High-order Joint Constituency and Dependency Parsing

Yanggan Gu☆, Yang Hou☆, Zhefeng Wang☾, Xinyu Duan☾, Zhenghua Li☆*
☆School of Computer Science and Technology, Soochow University, China
☾Huawei Inc., China
yanggangu@outlook.com, yhou1@stu.suda.edu.cn
{wangzhefeng, duanxinyu}@huawei.com, zhli13@suda.edu.cn

Abstract

This work revisits the topic of jointly parsing constituency and dependency trees, i.e., to produce compatible constituency and dependency trees simultaneously for input sentences, which is attractive considering that the two types of trees are complementary in representing syntax. The original work of Zhou and Zhao (2019) performs joint parsing only at the inference phase. They train two separate parsers under the multi-task learning framework (i.e., one shared encoder and two independent decoders). They design an ad-hoc dynamic programming-based decoding algorithm of $O(n^5)$ time complexity for finding optimal compatible tree pairs. Compared to their work, we make progress in three aspects: (1) adopting a much more efficient decoding algorithm of $O(n^4)$ time complexity, (2) exploring joint modeling at the training phase, instead of only at the inference phase, (3) proposing high-order scoring components to promote constituent-dependency interaction. We conduct experiments and analysis on seven languages, covering both rich-resource and low-resource scenarios. Results and analysis show that joint modeling leads to a modest overall performance boost over separate modeling, but substantially improves the complete matching ratio of whole trees, thanks to the explicit modeling of tree compatibility.

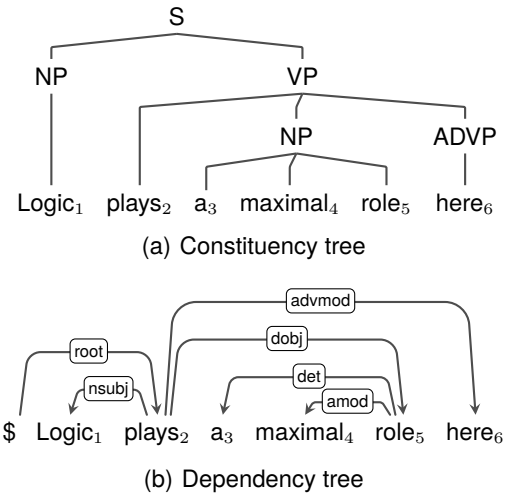
Keywords: joint modeling, constituency parsing, dependency parsing, high-order

1. Introduction

As one of the most fundamental and long-standing NLP tasks, syntactic parsing aims to reveal how sentences are syntactically structured. Among many paradigms for representing syntax, constituency trees (c-trees) and dependency trees (d-trees) are the most popular and have gained tremendous research attention in both data annotation and parsing techniques. Figure 1 gives example trees.

As is well known, c-trees and d-trees capture syntactic structure from different yet complementary perspectives (Abeillé, 2003). On the one hand, c-trees can clearly illustrate how sentences are composed hierarchically. Constituents, especially major phrases like NPs and VPs, are often self-evident and thus easily agreed upon by people. On the other hand, d-trees emphasize pairwise syntactic (sometimes even semantic) relationships between words, i.e., what role (function) the modifier word plays for the head word. It is usually more simple and flexible to draw dependency links than to add constituent nodes.

Therefore, it can be an attractive and useful feature that a parsing model outputs both c-trees and d-trees at the same time. Of course the two trees must be compatible with each other. Basically, **compatibility** means that for any constituent, only the single head word can compose dependency links, either inwards or outwards, with words outside the constituent. Please note that this work and most previous works consider only conventional continuous c-trees and projective d-trees.[1]



(a) Constituency tree

(b) Dependency tree

Figure 1: Constituency and dependency trees.

From the modeling perspective, parsing techniques have achieved immense progress, thanks to the development of deep learning techniques, especially of the pre-trained language models (PLMs) (Peters et al., 2018; Devlin et al., 2019). One amazing characteristic is that constituency parsing and

*Corresponding author.

[1] In the sentence "A hearing is scheduled tomorrow on this issue", "A hearing on this issue" composes a typical discontinuous constituent, which also results in a non-projective dependency tree.

LREC-COLING 2024, pages 8144–8154

dependency parsing have a nearly identical model architecture nowadays. Under the graph-based parsing framework, besides the encoder, the scoring components are the same as well if we treat a constituent as a link between the beginning word and the end word (Zhang et al., 2020b). The only difference lies in the decoding algorithms for finding optimal c- or d-trees. To some extent, such development stimulates research on joint constituency and dependency parsing (Zhou and Zhao, 2019).

The original work of Zhou and Zhao (2019) performs joint parsing *only at the inference phase*. First, they use the simplified head-driven phrase structure grammar (HPSG) trees to encode both c- and d-trees simultaneously. In fact, their HPSG trees are inherently the same as lexicalized constituency trees (l-trees), in which each constituent is annotated with its head word (Collins, 2003). Figure 2 gives an example l-tree. Then, they design an ad-hoc decoding algorithm of $O(n^5)$ time complexity for finding optimal l-trees. Under the multi-task learning (MTL) framework, they use one shared encoder, and two separate decoders for the two parsing tasks respectively. Thus there is no explicit interaction between the two types of parse trees.

In this paper, we carry forward this interesting research topic from the following three aspects.

1. Inspired by the recent work of Lou et al. (2022) on nested named entity recognition (NER), we employ the Eisner-Satta algorithm (Eisner and Satta, 1999) of $O(n^4)$ time complexity for finding optimal l-trees, which runs very efficiently on GPUs after proper batchification (Zhang et al., 2020a). Experiments show that our model is about 2.5$\times$ faster than that of Zhou and Zhao (2019).

2. We propose to jointly model the two parsing tasks at both training and inference phases.

3. We propose high-order scoring components so that the two types of parse trees can interact with and influence each other more tightly.

