Dependency Parsing with Partial Annotations: An Empirical Comparison

Yue Zhang[†], **Zhenghua Li**[†,*], **Jun Lang**[‡], **Qingrong Xia**[†], **Min Zhang**[†]
[†]Soochow University, Suzhou, China
{yzhang1107,qrxia}@stu.suda.edu.cn,{zhli13,minzhang}@suda.edu.cn
[‡]Alibaba Group, Hangzhou, China
langjun.lj@alibaba-inc.com

Abstract

This paper describes and compares two straightforward approaches for dependency parsing with partial annotations (PA). The first approach is based on a forest-based training objective for two CRF parsers, i.e., a biaffine neural network graph-based parser (Biaffine) and a traditional log-linear graph-based parser (LLGPar). The second approach is based on the idea of constrained decoding for three parsers, i.e., a traditional linear graph-based parser (LGPar), a globally normalized neural network transition-based parser (GN3Par) and a traditional linear transition-based parser (LTPar). For the test phase, constrained decoding is also used for completing partial trees. We conduct experiments on Penn Treebank under three different settings for simulating PA, i.e., random, most uncertain, and divergent outputs from the five parsers. The results show that LLGPar is most effective in directly learning from PA, and other parsers can achieve best performance when PAs are completed into full trees by LLGPar.

1 Introduction

Traditional supervised approaches for structural classification assume full annotation (FA), meaning that the training instances have complete manually-labeled structures. In the case of dependency parsing, FA means a complete parse tree is provided for each training sentence. However, recent studies suggest that it is more economic and effective to construct labeled data with partial annotation (PA). A lot of research effort has been attracted to obtain partially-labeled data for different tasks via active learning (Sassano and Kurohashi, 2010; Mirroshandel and Nasr, 2011; Li et al., 2012; Marcheggiani and Artières, 2014; Flannery and Mori, 2015; Li et al., 2016), cross-lingual syntax projection (Spreyer and Kuhn, 2009; Ganchev et al., 2009; Jiang et al., 2010; Li et al., 2014), or mining natural annotation implicitly encoded in web pages (Jiang et al., 2013; Liu et al., 2014; Nivre et al., 2014; Yang and Vozila, 2014). Figure 1) gives an example sentence partially annotated with two dependencies. However, there still lacks systematic study on how to build dependency parsers with PA. Most previous works listed above rely on ad-hoc strategies designed for only basic dependency parsers. One exception is that Li et al. (2014) convert partial trees into forests and train a traditional log-linear graph-based dependency parser (LLGPar) with PA based on a forest-based objective, showing promising results. Meanwhile, it is still unclear how PAs can be used by other main-stream dependency parsers, such as the traditional linear graph-based parser (LGPar) and transition-based parser (LTPar), and the newly proposed biaffine neural network graph-based parser (Biaffine) (Dozat and Manning, 2017) and globally normalized neural network transition-based parser (GN3Par) (Andor et al., 2016).

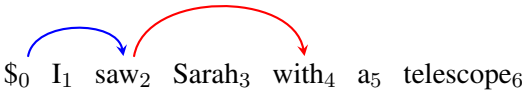

Figure 1: An example partial tree, where only the heads of "*saw*" and "*with*" are given.

This paper aims to thoroughly study this issue and make systematic comparison on different approaches for dependency parsing with PA. In sum-

[*]Correspondence author

Proceedings of the The 8th International Joint Conference on Natural Language Processing, pages 49–58,
Taipei, Taiwan, November 27 – December 1, 2017

mary, we make the following contributions.

- We present a general framework for directly training GN3Par, LGPar and LTPar with PA based on *constrained decoding*. The basic idea is to use the current feature weights to parse the sentence under the PA-constrained search space, and use the best parse as a pseudo gold-standard reference for feature weight update during perceptron training.
- We also implement the forest-objective based approach of Li et al. (2014) for the two CRF parsers, i.e., LLGPar and Biaffine.
- We have made thorough comparison among different *directly-train* approaches under three different settings for simulating PA, i.e., random dependencies, most uncertain dependencies, and dependencies with divergent outputs from the five parsers. We have also compared the proposed directly-train approaches with the straightforward *complete-then-train* approach.
- Extensive experiments lead to several interesting and clear findings.

2 Dependency Parsing

Given an input sentence $\mathbf{x} = w_0 w_1 ... w_n$, a dependency tree comprises a set of dependencies, namely $\mathbf{d} = \{i \curvearrowright j : 0 \leq i \leq n, 1 \leq j \leq n\}$, where $i \curvearrowright j$ is a dependency from a *head* word i to a *modifier* word j. A complete dependency tree contains n dependencies, namely $|\mathbf{d}| = n$, whereas a partial dependency tree contains less than n dependencies, namely $|\mathbf{d}| < n$. Alternatively, FA can be understood as a special form of PA. For clarity, we denote a complete tree as $\mathbf{d}$ and a partial tree as $\mathbf{d}^p$.

The decoding procedure aims to find an optimal complete tree $\mathbf{d}^*$:

$$\mathbf{d}^* = \arg\max_{\mathbf{d} \in \mathcal{Y}(\mathbf{x})} Score(\mathbf{x}, \mathbf{d}; \mathbf{w}) \quad (1)$$

where $\mathcal{Y}(\mathbf{x})$ defines the search space containing all legal trees for $\mathbf{x}$ and $\mathbf{w}$ is the model parameters.

