as t_k and w_{k+1} as t_{k+1} is collected when two complete spans split at k are combined into one incomplete span. For example, the operation in Fig. 4(b) triggers the collection of $\mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, r+1, t_r, t_{r+1})$. It can be proved that the POS scores contributed by the POS bigram features are correctly computed in this way, though the proof is slightly more trivial and complex.

The algorithm needs to store $2n^2q^2$ incomplete spans and $2n^2q^2$ complete spans, where $q = |\mathcal{T}|$. Therefore, the space complexity is $O(n^2q^2)$. Line 6 and 7 both have time complexity of $O(nq^2)$; whereas line 8 and 9 both require $O(nq)$. The outside loops over w, i, t_i, and t_j require $O(n^2q^2)$. Therefore, the time complexity of Algorithm 2 is $O(n^3q^4)$.

[1] In [7], we try to encode some context POS tags like $t_{i\pm1}$ and $t_{j\pm1}$ in the spans and comes up with more complicated decoding algorithms, which we name as *the joint models of version 2*. Experiments show that the joint models of version 2 achieve similar performance in both tagging and parsing, but is much slower.

Algorithm 3: The decoding algorithm of the third-order joint model (JO3)

1. $\forall 0 \le i \le n, -1 \le g \le n, (t_g, t_i) \in \mathcal{T}^2 C^{t_{g,i}}_{g \curvearrowright i, i \rightarrow i} = 0, C^{t_{g,i}}_{g \curvearrowright i, i \leftarrow i} = 0$ // *initialization*
2. **for** $w = 1$ **to** n **do** // *span width*
3. **for** $i = 0$ **to** $(n - w)$ **do** // *span start index*
4. $j = i + w$ // *span end index*
5. **for all** g such that $-1 \le g < i$ **or** $j < g \le n$ **do** // *parent index*
6. **for all** $(t_g, t_i, t_j) \in \mathcal{T}^3$ **do**
7. $S^{t_{g,i,j}}_{g \curvearrowright i, g \curvearrowright j} = \max_{i \le r < j} \max_{(t_r, t_{r+1}) \in \mathcal{T}^2} \{C^{t_{g,i,r}}_{g \curvearrowright i, i \rightarrow r} + C^{t_{g,r+1,j}}_{g \curvearrowright j, r+1 \leftarrow j} + \mathsf{Score}_{\mathrm{sib}}(\mathbf{x}, [t_g t_i t_j], g, i, j) + \mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, r+1, t_r, t_{r+1})\}$ // *corresponding to Fig. 5(a).*
8. $I^{t_{g,i,j}}_{g \curvearrowright i, i \curvearrowright j} = \max_{t_{i+1} \in \mathcal{T}} \{C^{t_{g,i}}_{g \curvearrowright i, i \rightarrow i} + C^{t_{i,i+1,j}}_{i \curvearrowright j, i+1 \leftarrow j} + \mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, [t_i t_j], i, j) + \mathsf{Score}_{\mathrm{grd}}(\mathbf{x}, [t_g t_i t_j], g, i, j) + \mathsf{Score}_{\mathrm{sib}}(\mathbf{x}, [t_i t_j], i, -, j) + \mathsf{Score}_{\mathrm{gsib}}(\mathbf{x}, [t_g t_i t_j], g, i, -, j) + \mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, i+1, t_i, t_{i+1}) + \mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, j, t_j)\}$ // *j is the first child of i; corresponding to Fig. 5(b).*
9. $I^{t_{g,i,j}}_{g \curvearrowright j, i \curvearrowleft j} = \max_{t_{j-1} \in \mathcal{T}} \{C^{t_{i,j-1,j}}_{i \curvearrowleft j, i \rightarrow j-1} + C^{t_{g,j}}_{g \curvearrowright j, j \leftarrow j} + \mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, [t_i t_j], j, i) + \mathsf{Score}_{\mathrm{grd}}(\mathbf{x}, [t_g t_i t_j], g, j, i) + \mathsf{Score}_{\mathrm{sib}}(\mathbf{x}, [t_i t_j], j, -, i) + \mathsf{Score}_{\mathrm{gsib}}(\mathbf{x}, [t_g t_i t_j], g, j, -, i) + \mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, j, t_{j-1}, t_j) + \mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, i, t_i)\}$ // *i is the first child of j.*
10. $I^{t_{g,i,j}}_{g \curvearrowright i, i \curvearrowright j} = \max\{I^{t_{g,i,j}}_{g \curvearrowright i, i \curvearrowright j}, \max_{i<r<j} \max_{t_r \in \mathcal{T}} \{I^{t_{g,i,r}}_{g \curvearrowright i, i \curvearrowright r} + S^{t_{i,r,j}}_{i \curvearrowright r, i \curvearrowright j} + \mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, [t_i t_j], i, j) + \mathsf{Score}_{\mathrm{grd}}(\mathbf{x}, [t_g t_i t_j], g, i, j) + \mathsf{Score}_{\mathrm{gsib}}(\mathbf{x}, [t_g t_i t_r t_j], g, i, r, j) + \mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, j, t_j)\}\}$ // *corresponding to Fig. 5(c).*
11. $I^{t_{g,i,j}}_{g \curvearrowright j, i \curvearrowleft j} = \max\{I^{t_{g,i,j}}_{g \curvearrowright j, i \curvearrowleft j}, \max_{i<r<j} \max_{t_r \in \mathcal{T}} \{S^{t_{i,r,j}}_{i \curvearrowleft j, r \curvearrowleft j} + I^{t_{g,r,j}}_{g \curvearrowright j, r \curvearrowleft j} + \mathsf{Score}_{\mathrm{dep}}(\mathbf{x}, [t_i t_j], j, i) + \mathsf{Score}_{\mathrm{grd}}(\mathbf{x}, [t_g t_i t_j], g, j, i) + \mathsf{Score}_{\mathrm{gsib}}(\mathbf{x}, [t_g t_i t_r t_j], g, j, r, i) + \mathsf{Score}_{\mathrm{pu}}(\mathbf{x}, i, t_i)\}\}$
12. $C^{t_{g,i,j}}_{g \curvearrowright i, i \rightarrow j} = \max_{i<r \le j} \max_{t_r \in \mathcal{T}} \{I^{t_{g,i,r}}_{g \curvearrowright i, i \curvearrowright r} + C^{t_{i,r,j}}_{i \curvearrowright r, r \rightarrow j} + \mathsf{Score}_{\mathrm{sib}}(\mathbf{x}, [t_i t_r], i, r, -) + \mathsf{Score}_{\mathrm{gsib}}(\mathbf{x}, [t_g t_i t_r], g, i, r, -)\}$ // *r is the last child of i; corresponding to Fig. 5(d).*
13. $C^{t_{g,i,j}}_{g \curvearrowright j, i \leftarrow j} = \max_{i \le r<j} \max_{t_r \in \mathcal{T}} \{C^{t_{i,r,j}}_{r \curvearrowleft j, i \leftarrow r} + I^{t_{g,r,j}}_{g \curvearrowright j, r \curvearrowleft j} + \mathsf{Score}_{\mathrm{sib}}(\mathbf{x}, [t_r t_j], j, r, -) + \mathsf{Score}_{\mathrm{gsib}}(\mathbf{x}, [t_g t_r t_j], g, j, r, -)\}$ // *r is the last child of j.*
14. **end for**
15. **end for**
16. **end for**
17. **end for**
18. **return** $\max_{t_{-1}=\text{"\#"}, t_0=\text{"\#"}, t_n \in \mathcal{T}} \{C^{t_{-1,0,n}}_{-1 \curvearrowright 0, 0 \rightarrow n}\}$