We conduct experiments on benchmark datasets in seven languages, covering both rich-resource and low-resource scenarios. The results and analysis lead to several interesting findings. We release our code at https://github.com/EganGu/high-order-joint-parsing.

2. Related Work

Before the deep learning era, there exist three competitive families of constituency parsing approaches, i.e., lexicalized PCFG parsing (Collins, 1999), unlexicalized PCFG parsing (Petrov et al., 2006), and discriminative shift-reduce parsing (Zhu et al., 2013). Due to the importance of head word features, all three families are more or less related to our work, in the sense that the parser may output d-trees as byproduct at the inference phase.

Lexicalized PCFG parsers generate a c-tree in a head-centered, top-down manner (Collins, 1999). The head token of each constituent is settled when parsing is finished. Therefore, it is straightforward to acquire *unlabeled* d-trees.

The unlexicalized parser of Klein and Manning (2003) splits each constituent labels into multiple sub-labels according to linguistic heuristics. For N-ary production rules, they markovize out from the head child, making it feasible to recover unlabeled d-trees likewise.

The discriminative shift-reduce parser of Crabbé (2015) makes heavy use of head tokens for composing features, and therefore is also capable of producing unlabeled d-trees.

Our work is also closely related to Klein and Manning (2002), who factorize a l-tree into an unlexicalized c-tree and a d-tree, corresponding to two generative models that are separately trained. They propose an efficient yet inexact A* algorithm for finding the optimal l-tree, instead of using the Eisner-Satta algorithm.

In the deep learning era, besides Zhou and Zhao (2019), there exist two works that tackle constituency parsing and dependency parsing simultaneously.

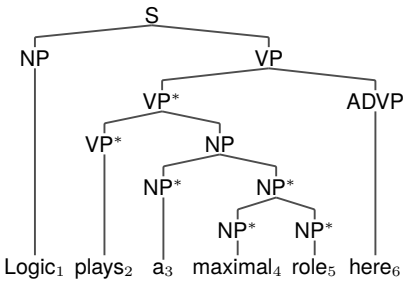
Strzyz et al. (2019) transform both constituency parsing and dependency parsing into sequence labeling tasks. However, the parsing performance lags behind state-of-the-art models by large margins. Fernández-González and Gómez-Rodríguez (2022) transform constituency parsing into a dependency parsing task and employ a pointer network architecture to obtain the dependency trees.

The above two works have two key differences from our work. First, during training, both works employ the MTL framework, hence blocking explicit interaction between two sub-modules. Second and more importantly, both works do not consider the compatibility of output c-trees and d-trees at the inference phase.

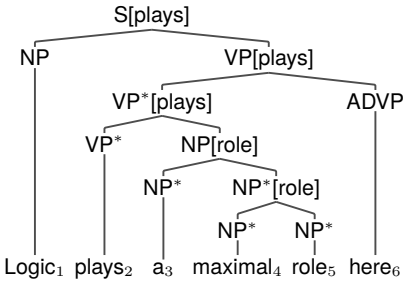
3. Lexicalized Tree Representation

Given an input sentence $\boldsymbol{x} = w_1 \dots w_n$, we use $\{(i,j,l), 0 \leq i,j \leq n\}$ to denote a c-tree $\boldsymbol{c}$, and $\{(h \rightarrow m, r), 0 \leq h,m \leq n\}$ to a d-tree $\boldsymbol{d}$. For a c-tree, (i,j,l) denotes a constituent spanning $w_i \dots w_j$ and labeled as $l \in \mathcal{L}$, while for a d-tree, $(h \rightarrow m, r)$ denotes a dependency from the head word w_h to the modifier word w_m and labeled as $r \in \mathcal{R}$.

It is a natural way to encode both a c-tree $\boldsymbol{c}$ and a d-tree $\boldsymbol{d}$ into an l-tree as the joint representation (Collins, 2003). Figure 2 gives examples. We con-



(a) A regular c-tree (head-binarized)



(b) A lexicalized c-tree

Figure 2: Example c-trees. The head word of each lexicalized constituent is marked by $[\cdot]$.

vert the original c-tree into Chomsky normal form (CNF), as shown in Figure 2(a), which is required by the graph-based parsing model adopted in this work. Figure 2(b) presents the corresponding l-tree, in which the head words are decided by the d-tree in Figure 1(b).

Formally, we denote an l-tree as $\boldsymbol{t} = \{(i, j, h, l)\}$, where (i, j, h, l) represents a lexicalized constituent spanning $w_i \dots w_j$ with a head word $w_h (i \leq h \leq j)$ and a label $l \in \mathcal{L}$. We use $\mathrm{NP}[\mathrm{role}]_{3,5}$ as a simplified notation of $(3, 5, 5, \mathrm{NP})$.

Constituent-to-dependency conversion and the compatibility rationale. To the best of our knowledge, many dependency treebanks are automatically converted from constituent treebanks, instead of constructed from scratch. Usually, the const-to-dep conversion process consists of two major steps: 1) determining unlabeled dependencies using head-finding rules, and 2) determining dependency labels (de Marneffe et al., 2006). In the first step, the head-finding rules are applied on c-trees to form l-trees, in which each constituent has only one head word. Further, unlabeled d-trees are constructed according to head words. Such conversion process determines that the resulting dependency trees are compatible with the original constituent trees.

Head-binarization. As shown earlier, given a c-tree $\boldsymbol{c}$ and a d-tree $\boldsymbol{d}$, we first convert $\boldsymbol{c}$ into CNF before constructing an l-tree $\boldsymbol{t}$, which is required by the CKY algorithm.

To avoid incompatibility after CNF conversion, we adopt the head-binarization strategy (Collins, 2003), instead of left- or right-binarization. Basically, head-binarization dynamically selects left- or right-binarization to comply with a given d-tree. For example, the constituent $\mathrm{VP}[\mathrm{plays}]_{2,6}$ in Figure 2(b) is left-binarized, whereas $\mathrm{NP}[\mathrm{role}]_{3,5}$ is right-binarized. If both binarization choices are feasible, we prioritize left-binarization.

Recovering constituency/dependency trees. It is obvious that an l-tree becomes a c-tree when discarding the head words. Meanwhile, since l-trees are strictly binarized, we can construct the corresponding d-trees according to the head words.

Moreover, please note that dependency relations are not encoded in an l-tree $\boldsymbol{t}$, and are handled via a separate labeling step (see Section 4.1), which is now common practice in parsing models (Zhang et al., 2020a).

Labeling compatibility. Since the dependency relations are not contained in l-trees, our joint representation can only guarantee that the encoded c- and d-trees are compatible in an unlabeled fashion. Given that the dependency relations are usually derived from c-trees, it leads to an interesting question: *is it possible to make the parsed c- and d-trees compatible at the labeled level according to the const-to-dep conversion process?* We believe the answer is no. We give our explanations as follows.

During the two steps of the conversion process, the label determination algorithm makes use of non-local constituent features (i.e., multiple constituents and their labels). Therefore, it is actually difficult to impose such labeling compatibility into our dynamic programming decoding (described in Section 4.3). Moreover, some const-to-dep conversion processes make use of not only structural constituent labels like NP/VP, but also functional labels like -SBJ/OBJ (Surdeanu et al., 2008; Seddah et al., 2014). Functional labels are usually overlooked in constituency parsing research.

4. The Joint Parsing Approach

In this section, we first describe a first-order model, and then introduce a second-order extension.

To facilitate explanation, we give alternative notations. We use $\boldsymbol{y} = \{(i, j, h)\}$ to denote an unlabeled l-tree, i.e., $\boldsymbol{t}$ without the constituent labels, $\boldsymbol{l}$ to denote the set of constituent labels in an l-tree $\boldsymbol{t}$, and $\boldsymbol{r}$ to denote the set of dependency relations in a d-tree $\boldsymbol{d}$. Then we have the following equivalent notations.

$$(\boldsymbol{c}, \boldsymbol{d}) \equiv (\boldsymbol{t}, \boldsymbol{r}) \equiv (\boldsymbol{y}, \boldsymbol{l}, \boldsymbol{r})$$

Please kindly note that we assume that $\boldsymbol{c}$ is binarized and conforms to CNF, which guarantees that a unique unlabeled dependency tree $\boldsymbol{d}$ can be induced from $\boldsymbol{t}$ and $\boldsymbol{y}$.

4.1. Two-stage Parsing and First-order Factorization

Following Dozat and Manning (2017) and Zhang et al. (2020b), we adopt the two-stage parsing strategy, which has been proven to be able to simplify the model architecture and improve efficiency, without hurting performance.

Stage I: Lexicalized Bracketing. Given $\boldsymbol{x}$, the goal of the first stage is to find an optimal unlabeled l-tree $\boldsymbol{y}$, whose score is:

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j)\in \boldsymbol{c}} s^c(i,j) + \sum_{h\rightarrow m \in \boldsymbol{d}} s^d(h,m) \quad (1)$$

where $s^{c/d}$ denote the scores of unlabeled constituents and unlabeled dependencies, respectively. Here we adopt the first-order factorization, i.e., the scores of constituents and dependencies being mutually independent. We present second-order factorization in Section 4.4.

Given all $s^c(i,j)$ and $s^d(h,m)$, the decoding algorithm determines the 1-best l-tree.

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x})} s(\boldsymbol{x}, \boldsymbol{y}) \quad (2)$$

where $\mathcal{Y}(\boldsymbol{x})$ denotes the set of all legal l-trees for $\boldsymbol{x}$.

Stage II: Labeling. Given unlabeled constituents/dependencies in $\hat{\boldsymbol{y}}$ obtained in the first stage, this stage independently predicts labels for them.

$$\hat{l} = \arg\max_{l\in\mathcal{L}} s^c(i,j,l) \quad (3)$$

$$\hat{r} = \arg\max_{r\in\mathcal{R}} s^d(h,m,r) \quad (4)$$

4.2. Training Loss

Given a training instance $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{r})$, the training loss consists of two parts, corresponding to the two stages.[2]

$$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{r}) = L^{\text{bracket}}(\boldsymbol{x}, \boldsymbol{y}) + L^{\text{label}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{r}) \quad (5)$$

For the first stage, we adopt the max-margin loss (Taskar et al., 2004)[3], which is based on the margin between the scores of the gold-standard l-tree $\boldsymbol{y}$ and the predicted one $\hat{\boldsymbol{y}}$.

$$L^{\text{bracket}} = \max\left(0, \max_{\boldsymbol{y}\neq\hat{\boldsymbol{y}}}\left(s(\boldsymbol{y}) - s(\hat{\boldsymbol{y}}) + \Delta(\boldsymbol{y}, \hat{\boldsymbol{y}})\right)\right) \quad (6)$$

where Δ is the Hamming distance, which we set as the number of incorrect dependencies and constituents.

For the second-stage loss, following Dozat and Manning (2017), we employ local cross-entropy losses, i.e., the sum of classification loss for labeling all constituents and dependencies in the gold-standard c-tree and d-tree.

Besides, we tried to balance the losses from the two stages via weighted summation, which we found has negligible influence on performance according to our preliminary experiments.