B. The Second- and Third-order Joint Models (JO2 & JO3)

[10] propose a DP based decoding algorithm for the third-order parsing model. We extend their algorithm and propose the decoding algorithm for the third-order joint model, as illustrated in Fig. 5 and Algorithm 3. In order to incorporate the higher-order features, a new type of spans, namely *sibling spans*, is invented to encode sibling structures; moreover, each span is augmented with a parent index. Analogous to the extensions made for the first-order joint decoding algorithm, we augment each span with the POS tags at the boundary indices as well as the parent index. The left-side span in Fig. 5(a), denoted as $S^{t_{g,i,j}}_{g \curvearrowright i, g \curvearrowright j}$, represents a sibling span in which i and j are two adjacent modifiers of g and their POS tags are respectively t_g, t_i, and t_j. $I^{t_{g,i,j}}_{g \curvearrowright i, i \curvearrowright j}$ and $C^{t_{g,i,j}}_{g \curvearrowright i, i \rightarrow j}$ denotes the same spans as $I^{t_{i,j}}_{i \curvearrowright j}$ and $C^{t_{i,j}}_{i \rightarrow j}$ except the extra requirements that g is the head of i and is tagged as t_g.

The work flow of Algorithm 3 is analogous to the case for the first-order joint model. Line 7 tries to find the highest-scoring sibling span $S^{t_{g,i,j}}_{g \curvearrowright i, g \curvearrowright j}$. For all possible r, t_r, t_{r+1}, two spans $C^{t_{g,i,r}}_{g \curvearrowright i, i \rightarrow r}$ and $C^{t_{g,r+1,j}}_{g \curvearrowright j, r+1 \leftarrow j}$ are combined, and the scores triggered by this operation are collected. $\mathsf{Score}_{\mathrm{sib}}(\mathbf{x}, [t_g t_i t_j], g, i, j)$ is the collapsed version of $\mathsf{Score}_{\mathrm{sib}}(\mathbf{x}, \mathbf{t}, g, i, j)$ defined in Eq. (7) with incomplete POS tags. $\mathsf{Score}_{\mathrm{pb}}(\mathbf{x}, r+1, t_r, t_{r+1})$ is defined in Eq. (5). The highest-scoring one will be kept in $S^{t_{g,i,j}}_{g \curvearrowright i, g \curvearrowright j}$.

Line 8 and 10 collaboratively determine the best incomplete span $I^{t_{g,i,j}}_{g \curvearrowright i, i \curvearrowright j}$. Line 8 handles the case when j is the first child of i, whereas line 10 handles other cases. $\mathsf{Score}_{\mathrm{sib}}(\mathbf{x}, [t_i t_j], i, -, j)$ indicates the sibling score of creating a dependency $i \curvearrowright j$ where j is the first right-side child of i. Similarly, $\mathsf{Score}_{\mathrm{gsib}}(\mathbf{x}, [t_g t_i t_j], g, i, -, j)$ represents the grand-parent-sibling score of j being the first right-side child of i and i modifying g.

Line 12 finds the optimal complete span $C^{t_{g,i,j}}_{g \curvearrowright i, i \rightarrow j}$. The algorithm implies that once $C^{t_{g,i,j}}_{g \curvearrowright i, i \rightarrow j}$ is built, i will not append new right-side children in the future. Therefore, this operation triggers last-child syntactic scores. $\mathsf{Score}_{\mathrm{sib}}(\mathbf{x}, [t_i t_r], i, r, -)$ represents the sibling score of r being the last right-side child of i. Similar explanation goes to $\mathsf{Score}_{\mathrm{gsib}}(\mathbf{x}, [t_g t_i t_r], g, i, r, -)$.

For conciseness, we omit the illustration of line 9, 11, and 13, which create spans headed by w_j with symmetric operations. Analogous to the first-order case, the context POS tags like $t_{g \pm 1}$ and $t_{s \pm 1}$, which are missed by the collapsed score functions $\mathsf{Score}_{\mathrm{dep/sib/grd/gsib}}(.)$ in the algorithm (see Eq. (7) and Table II), are fixed as the most likely POS tags from the baseline CRF-based tagger.