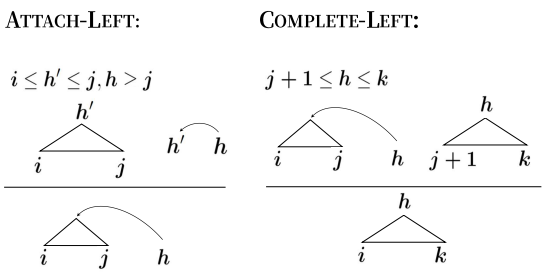


Figure 3: Deduction rules for Eisner-Satta algorithm (Eisner and Satta, 1999). We show only the leftward rules, omitting the symmetric rightward one as well as initial conditions for brevity.

4.3. Inference: Batchified Eisner-Satta

Zhou and Zhao (2019) propose a naive CKY-style algorithm in $O(n^5)$ complexity to the simplified HPSG inference, which is highly time-consuming. For l-trees, Eisner and Satta (1999) propose a relatively fast inference that reduces the complexity to $O(n^4)$ by merging dependency arcs in advance. In this paper, we adopt the Eisner-Satta algorithm to jointly infer the c- and d-trees. The deduction rules (Pereira and Warren, 1983) of the algorithm are illustrated in Figure 3.

To further increase the speed of inference, following Zhang et al. (2020a), we batchify the Eisner-Satta algorithm to fully utilize GPUs, as shown in Algorithm 1. The basic idea is handling spans of the same length simultaneously. The time complexity of our algorithm is practically linear.

4.4. Second-order Extension

In order to further bridge the c- and d-trees for more in-depth joint modeling, we extend the score

[2] At a training step, the cumulative loss of all instances in a mini-batch is divided by the total number of tokens.

[3] We also try using the CRF loss for the l-tree and find it to be slightly inferior to the max-margin loss in terms of performance and training speed.

Algorithm 1 The Eisner-Satta Decoding Algorithm.

define: $\alpha, \beta \in \mathbb{R}^{n \times n \times n \times B}$ ▷ B is #sents in a batch
initialize: all $\alpha^{:,:,:} = 0$, $\beta^{:,:,:} = 0$
for $w = 1$ **to** n **do** ▷ span width
  Batchify: $0 \leq i, j \leq n, j = i + w, k, h$
  $\alpha^{i,j,:} = s^c(i,j) + \max_{i \leq k < j} \left(\alpha^{i,k,:} + \beta^{k+1,j,:}, \beta^{i,k,:} + \alpha^{k+1,j,:}\right)$ ▷ add $s(i,j,:)$ to α/β for the 2rd-order extension
  $\beta^{i,j,:} = \max_{i \leq h \leq j} \left(\alpha^{i,j,h} + s^d(:,h)\right)$
end for
return $\alpha^{0,n-1,0}$

decomposition of $\boldsymbol{y}$ to the second-order case. We define two types of span structure, as

- Headed spans: a span (i, j) with h being the lexical head (parent, or root), in which $i \leq h \leq j$.
- Hooked spans: a span (i, j) with h being the lexical grandparent, in which $h < i$ *or* $h > j$.

Both headed spans and hooded spans can be denoted as (i, j, h), and can be distinguished by the position of h.

As shown in Figure 3, the left part illustrates the construction of a hooked span, whereas the right part illustrates how to create a headed span.

We further incorporate scores of headed spans and hooked spans into the basic first-order model.

$$s^{2o}(\boldsymbol{y}) = s(\boldsymbol{y}) + \sum_{(i,j,h) \in \boldsymbol{y}} s^{\text{span}}(i,j,h) \tag{7}$$

where $s(i, j, h) \in \mathbb{R}^{n \times n \times n}$.

5. Model Architecture

In this section, we introduce how to compute scores of lexicalized subtrees.

Encoding. Given a sentence $\boldsymbol{x} = w_0, \ldots, w_{n+1}$, where w_0 and w_{n+1} denote the ‹*bos*› (begin of a sentence) and ‹*eos*› (end of a sentence) tokens respectively, the corresponding hidden representations $\boldsymbol{e}$ can be obtained via BERT encoder (Devlin et al., 2019), without cascading word embedding and LSTM layers for simplicity.

Scoring. We employ the same scoring functions as Zhang et al. (2020a,b) to compute span scores $s^c(i, j)$ and arc scores $s^d(h, m)$.

$$r_i^{\text{left/right}} = \text{MLP}^{\text{left/right}}(\overrightarrow{e_i} \oplus \overleftarrow{e_{i+1}}) \tag{8}$$
$$r_i^{\text{head/mod}} = \text{MLP}^{\text{head/mod}}(\overrightarrow{e_i} \oplus \overleftarrow{e_i}) \tag{9}$$
$$s^c(i,j) = \left[r_i^{\text{left}} \oplus 1\right]^{\text{T}} \boldsymbol{W}^c r_j^{\text{right}} \tag{10}$$
$$s^d(h,m) = \left[r_m^{\text{mod}} \oplus 1\right]^{\text{T}} \boldsymbol{W}^d r_h^{\text{head}} \tag{11}$$

where MLPs are applied to obtain lower k-dimensional vectors; $r_i^{\text{head/mod}}$ are the representation vectors of w_i as a head/modifier respectively, and analogously, $r_i^{\text{left/right}}$ are the left/right boundary representation of w_i; $\overrightarrow{e_i}/\overleftarrow{e_i}$ are hidden representations of w_i in different contextual orientations[4]. $\mathbf{W}^{d/c} \in \mathbb{R}^{(k+1)\times k}$ are trainable parameters.

For second-order extension, we follow Yang and Tu (2022b) to calculate scores of headed spans and hooked spans.

$$r_i^{\text{word}} = \text{MLP}^{\text{word}}(\overrightarrow{e_i} \oplus \overleftarrow{e_i}) \tag{12}$$
$$h_{i,j}^{\text{span}} = [\overrightarrow{e_i} \oplus \overleftarrow{e_{i+1}}] - [\overrightarrow{e_j} \oplus \overleftarrow{e_{j+1}}] \tag{13}$$
$$r_{i,j}^{\text{span}} = \text{MLP}^{\text{span}}(h_{i,j}^{\text{span}}) \tag{14}$$
$$s^{\text{span}}(i,j,h) = \left[r_h^{\text{word}} \oplus 1\right]^{\text{T}} \boldsymbol{W}^{\text{span}} \left[r_{i,j}^{\text{span}} \oplus 1\right] \tag{15}$$

where $\boldsymbol{W}^{\text{span}} \in \mathbb{R}^{(k+1)\times(k+1)}$. Please notice that the computation of scores of headed spans and hooked spans share the same parameters. Our preliminary experiments show that the performance changes little if we use separate parameters. This may indicate that the model can distinguish the two types of scores only according to the position of h.

For the second-stage labeling tasks, we follow Zhang et al. (2020a,b) to calculate dependency and constituency labels scores.

6. Experiments

6.1. Settings

Datasets. We conduct experiments on the English Penn Treebank (PTB) (Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor, 1999), the Chinese Penn Treebank 5.1 (CTB) (Martha Palmer, Fu-Dong Chiou, Nianwen Xue, and Tsan-Kuang Lee, 2005) and the SPMRL datasets (Seddah et al., 2014).

For PTB, we follow the standard splits, i.e., sections 02-21 for training, section 22 for development and section 23 for testing. The d-trees are obtained

[4]For BERT encoder, we split each $e_i \in \boldsymbol{e}$ in half to construct each $(\overrightarrow{e_i}, \overleftarrow{e_i})$ the same as Kitaev and Klein (2018).

	Train	Dev	Test
	Rich resource		
English	39.8K (99.8)	1.7K (99.7)	2.4K (99.9)
Chinese	16.1K (99.9)	0.8K (99.8)	1.9K (99.8)
French	14.7K (99.5)	1.2K (99.6)	2.5K (99.7)
Korean	23.0K (100.0)	2.1K (100.0)	2.3K (100.0)
	Low resource		
Hebrew	5.0K (94.9)	0.5K (95.0)	0.7K (96.9)
Polish	6.6K (95.1)	0.8K (94.1)	0.8K (94.2)
Swedish	5.0K (92.0)	0.5K (88.6)	0.7K (94.1)

Table 1: Number of sentences and percentage of compatible sentences (in parentheses).

with Stanford Typed Dependencies (SD) (de Marneffe et al., 2006) using the Stanford parser v3.3.0[5].

For CTB, we use the same split as Zhang and Clark (2008). The c-trees are converted to the corresponding d-trees using the Penn2Malt tool[6].

For SPMRL, we adhere to the default data split and utilize the provided parallel constituency and dependency treebanks. We specifically choose five languages from SPMRL—French, Hebrew, Korean, Polish, and Swedish—where d-trees are well compatible with c-trees. Based on different characteristics of languages, the multilingual d-trees are obtained from various const-to-dep conversion processes.

Table 1 presents the data scale and compatibility of different languages. The languages are categorized into two scenarios: rich- and low-resource. We notice that in almost all incompatible cases, multiple words within the same constituent are headed by words outside the constituent, leading to the failure of building valid l-trees. To address this issue, we remove all incompatible instances from the training sets.

Evaluation metrics. For dependency parsing, we adopt unlabeled and labeled attachment scores (UAS/LAS) as metrics. For constituency parsing, we adopt the standard constituent-level labeled precision (P), recall (R), and F1-score as metrics. Consistent with Zhou and Zhao (2019), we omit all punctuation for dependency parsing.

Models. We conduct experiments with our joint parsing model: **Joint1o** and **Joint2o**, which both perform lexicalized modeling but with the first/second-order scores separately. To facilitate a comprehensive comparison, we introduce two kinds of baseline models, using the same network architectures as joint models:

- **Separate parsing models (SEP)**. We train two separate constituency and dependency parsers. Following joint models, we also apply max-margin loss for each task, setting their Hamming distance to the total number of mismatched constituents and dependency arcs, respectively. At the inference phase, we use the CKY algorithm for constituency parsing and the Eisner algorithm for dependency parsing.

- **Multi-task learning model (MTL)**. Similar to Zhou and Zhao (2019), we use the MTL framework at the training phase. The loss of MTL is the sum of losses for SEP. We use the head-binarization since preliminary experiments show that is superior to left- and right-binarization. Meanwhile, we apply the Eisner-Satta algorithm to perform joint parsing at the inference phase and ensure the compatibility of parsing results.

Hyper-parameters. We employ *bert-large-cased* for English, *bert-base-chinese* for Chinese, and *bert-base-multilingual-cased* for SPMRL. We mainly adopt the same hyper-parameters of parsers from Zhang et al. (2020a,b). For the second-order model, we set the dimensions of $r_i^{\text{word}}/r_{i,j}^{\text{span}}$ to 500. We report results averaged over 3 runs with different random seeds for all experiments.

6.2. Main Results

Table 2 presents the results of our model study on multilingual test sets. On average, compared to SEP, all joint models (including MTL, Joint1o, and Joint2o) showed a substantial improvement (>0.2) for dependency parsing; and only MTL showed little impact on constituency parsing, where Joint1/2o instead slightly decreased the performance. In particular, Polish is a counterexample, with a trend just the opposite of the others.

Between the joint models, we can see that Joint1o performs close to MTL in dependency parsing but lower in constituency parsing. Meanwhile, Joint2o incorporating higher-order features outperformed both Joint1o and MTL in dependency parsing. Even though Joint2o is higher than the first-order counterpart on constituency parsing, it is still lower than MTL.

The above findings bring three insights: 1) joint parsing at the inference phase can indeed help dependency parsing and has little impact on constituency parsing; 2) further joint modeling at the training phase does not improve performance in first-order cases, 3) high-order modeling leads to tiny but steady improvements on both constituency and dependency parsing.