Similarly, we can prove that this algorithm can correctly compute the POS score according to Eq. (4) and the syntactic score according to Eq. (7). We can also find out that the space and time complexity of the algorithm are respectively $O(n^3q^3)$ and $O(n^4q^5)$.

The second-order joint model (JO2) also adopts Algorithm 3 for decoding, except that the scores contributed by the third-order grandparent-sibling features $\mathsf{Score}_{\mathrm{gsib}}(.)$ are deactivated.

C. POS Tag Pruning

The time complexity of the joint decoding algorithm is high with regard to the number of candidate POS tags for each word ($q = |\mathcal{T}|$). [17] constrain the search space and use two most likely POS tags for each word according to a baseline ME based POS tagger ($q = 2$). However, we find that it is still too time-consuming, especially for JO2 and JO3. To deal with this problem, we propose a pruning method that can effectively reduce the POS tag space based on marginal probabilities provided by the baseline CRF-based tagger.

The marginal probability of tagging a word w_i as t is

$$P(t_i = t|\mathbf{x}) = \sum_{\mathbf{t}:\mathbf{t}[i]=t} P(\mathbf{t}|\mathbf{x}) = \frac{\sum_{\mathbf{t}:\mathbf{t}[i]=t} \exp(\mathsf{Score}_{\mathrm{pos}}(\mathbf{x},\mathbf{t}))}{\sum_{\mathbf{t}'} \exp(\mathsf{Score}_{\mathrm{pos}}(\mathbf{x},\mathbf{t}'))} \quad (11)$$

where $P(\mathbf{t}|\mathbf{x})$ and $\mathsf{Score}_{\mathrm{pos}}(\mathbf{x},\mathbf{t})$ is defined in Eq. (3) and (4). This can be efficiently computed using the *forward-backward* algorithm.

We define $\mathtt{ptmax}_i(\mathbf{x})$ to be the highest marginal probability of tagging the word w_i:

$$\mathtt{ptmax}_i(\mathbf{x}) = \max_{t\in\mathcal{T}} P(t_i = t|\mathbf{x}) \quad (12)$$

We then define the allowable candidate POS tags of the word w_i to be

$$\mathcal{T}_i(\mathbf{x}) = \{t : t \in \mathcal{T}, P(t_i = t|\mathbf{x}) \geq \lambda_t \times \mathtt{ptmax}_i(\mathbf{x})\} \quad (13)$$

where λ_t is the pruning threshold ($0 \leq \lambda_t \leq 1$). $\mathcal{T}_i(\mathbf{x})$ is used to constrain the POS search space by replacing $\mathcal{T}$ in Algorithm 2 and 3. For example, the enumeration $(t_g, t_i, t_j) \in \mathcal{T}^3$ in line 6 of Algorithm 3 becomes $t_g \in \mathcal{T}_g(\mathbf{x})$, $t_i \in \mathcal{T}_i(\mathbf{x})$, and $t_j \in \mathcal{T}_j(\mathbf{x})$. Experiments in Section IV show that this pruning strategy is effective. When $\lambda_t = 0.01$, each word has only 1.40 candidate POS tags on average and the oracle tagging accuracy is 99.27% on CTB5. We adopt 10-fold jackknifing to do pruning for the training dataset. First, training sentences are randomly divided into ten folds. Then, we prune the sentences in one fold using a CRF-based model trained with the sentences in the left nine folds. For the development and test datasets, we train a CRF-based model with the entire training dataset to do pruning.

D. Training with Averaged Perceptron

Averaged perceptron training has proven successful in several structured classification problems such as POS tagging [3] and parsing [11], [18]. Our preliminary experiments on dependency parsing show that averaged perceptron achieves similar performance to other more sophisticated training algorithms such as the passive-aggressive algorithm [19] and the margin infused relaxed algorithm (MIRA) [20]. Therefore, we adopt average perceptron to train both the pipeline parsing and joint models. Algorithm 4 shows the workflow for training the joint models. The procedure iteratively traverses the entire training dataset and use one instance to update the feature weights at each time. First, the best hypothesis for the instance is found based on the current feature weights at line 6. Then, the feature weights are updated such that the features in the gold-standard result get higher weights, whereas the features in the returned hypothesis are penalized. If the current model finds a completely correct result, then the two feature sets are equal and no update is made.

The training algorithm for the pipeline parsing models is similar to Algorithm 4 except that 1) the decoding algorithm for the parsing model is applied to find the best tree in line 6, and 2) only the syntactic features $\mathbf{f}_{\mathrm{syn}}(.)$ and their weights $\mathbf{w}_{\mathrm{syn}}$ are involved. For both the pipeline parsing and joint tasks, we train each model for 20 iterations and select the parameters that perform best on the development set.

Algorithm 4: Averaged perceptron training for joint POS tagging and dependency parsing

1. **Input:** Training Data $\mathbb{D} = \{(\mathbf{x}^{(j)}, \mathbf{t}^{(j)}, \mathbf{d}^{(j)})\}_{j=1}^{N}$
2. **Output:** $\mathbf{w}_{\mathrm{joint}} (\equiv \mathbf{w}_{\mathrm{pos}\oplus\mathrm{syn}})$
3. **Initialization:** $\mathbf{w}_{\mathrm{joint}}^{(0)} = \mathbf{0}; \mathbf{v} = \mathbf{0}; k = 0$
4. **for** $i = 1$ **to** I **do** // *iterations*
5. **for** $j = 1$ **to** N **do** // *traverse the samples*
6. $(\hat{\mathbf{t}}, \hat{\mathbf{d}}) = \arg\max_{\mathbf{t},\mathbf{d}} \mathbf{w}_{\mathrm{joint}}^{(\mathbf{k})} \cdot \mathbf{f}_{\mathrm{joint}}(\mathbf{x}^{(\mathbf{j})}, \mathbf{t}, \mathbf{d})$ // *decode based on current feature weights*
7. $\mathbf{w}_{\mathrm{joint}}^{(k+1)} = \mathbf{w}_{\mathrm{joint}}^{(k)} + \mathbf{f}_{\mathrm{joint}}(\mathbf{x}^{(j)}, \mathbf{t}^{(j)}, \mathbf{d}^{(j)}) - \mathbf{f}_{\mathrm{joint}}(\mathbf{x}^{(j)}, \hat{\mathbf{t}}, \hat{\mathbf{d}})$
8. $\mathbf{v} = \mathbf{v} + \mathbf{w}_{\mathrm{joint}}^{(k+1)}$
9. $k = k + 1$
10. **end for**
11. **end for**
12. $\mathbf{w}_{\mathrm{joint}} = \mathbf{v}/(I \times N)$ // *average the weights*