[5]https://nlp.stanford.edu/software/lex-parser.shtml

[6]https://cl.lingfil.uu.se/~nivre/research/Penn2Malt.html

	English		Chinese		French		Hebrew		Korean		Polish		Swedish		*Average*	
	LAS	F1	LAS	F1	LAS	F1	LAS	F1	LAS	F1	LAS	F1	LAS	F1	LAS	F1
SEP	95.55	95.92	90.55	90.79	89.31	87.51	85.12	**93.15**	89.67	89.44	**91.20**	96.21	87.94	89.70	89.91	91.82
MTL	95.59	**95.97**	90.86	**90.83**	89.42	**87.58**	85.77	93.09	**89.93**	**89.46**	90.83	96.33	88.11	**89.73**	90.07	**91.86**
Joint1o	95.62	95.76	90.88	90.77	89.50	87.25	85.63	92.93	89.83	89.18	90.80	96.29	88.32	89.52	90.08	91.67
Joint2o	**95.64**	95.80	**90.96**	90.76	**89.52**	87.48	**85.83**	92.96	89.83	89.21	91.15	**96.37**	**88.38**	89.69	**90.19**	91.75

Table 2: Results on multilingual test sets (including the PTB, CTB and SPMRL).

	Dependency		Constituency		
	UAS	LAS	P	R	F1
M&H21†	96.66	95.01	–	–	–
Y&T22†	**97.24**	**95.73**	**96.19**	**95.83**	**96.01**
HPSG	97.00	95.43	95.98	95.70	95.84
F&G22	96.97	95.46	–	–	95.23
MTL	97.14	95.59	96.14	95.80	95.97
Joint2o	97.17	95.64	95.95	95.65	95.80

Table 3: Comparison with previous results on PTB-test. HPSG: Zhou and Zhao (2019). M&H21: Mohammadshahi and Henderson (2021). F&G22: Fernández-González and Gómez-Rodríguez (2022). Y&T22: Yang and Tu (2022a,b). † indicates using only the constituency or dependency treebank.

	Dependency		Constituency		
	UAS	LAS	P	R	F1
Zhang+20†	91.71	90.38	**91.00**	90.40	90.70
MTL	92.11	90.86	90.75	**90.91**	**90.83**
Joint2o	**92.21**	**90.96**	90.73	90.76	90.76
			w/ gold POS tags		
Zhang+20†	92.43	92.04	93.00	92.98	93.00
M&H21†	92.98	91.18	–	–	–
Y&T22†	93.33	92.30	–	–	–
MTL	93.16	92.78	**93.13**	93.23	93.18
Joint2o	**93.36**	**92.97**	**93.13**	**93.25**	**93.19**

Table 4: Comparison with previous results on CTB-test. Zhang+20: Zhang et al. (2020a,b). M&H21: Mohammadshahi and Henderson (2021). Y&T22: Yang and Tu (2022a). † indicates using only the constituency or dependency treebank.

6.3. Comparison with Previous Works

Table 3, 4 and 5 compare our MTL and Joint2o with the existing state-of-the-art of both the separate and joint parsing on test sets. The performance gap between our parsers and recent state-of-the-art parsers is negligibly small.

On CTB, we re-run the code released by Zhang et al. (2020a,b), which provides the two strong TreeCRF parsers for c- and d-trees, to reproduce their results. We also note that reporting the results of using gold Part-Of-Speech (POS) tags for CTB is a more prevalent practice (Zhou and Zhao, 2019; Mohammadshahi and Henderson, 2021; Fernández-González and Gómez-Rodríguez, 2022; Yang and Tu, 2022b). Therefore, for comprehensive comparisons, we also conduct experiments with gold POS tags, by adding POS tag embeddings element-wisely to the hidden representations from the encoder.

6.4. Versus the Pipeline Method

As discussed in Section 3, we recognize that our model may result in labeling incompatibility. To verify whether labeling incompatibility affects parsing performance, we conduct experiments on the PTB/CTB using the pipeline method, first applying constituent parsing and then obtaining d-trees via const-to-dep conversion. Please note that the conversion processes utilized for PTB/CTB do not require function tags and thus we can directly apply them to the parsed c-trees.

Table 6 shows that the performances of the pipeline method are inferior to that of Joint2o, especially on the CTB. We believe this is probably because such a conversion process suffers from error propagation, although compatibility is guaranteed.

Notably, when evaluated on the relatively high-performing PTB, the pipeline method exhibits a smaller performance drop on LAS (0.27) compared to UAS (0.67). This suggests that enforcing the labeling compatibility may lead to a better label prediction and therefore become a promising extension. While it seems impossible to extend strict labeling compatibility in our model, we can try to achieve approximate compatibility. By simplifying the label determination rules to be similar to those for finding head words, we can count such rules in the training set and adopt them to constrain the label prediction. We leave such extension of compatibility as future work.

6.5. Speed Comparison

Table 7 compares different parsing models in terms of parsing speed, including HPSG from Zhou and

	French		Hebrew		Korean		Polish		Swedish		*Average*	
	LAS	F1	LAS	F1	LAS	F1	LAS	F1	LAS	F1	LAS	F1
Kitaev et al. (2019)†	–	87.42	–	92.99	–	88.80	–	96.36	–	88.86	–	90.89
Nguyen et al. (2020)†	–	86.69	–	**93.67**	–	88.71	–	96.14	–	89.10	–	90.86
Yang and Tu (2023)†	–	**87.89**	–	–	–	89.31	–	96.18	–	–	–	–
Strzyz et al. (2019)	83.85	81.33	74.94	91.83	85.93	83.39	85.86	93.36	79.77	86.53	82.07	87.29
Li et al. (2022)	88.53	87.27	85.21	93.17	**90.73**	89.53	**91.37**	**96.55**	87.69	89.52	87.60	90.56
MTL	89.42	87.58	85.77	93.09	89.93	**89.46**	90.83	96.33	88.11	**89.73**	88.81	**91.24**
Joint2o	**89.52**	87.48	**85.83**	92.96	89.83	89.21	91.15	96.37	**88.38**	89.69	**88.94**	91.14

Table 5: Comparison with previous results on SPMRL-test. † indicates using only the constituency or dependency treebank. Note that we only compare with the results of the HPSG parser in Li et al. (2022).