IV. Experiments

We conduct the experiments on the widely used CTB5 [21] to compare with the state-of-the-art results. Following the settings of [4], [5], [6], we split CTB5 into training (secs 001-815 and 1001-1136), development (secs 886-931 and 1148-1151), and test (secs 816-885 and 1137-1147) sets, and use the head-finding rules of [5] to turn the bracketed sentences into dependency structures.

We also compare the pipeline and joint models on CTB7, a newer and larger-scale version, to further examine the impact of joint modeling. We follow the data split suggested in the official manual and convert the phrase-structure trees into dependency structures with the head-finding rules of [5]. Table III summarizes the data sets used in this paper.

Evaluation metrics. We use the standard tagging accuracy (TA) to evaluate POS tagging. For dependency parsing, we use

TABLE III
DATA SETS (IN SENTENCE NUMBER)

	Train	Dev	Test
CTB5	16,091	803	1,910
CTB7	46,572	2,079	2,796

TABLE IV
RESULTS OF THE JO2 MODEL WITH DIFFERENT λ_t ON THE DEVELOPMENT SET

λ_t	$\lvert\mathcal{T}_i(\mathbf{x})\rvert$	Oracle	UAS	RA	CM	TA	Speed
0.001	1.83	99.55	82.85	76.75	31.50	94.52	0.5
0.01	1.40	99.27	**83.02**	**78.63**	**31.75**	94.52	1.0
0.1	1.16	98.07	82.77	77.38	31.25	**94.55**	1.7

the unlabeled attachment score (UAS) as the main evaluation metric. We also provide the root accuracy (RA) and complete match rate (CM) for detailed comparison with previous results. Following previous practice, all metrics for parsing ignore punctuation marks. For significance test, we adopt Dan Bikel's randomized parsing evaluation comparator [22].[2]

A. Impact of POS Tag Pruning

To study the effect of POS tag pruning, we train and evaluate the JO2 model on the training and development sets with different pruning threshold λ_t. We adopt the JO2 model because of its efficiency compared with the JO3 model and its capability of capturing rich features compared with the JO1 model. Table IV shows the results. $|\mathcal{T}_i(\mathbf{x})|$ means the averaged number of POS tag candidates of each word after pruned with the corresponding threshold. "Oracle" refers to the oracle TA in the pruned POS space. "Speed" refers to the parsing speed measured in the number of sentences processed per second. We can see that pruning with $\lambda_t = 0.01$ leads to the highest parsing accuracy. Moreover, it seems that λ_t does not largely influence the tagging accuracy. Therefore, we choose $\lambda_t = 0.01$ with which the constrained search space contains 1.40 POS tag candidates for each word on average and achieve a high oracle tagging accuracy of 99.27%. For simplicity, we do not tune λ_t for other joint models, and adopt $\lambda_t = 0.01$ for all joint models in later experiments.

B. Final Results on CTB5

Experimental results of the pipeline models with gold-standard POS tags: Table V shows the results of our baseline parsing models on the test set with gold-standard POS tags. We also list a few state-of-the-art results in the bottom. Duan07 refers to the results of [4]. They enhance the transition-based parsing model with the beam search. Z&C08 refers to the results of [5]. They use a hybrid model to combine the advantages of both graph-based and transition-based models. H&S10 refers to the results of [6]. They expand the search space of the transition-based model by merging equivalent states with DP. Z&N11 refers to the feature-rich transition-based parser of [23], which achieve the best performance so far for transition-based models on CTB5. Hatori11 refers to the pipeline transition-based parsing model in [24].

TABLE V
RESULTS OF THE PIPELINE PARSING MODELS WITH GOLD-STANDARD POS TAGS ON THE TEST SET

	UAS	RA	CM
O3	86.60	80.26	36.07
O2	**86.72**	80.26	36.07
O1	83.77	73.40	28.38
Hatori11 [24]	85.96	**80.87**	35.03
Z&N11 [23]	86.0	—	**36.9**
H&S10 [6]	85.20	78.32	33.72
Z&C08 [5]	85.77	76.26	34.41
Duan07 [4]	83.88	73.70	32.70

TABLE VI
RESULTS OF THE PIPELINE MODELS WITH AUTOMATIC POS TAGS ON THE TEST SET

	UAS	RA	CM	TA	Speed
O3	80.50	**77.33**	28.22		0.7
O2	**80.64**	**77.33**	**28.59**	**93.88**	2.7
O1	77.19	70.05	22.72		5.8
Hatori11 [24]	78.04	75.55	26.07	93.82	**9.0**

We can see that when using gold-standard POS tags, our pipeline O2 and O3 parsing models achieve the best parsing accuracy. It is a little unexpected that the O2 model slightly outperforms the O3 one. However, the difference is not statistically significant ($p > 0.1$). We suspect that the reason may be that the grandparent-sibling features are too sparse to be useful for this dataset. [10] also show that their third-order model outperforms the second-order one by only 0.32% on English and 0.07% on Czech.

Experimental results of the pipeline models with automatic POS tags: Table VI shows the results on the test set when using automatic POS tags. Compared with the results in Table V, the parsing accuracy of the pipeline models degrades by 6∼8%. This demonstrates that the tagging errors severely influence the parsing process. Since POS tags provides the most informative clues for parsing, the parsing accuracy suffers a lot when the POS tags are unreliable.

Experimental results of the joint models: Table VII shows the performance of the joint models on the test set. Compared with the results in Table VI, we can see that the joint models can help both tagging and parsing. The parsing accuracy is significantly improved by $0.7 \sim 0.9\%$ for all first-, second- and third-order cases ($p < 10^{-3}$). The tagging accuracy also significantly increases from 93.88% to 94.28%, and the improvement is statistically significant ($p < 10^{-3}$). Compared with the transition-based joint model (Hatori11) of [24], our JO2 and JO3 models achieve similar parsing accuracy and better tagging accuracy.

Comparing the parsing speed, we can find that the pruning of POS tags is very effective. The JO2 model can parse 1.0 sentences per second, whereas the pipeline O2 model can parse 2.7

[2]http://www.cis.upenn.edu/~dbikel/software.html

TABLE VII
RESULTS OF THE JOINT MODELS ON THE TEST SET

	UAS	RA	CM	TA	Speed
JO3	81.26	77.07	**30.47**	94.19	0.27
JO2	81.30	77.17	29.32	**94.28**	1.0
JO1	78.09	70.68	23.56	94.05	**5.8**
Hatori11 [24]	**81.33**	**77.93**	29.90	93.94	1.5

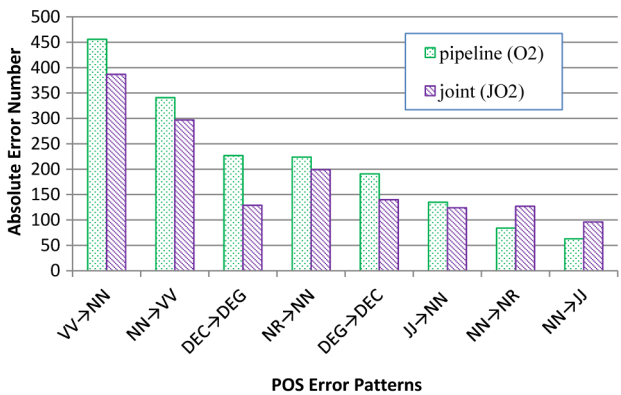

Fig. 6. POS error analysis.

sentences per second, which is not a large difference considering that there is a factor of q^5.

C. Error Analysis

To find out the impact of our joint models on the individual tasks, we conduct detailed error analysis through comparing the results of the pipeline O2 model and the JO2 model.

Impact on POS tagging: Fig. 6 shows how the joint model changes the distribution of a number of high-frequency POS error patterns compared with the pipeline model. An error pattern "X → Y" means that the focus word, whose true tag is "X", is assigned a tag "Y". We can see that the joint method is clearly better at resolving tagging ambiguities like {VV, NN} and {DEG, DEC}.[3] One common characteristic of these ambiguous pairs is that the local or even whole syntactic structure will be destructed if the wrong tag is chosen. In other words, correctly resolving these ambiguities is critical and helpful from the parsing viewpoint. From another perspective, the joint model is capable of preferring the right tag with the help of syntactic structures, which is impossible for the baseline sequential labeling model. In this sense, we call such POS ambiguities like {VV, NN} and {DEG, DEG} *syntax-sensitive ambiguities*.

TABLE VIII
COMPARISON OF PARSING ERROR RATES ON DIFFERENT POS TAG PATTERNS BETWEEN THE PIPELINE AND JOINT MODELS

POS patterns		pipeline		joint	
		percent	error rate	percent (±)	error rate (±)
Correct	NN → NN	95.7	16.0	-0.6	-1.3
	VV → VV	90.4	31.7	+0.3	-1.3
	DEG → DEG	85.0	9.8	+4.0	-2.8
	DEC → DEC	80.7	18.3	+8.3	-5.3
	NR → NR	90.7	14.4	+1.1	-0.6
	JJ → JJ	81.3	8.9	0	-0.6
Syntax-sensitive	NN → VV	2.5	54.3	-0.3	+20.8
	VV → NN	6.6	62.5	-1.0	+4.7
	DEG → DEC	15.0	50.3	-4.0	+49.0
	DEC → DEG	19.1	45.8	-8.2	+51.1
Syntax-insensitive	NN → NR	0.6	23.8	+0.3	+0.6
	NN → JJ	0.5	17.5	+0.2	+0.2
	NR → NN	7.1	24.6	-0.8	-8.0
	JJ → NN	10.4	17.0	-0.9	+2.4

In contrast, pairs like {NN, NR} and {NN, JJ} only slightly influence the syntactic structure when mis-tagged. Therefore, we refer to them as *syntax-insensitive ambiguities*. The joint method slightly reduces the errors of "NR → NN" and "JJ → NN", but also increases the errors of "NN → NR" and "NN → JJ".

In summary, we can conclude that *the joint model can help resolve syntax-sensitive POS ambiguities which are more discriminative and helpful from the parsing viewpoint.*

Impact on parsing: Table VIII studies how the joint model influences the percentage and the parsing error rates of different POS tag patterns. Given a pattern "X → Y", "percent" means the percentage of words that are tagged as "X" in the gold-standard reference and assigned "Y" by the pipeline or joint model; while parsing "error rate" is the proportion of words that belong to the corresponding pattern and are assigned an incorrect head word according to the gold-standard dependency tree. The last two columns give the absolute reduction (-) or increase (+) made by the joint model compared with the pipeline model.

For the correct POS patterns, the joint model increases their percentages except for "NN" and "JJ", which means more words of the corresponding POS tags are correctly recalled. More importantly, the parsing error rates of such patterns are all

[3]DEG and DEG are the two POS tags for the frequently used auxiliary word "的" (dē, translated as "of" or "that" in English) in Chinese. The associative "的" is tagged as DEG, such as "(父亲/father) (的) (眼睛/eyes)" (eyes of the father); whereas the one in a relative clause is tagged as DEG, such as "(他/he) (取得/made) (的) (进步/progress)" (progress that he made).