	PTB		CTB	
	UAS	LAS	UAS	LAS
Pipeline	96.50	95.37	92.72	92.14
Joint2o	**97.17**	**95.64**	**93.36**	**92.97**

Table 6: Comparison with pipeline method on PTB/CTB-test.

Model	Sents/sec
HPSG	122.01
MTL w/ CKY+Eisner	299.31
MTL w/ Eisner-Satta	311.80
Joint2o w/ Eisner-Satta2o	305.79

Table 7: Speed comparison on PTB-test.

Zhao (2019) with $O(n^5)$ parsing algorithm in Cython implementation, MTL with batchified CKY and Eisner from Zhang et al. (2020b,a), and MTL/Joint2o with our batchified Eisner-Satta and its 2nd-order extension respectively. Our models are run on a machine with Intel Xeon Gold 6248R CPU and NVIDIA A100 40GB GPU. We set the batch size to 100 sentences and report the average time over ten runs.

We can observe that models with batchified algorithms are roughly 2.5× faster than HPSG19. Also, the speeds of "CKY+Eisner" and "Eisner-Satta(2o)" have similar speeds, probably because their time complexity after batchifying are all $O(n)$ on GPUs.

6.6. Analysis

In the previous experiments, we only used attachment scores and F1 scores to evaluate the performance of parsing, which assess the accuracy of dependencies and constituents respectively. However, upon analyzing the results in Table 2, we observe that the differences between the models are relatively small, especially for constituency parsing. For more detailed analyses, we evaluate the parsing results from other perspectives.

	LCM_{con}	LCM_{dep}	$\text{LCM}_{con+dep}$
		PTB	
SEP	$55.25_{\pm 0.6}$	$54.66_{\pm 0.5}$	$43.27_{\pm 0.6}$
MTL	$\mathbf{55.45}_{\pm 0.6}$	$55.74_{\pm 0.4}$	$46.74_{\pm 0.6}$
Joint1o	$55.02_{\pm 0.4}$	$55.84_{\pm 0.4}$	$46.88_{\pm 0.5}$
Joint2o	$55.37_{\pm 0.5}$	$\mathbf{55.95}_{\pm 0.5}$	$\mathbf{46.97}_{\pm 0.5}$
		CTB	
SEP	$29.37_{\pm 0.4}$	$41.53_{\pm 0.4}$	$26.03_{\pm 0.5}$
MTL	$\mathbf{29.39}_{\pm 0.3}$	$43.08_{\pm 0.4}$	$27.99_{\pm 0.3}$
Joint1o	$29.16_{\pm 0.4}$	$43.14_{\pm 0.3}$	$28.25_{\pm 0.4}$
Joint2o	$29.11_{\pm 0.3}$	$\mathbf{43.36}_{\pm 0.4}$	$\mathbf{28.33}_{\pm 0.5}$

Table 8: Complete matching on PTB/CTB-test.

Complete match of syntactic trees. To facilitate a more evident comparison between different models, following Zhang et al. (2019), we adopt a more challenging metric known as the sentence-level labeled complete match (LCM). Specifically, we evaluated the LCM of the predicted c-trees, d-trees, and their combination, denoted as LCM_{con}/LCM_{dep}/$\text{LCM}_{con+dep}$ respectively. Please note that an instance where the $\text{LCM}_{con+dep}$ is true only if both the LCM_{con} and LCM_{dep} are true.

For LCM_{con} and LCM_{dep}, their trends are quite similar to the model study in Table 2. For $\text{LCM}_{con+dep}$, we can clearly see that all the joint models have a significant improvement compared to SEP (>3 on PTB), suggesting that joint parsing c- and d-trees can lead to stronger compatibility.

Among joint parsing models, Joint1o has a slight improvement in $\text{LCM}_{con+dep}$ compared to MTL, and there is a further gain on Joint2o. This trend on both PTB and CTB indicates that joint parsing further at the training phase and the second-order extension both help the compatibility.

Fine-grained analysis. Yang and Tu (2022b) shows that different tree modelings may influence the performance on different lengths of sequence, span, and dependency. To investigate this, we follow Yang and Tu (2022b) and plot LAS/F1-score

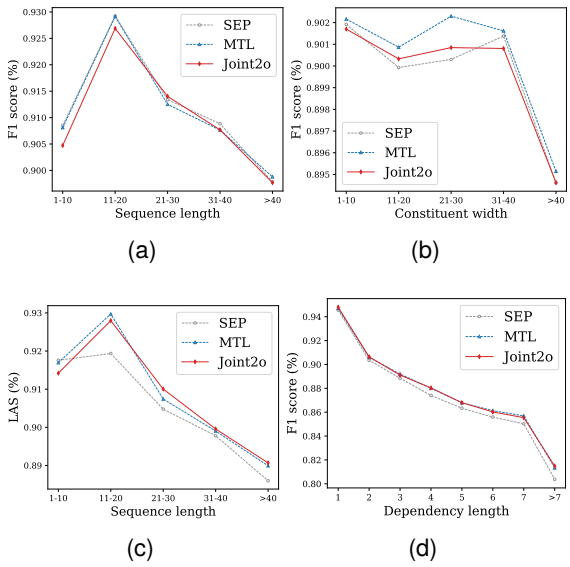


Figure 4: Fine-grained analysis on CTB-test.

as functions of the sequence lengths, constituent widths, and dependency lengths on CTB-test[7]. The width of a constituent from w_i to w_j is equal to $d := |i - j|$.