TABLE IX
RESULTS ON THE TEST SET OF CTB7

		UAS	RA	CM	TA
Joint	JO3	**83.18**	79.81	**29.42**	94.54
	JO2	83.11	**80.19**	29.40	**94.61**
	JO1	79.68	71.17	22.35	94.37
Pipeline	O3	**82.50**	**80.90**	28.11	94.01
	O2	82.40	79.54	**28.43**	
	O1	78.92	70.96	21.60	
Gold POS	O3	88.44	**83.43**	**35.97**	100.0
	O2	**88.45**	82.89	35.93	
	O1	85.11	74.39	27.72	

reduced. This implies that *the joint model can do better given the correct tags than the pipeline model.*

For the syntax-sensitive POS error patterns, the parsing error rates are largely increased. However, the proportion of such error patterns are largely reduced at the same time. In other words, the joint model does much better in resolving such syntax-sensitive ambiguities, but produces much worse parsing results when mis-tagged. This indicates that *the syntax-sensitive POS tags become more accurate and the parsing component puts more trust in them under the joint framework*.

For the syntax-insensitive POS error patterns, it seems that the joint approach has slight and somewhat unstable impact on them. It sounds reasonable since such POS tags play similar syntactic roles and therefore the dependency structure is unable to provide useful feedback to their disambiguation.

In summary, we can conclude that *the joint model helps resolve syntax-sensitive POS ambiguities and the POS tags become more reliable and helpful for parsing in return.* This is the fundamental reason for the performance improvements.

D. Experiments on CTB7

We conduct experiments on CTB7 to investigate whether the joint models can improve the performance on larger datasets. Table IX shows the results. We can draw the following findings which are consistent with those on CTB5.

- Error propagation from POS tagging degrades parsing accuracy by about 6.0%.
- The joint method significantly boosting tagging accuracy by about 0.6% ($p < 10^{-3}$), which is an error reduction of 10%.
- The joint method significantly improves the parsing accuracy by 0.7% ($p < 10^{-3}$), which is an error reduction of 4%.

Different from the results on CTB5, the third-order pipeline and joint models slightly but insignificantly outperform the second-order counterparts on the parsing accuracy on the larger dataset ($p > 0.1$), which may confirm our earlier discussions that the third-order features are ineffective on CTB5 due to the data sparseness problem.

V. RELATED WORK

Joint modeling has been a popular and effective approach to simultaneously solve related tasks. Recently, many successful joint models have been proposed, such as joint tokenization and POS tagging [8], [25], [26], joint lemmatization and POS tagging [27], joint POS tagging and named entity recognition [28], joint named entity recognition and parsing [29], joint named entity recognition and normalization for Tweets [30] joint entity and event coreference resolution [31], joint tokenization and parsing [32], [33], joint parsing and semantic role labeling (SRL) [34], joint word sense disambiguation and SRL [35], joint tokenization and machine translation (MT) [36], [37], joint parsing and MT [38], and joint parsing and word alignment [39]. Note that the aforementioned "parsing" all refer to *constituent parsing*. As far as we know, there are few successful models for jointly solving *dependency parsing* and other tasks before our work in [7].

Theoretically, [16] proposed a preliminary idea of extending the decoding algorithm for dependency parsing to handle polysemy. Here, word senses can be understood as POS-tagged words. Based on this idea, [17] extended his second-order graph-based parsing model and conducted a primitive study on joint POS tagging and dependency parsing. To make inference tractable, they used top two candidate POS tags for each word based on a baseline ME based tagger. Their experiments on English Penn Treebank showed that parsing accuracy can be improved from 91.5% to 91.9%. However, they found that the joint model was unbearably time-consuming. In this work, we present a systematical and in-depth study on the joint optimization of POS tagging and dependency parsing for Chinese. Several DP based joint decoding algorithms are designed to incorporate rich POS tagging and syntactic features. Moreover, we propose an effective marginal probability based pruning strategy to constrain the POS tag space. Our experiments on two standard Chinese datasets show that the joint models significantly outperform their counterparts on both POS tagging and parsing accuracies.

[40] applied loopy belief propagation (LBP) to dependency parsing and pointed out that LBP could naturally represent POS tags as latent variables so that the POS tags can be inferred jointly with the parse. [41] proposed a LBP based model to study joint morphological disambiguation and dependency parsing for morphologically-rich languages including Latin, Czech, Ancient Greek, and Hungarian. For these languages, morphological analysis requires the disambiguation of POS tags, gender, case, etc. They showed that the joint model could well capture the interaction between morphology and syntax and achieve gains on both subtasks. [42] proposed dual decomposition (DD) for integrating different NLP subtasks at the test phase. They experimented with two cases, one integrating a constituent parser and a dependency parser, and the other integrating a phrase-structure parser and a POS tagger. Both cases show that DD can help the individual subtasks. [43] conducted an extensive comparison of LBP and DD for joint CCG supertagging and parsing at the test phase. They showed that LBP and DD achieved similar parsing accuracy improvement but had largely different convergence characteristics. Moreover, their work focuses on integrating two separately-trained sub-models, and they find that training the integrated model with LBP leads to large improvement drops.

[24] proposed a transition-based joint model for Chinese POS tagging and dependency parsing. They also gained large

improvement on parsing accuracy but slight improvement on tagging accuracy. We make comparison with their work in this paper and find that our graph-based joint models achieve the same parsing accuracy but higher tagging accuracy. [44] studied joint POS tagging and labeled non-projective dependency parsing based on the transition-based framework. Recently, [45] and [46] proposed joint models on joint word segmentation, POS tagging, and dependency parsing for Chinese by extending the transition-based parser, and [47] presented a novel decoding algorithm to integrate separately trained models for joint word segmentation, POS tagging, and constituent parsing for Chinese.