Figure 4(a) and 4(b) show the F1 scores for different sequence lengths and constituent widths. Notably, MTL and SEP perform similarly on sentences of different lengths, while Joint2o is slightly weaker on short sentences (<20). For constituent widths, MTL and Joint2o both have a small increase compared to SEP on short and medium constituents (<30), however, Joint2o is lower than the other two on long constituents (>30).

From Figure 4(c) and 4(d), we can see that MTL and Joint2o both have a steady improvement over SEP on almost all sentence lengths and dependency lengths. Meanwhile, Joint2o performs slightly better than MTL on longer sentences (>20).

7. Conclusions

This paper revisits the topic of joint c-tree and d-tree parsing. Compared with previous works, we further explore joint parsing at the training phase, thanks to the improved efficiency from the Eisner-Satta (Eisner and Satta, 1999) decoding algorithm. We also design second-order scoring components for promoting interaction between constituents and dependencies.

Our experiments encompass benchmark datasets in seven languages, yielding the following key findings.

1. The Eisner-Satta algorithm leads to about 2.5× speed-up, compared with the decoding algorithm proposed by Zhou and Zhao (2019).

2. Compared with separate modeling, joint parsing only at the inference phase, i.e., the MTL approach of Zhou and Zhao (2019), leads to modest performance boost on d-tree parsing, and has little impact on c-tree parsing performance.

3. Joint modeling at both training and inference phases does not further improve performance, compared with joint modeling only at the inference phase.

4. High-order joint modeling leads to modest performance improvement on both d-tree and c-tree parsing, compared with the first-order counterpart.

5. Detailed analysis shows that joint parsing significantly improves the complete matching ratio for the combination of c- and d-trees.

[7]We also conduct the same experiment on PTB-test and find a similar trend. However, all these models perform well on PTB and the margins between them are relatively small, resulting in fewer distinctions.

Acknowledgements

We thank all the anonymous reviewers for their valuable comments. We also thank Yu Zhang for his well-designed package Supar[8], from which we borrow the batchified version of the first-order Eisner-Satta algorithms in our work. This work was supported by National Natural Science Foundation of China (Grant No. 62176173 and 62336006), and a Project Funded by the Priority Academic Program Development (PAPD) of Jiangsu Higher Education Institutions.

Bibliographical References

Anne Abeillé. 2003. *Introduction. Treebanks: Building and Using Parsed Corpora (editor: Anne Abeillé)*. Kluwer Academic Publishers.

Michael Collins. 1999. Head-driven statistical models for natural language parsing. *PhD thesis, University of Pennsylvania*.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *CL*.

Benoit Crabbé. 2015. Multilingual discriminative lexicalized phrase structure parsing. In *Proceedings of EMNLP*.

[8]https://github.com/yzhangcs/parser/tree/main

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of ICLR*.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of ACL*.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2022. Multitask pointer network for multi-representational parsing. *Knowledge-Based Systems*.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of ACL*.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of ACL*.

Dan Klein and Christopher Manning. 2002. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems*.

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

Zuchao Li, Junru Zhou, Hai Zhao, and Kevin Parnow. 2022. Hpsg-inspired joint neural constituent and dependency parsing in o(n^3) time complexity. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*.

Chao Lou, Songlin Yang, and Kewei Tu. 2022. Nested named entity recognition as latent lexicalized constituency parsing. In *Proceedings of ACL*.

Alireza Mohammadshahi and James Henderson. 2021. Recursive non-autoregressive graph-to-graph transformer for dependency parsing with iterative refinement. *TACL*.

Thanh-Tung Nguyen, Xuan-Phi Nguyen, Shafiq Joty, and Xiaoli Li. 2020. Efficient constituency parsing by pointing. In *Proceedings of ACL*.

Fernando C. N. Pereira and David H. D. Warren. 1983. Parsing as deduction. In *Proceedings of ACL*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL*.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of ACL*.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*.

Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez. 2019. Sequence labeling parsing by learning across representations. In *Proceedings of ACL*.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL 2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177, Manchester, England. Coling 2008 Organizing Committee.

Ben Taskar, Dan Klein, Mike Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In *Proceedings of EMNLP*.

Songlin Yang and Kewei Tu. 2022a. Bottom-up constituency parsing and nested named entity recognition with pointer networks. In *Proceedings of ACL*.

Songlin Yang and Kewei Tu. 2022b. Headed-span-based projective dependency parsing. In *Proceedings of ACL*.

Songlin Yang and Kewei Tu. 2023. Don't parse, choose spans! continuous and discontinuous constituency parsing via autoregressive span selection. In *Proceedings of ACL*.

Yu Zhang, Zhenghua Li, and Zhang Min. 2020a. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of ACL*.

Yu Zhang, Houquan Zhou, and Zhenghua Li. 2020b. Fast and accurate neural CRF constituency parsing. In *Proceedings of IJCAI*.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of EMNLP*.

Zhisong Zhang, Xuezhe Ma, and Eduard Hovy. 2019. An empirical investigation of structured output modeling for graph-based neural dependency parsing. In *Proceedings of ACL*.

Junru Zhou and Hai Zhao. 2019. Head-Driven Phrase Structure Grammar parsing on Penn Treebank. In *Proceedings of ACL*.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of ACL*.

Language Resource References

Martha Palmer, Fu-Dong Chiou, Nianwen Xue, and Tsan-Kuang Lee. 2005. *The Penn Chinese Treebank*. TIDES, GALE Projects. Philadelphia: Linguistic Data Consortium, 5.1, ISLRN 426-628-131-806-1.

Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. 1999. *The Penn Treebank*. TIDES, GALE Projects. Philadelphia: Linguistic Data Consortium, 3.0, ISLRN 141-282-691-413-2.