VI. Conclusions

In this paper, we have systematically investigated the issue of joint Chinese POS tagging and dependency parsing by extending state-of-the-art graph-based parsing models. We propose and compare several joint models and their corresponding decoding algorithms which can incorporate different feature sets. We also propose an effective POS tag pruning strategy which can greatly improve the decoding efficiency. The experimental results show that our joint models can significantly improve the state-of-the-art POS tagging parsing accuracies. Detailed error analysis shows that the fundamental reason for the parsing accuracy improvement is that the joint method is able to choose POS tags that are helpful and discriminative from parsing viewpoint.

In future, we would like to continue this work in the following directions. So far, we employ averaged perceptron [3] to learn the feature weights of the joint models, which equally treats the POS and syntactic features. We are interested in pursuing better training procedures to study the interaction between the two sets of features. Moreover, we will conduct detailed comparison with the transition-based joint model to gain more insights into the joint optimization of Chinese POS tagging and dependency parsing.

Acknowledgment

The authors would like to thank Meishan Zhang for helpful discussions during this work, and the three anonymous reviewers for their critical and insightful comments.

References

[1] A. Ratnaparkhi, "A maximum entropy model for part-of-speech tagging," in *Proc. EMNLP '96*, 1996, pp. 133–142.
[2] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. ICML '01*, 2001, pp. 282–289.
[3] M. Collins, "Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms," in *Proc. EMNLP '02*, 2002, pp. 1–8.
[4] X. Duan, J. Zhao, and B. Xu, "Probabilistic models for action-based Chinese dependency parsing," in *Proc. ECML/ECPPKDD*, 2007, pp. 559–566.
[5] Y. Zhang and S. Clark, "A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing," in *Proc. EMNLP '08*, 2008, pp. 562–571.
[6] L. Huang and K. Sagae, "Dynamic programming for linear-time incremental parsing," in *Proc. ACL '10*, 2010, pp. 1077–1086.
[7] Z. Li, M. Zhang, W. Che, T. Liu, W. Chen, and H. Li, "Joint models for Chinese POS tagging and dependency parsing," in *Proc. EMNLP '11*, 2011, pp. 1180–1191.
[8] Y. Zhang and S. Clark, "Joint word segmentation and POS tagging using a single perceptron," in *Proc. ACL-08: HLT*, 2008, pp. 888–896.
[9] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. Bartlett, "Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks," *J. Mach. Learn. Res.*, vol. 9, pp. 1775–1822, 2008.
[10] T. Koo and M. Collins, "Efficient third-order dependency parsers," in *Proc. ACL '10*, 2010, pp. 1–11.
[11] R. McDonald, K. Crammer, and F. Pereira, "Online large-margin training of dependency parsers," in *Proc. ACL '05*, 2005, pp. 91–98.
[12] R. McDonald and F. Pereira, "Online learning of approximate dependency parsing algorithms," in *Proc. EACL '06*, 2006, pp. 81–88.
[13] X. Carreras, "Experiments with a higher-order projective dependency parser," in *Proc. EMNLP/CoNLL*, 2007, pp. 141–150.
[14] B. Bohnet, "Top accuracy and fast dependency parsing is not a contradiction," in *Proc. COLING '10*, 2010, pp. 89–97.
[15] W. Che, V. Spitkovsky, and T. Liu, "A comparison of Chinese parsers for Stanford dependencies," in *Proc. ACL '12*, 2012, pp. 11–16.
[16] J. Eisner, "Bilexical grammars and their cubic-time parsing algorithms," in *Advances in Probabilistic and Other Parsing Technologies*. New York, NY, USA: Springer, 2000, pp. 29–62.
[17] R. McDonald, "Discriminative training and spanning tree algorithms for dependency parsing," Ph.D. dissertation, Univ. of Pennsylvania, State College, PA, USA, 2006.
[18] Y. Zhang and S. Clark, "Syntactic processing using the generalized perceptron and beam search," *Comput. Linguist.*, vol. 37, no. 1, pp. 105–151, 2011.
[19] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive aggressive algorithms," *J. Mach. Learn. Res.*, vol. 7, pp. 551–585, 2006.
[20] K. Crammer and Y. Singer, "Ultraconservative online algorithms for multiclass problems," *J. Mach. Learn. Res.*, vol. 3, pp. 951–991, 2003.
[21] N. Xue, F. Xia, F.-D. Chiou, and M. Palmer, "The Penn Chinese Treebank: Phrase structure annotation of a large corpus," *Natural Lang. Eng.*, vol. 11, no. 2, pp. 207–238, 2005.
[22] E. W. Noreen, *Computer-intensive methods for testing hypotheses: An introduction*. New York, NY, USA: Wiley, 1989, book (ISBN 0471611360).
[23] Y. Zhang and J. Nivre, "Transition-based dependency parsing with rich non-local features," in *Proc. ACL '11*, 2011, pp. 188–193.
[24] J. Hatori, T. Matsuzaki, Y. Miyao, and J. Tsujii, "Incremental joint POS tagging and dependency parsing in Chinese," in *Proc. IJCNLP '11*, 2011, pp. 1216–1224.
[25] W. Jiang, L. Huang, Q. Liu, and Y. Lü, "A cascaded linear model for joint Chinese word segmentation and part-of-speech tagging," in *Proc. ACL '08: HLT*, 2008, pp. 897–904.
[26] C. Kruengkrai, K. Uchimoto, J. Kazama, Y. Wang, K. Torisawa, and H. Isahara, "An error-driven word-character hybrid model for joint Chinese word segmentation and POS tagging," in *Proc. ACL-AFNLP '09*, 2009, pp. 513–521.
[27] K. Toutanova and C. Cherry, "A global model for joint lemmatization and part-of-speech prediction," in *Proc. ACL-AFNLP '09*, 2009, pp. 486–494.
[28] J. R. Finkel and C. D. Manning, "Nested named entity recognition," in *Proc. EMNLP '09*, 2009, pp. 141–150.
[29] J. R. Finkel and C. D. Manning, "Joint parsing and named entity recognition," in *Proc. NAACL '09*, 2009, pp. 326–334.
[30] X. Liu, M. Zhou, X. Zhou, Z. Fu, and F. Wei, "Joint inference of named entity recognition and normalization for tweets," in *Proc. ACL '12*, 2012, pp. 526–535.
[31] H. Lee, M. Recasens, A. Chang, M. Surdeanu, and D. Jurafsky, "Joint entity and event coreference resolution across documents," in *Proc. EMNLP '12*, 2012, pp. 489–500.
[32] S. B. Cohen and N. A. Smith, "Joint morphological and syntactic disambiguation," in *Proc. EMNLP-CoNLL '07*, 2007, pp. 208–217.
[33] Y. Goldberg and R. Tsarfaty, "A single generative model for joint morphological segmentation and syntactic parsing," in *Proc. ACL '08: HLT*, 2008, pp. 371–379.
[34] J. Li, G. Zhou, and H. T. Ng, "Joint syntactic and semantic parsing of Chinese," in *Proc. ACL '10*, 2010, pp. 1108–1117.
[35] W. Che and T. Liu, "Jointly modeling WSD and SRL with markov logic," in *Proc. COLING '10*, 2010, pp. 161–169.
[36] C. Dyer, "Using a maximum entropy model to build segmentation lattices for MT," in *Proc. NAACL '09*, 2009, pp. 406–414.
[37] X. Xiao, Y. Liu, Y. Hwang, Q. Liu, and S. Lin, "Joint tokenization and translation," in *Proc. COLING '10*, 2010, pp. 1200–1208.

[38] Y. Liu and Q. Liu, "Joint parsing and translation," in *Proc. 23rd Int. Conf. Comput. Linguist. (Coling '10)*, 2010, pp. 707–715.
[39] D. Burkett, J. Blitzer, and D. Klein, "Joint parsing and alignment with weakly synchronized grammars," in *Proc. NAACL '10*, 2010, pp. 127–135.
[40] D. A. Smith and J. Eisner, "Dependency parsing by belief propagation," in *Proc. EMNLP '08*, 2008, pp. 145–156.
[41] J. Lee, J. Naradowsky, and D. A. Smith, "A discriminative model for joint morphological disambiguation and dependency parsing," in *Proc. ACL '11*, 2011, pp. 885–894.
[42] A. M. Rush, D. Sontag, M. Collins, and T. Jaakkola, "On dual decomposition and linear programming relaxations for natural language processing," in *Proc. EMNLP '10*, 2010, pp. 1–11.
[43] M. Auli and A. Lopez, "A comparison of loopy belief propagation and dual decomposition for integrated CCG supertagging and parsing," in *Proc. ACL '11*, 2011, pp. 470–480.
[44] B. Bohnet and J. Nivre, "A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing," in *Proc. EMNLP '12*, 2012, pp. 1455–1465.
[45] J. Hatori, T. Matsuzaki, Y. Miyao, and J. Tsujii, "Incremental joint approach to word segmentation, POS tagging, and dependency parsing in Chinese," in *Proc. ACL '12*, 2012, pp. 1045–1053.
[46] Z. Li and G. Zhou, "Unified dependency parsing of Chinese morphological and syntactic structures," in *Proc. EMNLP '12*, 2012, pp. 1445–1454.
[47] X. Qian and Y. Liu, "Joint Chinese word segmentation, POS tagging and parsing," in *Proc. EMNLP '12*, 2012, pp. 501–511.

Zhenghua Li received his Bachelor degree, Master degree, and Ph.D. degree in computer science from Harbin Institute of Technology in 2006, 2008, and 2013, respectively. He joined Soochow University afterwards and is currently an assistant professor in the university. His current research interests include parsing, machine translation, and machine learning.

Min Zhang received his Bachelor degree and Ph.D. degree in computer science from Harbin Institute of Technology in 1991 and 1997, respectively. He joined the Soochow University in 2012 and is currently a distinguished professor at the university. From 1997 to 1999, he worked as a postdoctoral research fellow in Korean Advanced Institute of Science and Technology in South Korea. He began his academic and industrial career as a researcher at Lernout & Hauspie Asia Pacific (Singapore) in 1999. He joined Infotalk Technology (Singapore) as a researcher in 2001 and became a senior research manager in 2002. He joined the Institute for Infocomm Research (Singapore) in 2003. His current research interests include machine translation, natural language processing, information extraction, large scale text processing, intelligent computing and machine learning. He has co-authored 150 papers in leading journals and conferences. He is the vice president of COLIPS (2011-2013), a steering committee member of PACLIC (2011-now), an executive member of AFNLP (2013-2014) and a member of ACL (2006-).

Wanxiang Che received the B.S. and Ph.D. degrees in computer science from Harbin Institute of Technology (HIT), Harbin, China, in 2002 and 2008, respectively. He is an Associate Professor in the School of Computer Science and Technology, HIT. His current research interests include natural language processing and information retrieval.

Ting Liu received the Ph.D. degrees in computer science from Harbin Institute of Technology (HIT), Harbin, China, in 1998. He is a Full Professor in the School of Computer Science and Technology, HIT. His current research interests include natural language processing, information retrieval, and social computing.

Wenliang Chen received his Bachelor degree in Mechanical Engineering and Ph.D. degree in computer science from Northeastern University, China in 1999 and 2005, respectively. He joined Soochow University since 2013 and is currently a professor in the university. Prior to joining Soochow University, he was a research scientist in the Institute for Infocomm Research of Singapore from 2011 to 2013. From 2005 to 2010, he worked as an expert researcher in NICT, Japan. His current research interests include parsing, machine translation, and machine learning.