2.1 Graph-based Approach

The graph-based method factorizes the score of a dependency tree into those of small subtrees $\mathbf{p}$:

$$Score(\mathbf{x}, \mathbf{d}; \mathbf{w}) = \sum_{\mathbf{p} \subseteq \mathbf{d}} Score(\mathbf{x}, \mathbf{p}; \mathbf{w}) \quad (2)$$

Dynamic programming based *exact search* are usually applied to find the optimal tree (McDonald et al., 2005; McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010).

Biaffine belongs to the first-order model and only incorporates scores of single dependencies. In contrast, for LLGPar and LGPar, we follow Li et al. (2014) and adopt the second-order model of McDonald and Pereira (2006) considering scores of single dependencies and adjacent siblings. Biaffine and LLGPar both belong to CRF parser. Please note that the original Biaffine is locally trained on each word. In this work, we follow Ma and Hovy (2017) and add a global CRF loss in the projective case, in order to directly use the proposed approach of Li et al. (2014). In other words, we extend the original Biaffine Parser described in Dozat and Manning (2017) by adding a CRF layer. Under the CRF model, the conditional probability of $\mathbf{d}$ given $\mathbf{x}$ is:

$$p(\mathbf{d}|\mathbf{x}; \mathbf{w}) = \frac{e^{Score(\mathbf{x}, \mathbf{d}; \mathbf{w})}}{\sum_{\mathbf{d}' \in \mathcal{Y}(\mathbf{x})} e^{Score(\mathbf{x}, \mathbf{d}'; \mathbf{w})}} \quad (3)$$

For training, $\mathbf{w}$ is optimized using gradient descent to maximize the likelihood of the training data.

Biaffine uses a neural network to compute the score of each dependency. First, the input word and POS tag sequence are fully encoded with two BiLSTM layers. Then, two MLPs are applied to each word position i to obtain two word representations, i.e., $\mathbf{r}_i^h$ (w_i as head) $\mathbf{r}_i^m$ (w_i as modifier). Finally, a biaffine classifier predicts the score of an arbitary dependency $i \curvearrowright j$.

$$score(i \curvearrowright j) = \mathbf{r}_i^h \cdot W \cdot \mathbf{r}_j^m + \mathbf{r}_i^h \cdot V \quad (4)$$

where W (matrix) and V (vector) are the biaffine parameters.

LLGPar is a traditional discrete feature based model, which defines the score of a tree as

$$Score(\mathbf{x}, \mathbf{d}; \mathbf{w}) = \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, \mathbf{d}) \quad (5)$$

$\mathbf{f}(\mathbf{x}, \mathbf{d})$ is a sparse accumulated feature vector corresponding to $\mathbf{d}$.

LGPar uses perceptron-like online training to directly learn $\mathbf{w}$. The workflow is similar to Algorithm 1, except that the gold-standard reference $\mathbf{d}^+$ is directly provided in the training data without the need of constrained decoding in line 7.

Algorithm 1 Perceptron training based on constrained decoding.

1: **Input:** Partially labeled data $\mathcal{D} = \{(\mathbf{x}_j, \mathbf{d}_j^p)\}_{j=1}^N$; **Output:** $\mathbf{w}$;
2: **Initialization:** $\mathbf{w}^{(0)} = \mathbf{0}$, $k = 0$
3: **for** $i = 1$ **to** I **do** // *iterations*
4: **for** $(\mathbf{x}_j, \mathbf{d}_j^p) \in \mathcal{D}$ **do** // *traverse*
5: $\mathbf{d}^- = \arg\max_{\mathbf{d} \in \mathcal{Y}(\mathbf{x}_j)} Score(\mathbf{x}_j, \mathbf{d}; \mathbf{w})$ // *Unconstrained decoding: LGPar*
6: $\mathbf{a}^- = \arg\max_{\mathbf{a} \to \mathbf{d} \in \mathcal{Y}(\mathbf{x}_j)} Score(\mathbf{x}_j, \mathbf{a} \to \mathbf{d}; \mathbf{w})$ // *Unconstrained decoding: LTPar*
7: $\mathbf{d}^+ = \arg\max_{\mathbf{d} \in \mathcal{Y}(\mathbf{x}_j, \mathbf{d}_j^p)} Score(\mathbf{x}_j, \mathbf{d}; \mathbf{w})$ // *Constrained decoding: LGPar*
8: $\mathbf{a}^+ = \arg\max_{\mathbf{a} \to \mathbf{d} \in \mathcal{Y}(\mathbf{x}_j, \mathbf{d}_j^p)} Score(\mathbf{x}_j, \mathbf{a} \to \mathbf{d}; \mathbf{w})$ // *Constrained decoding: LTPar*
9: $\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{f}(\mathbf{x}, \mathbf{d}^+) - \mathbf{f}(\mathbf{x}, \mathbf{d}^-)$ // *Update: LGPar*
10: $\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{f}(\mathbf{x}, \mathbf{a}^+) - \mathbf{f}(\mathbf{x}, \mathbf{a}^-)$ // *Update: LTPar*
11: $k = k + 1$
12: **end for**
13: **end for**

2.2 Transition-based Approach

The transition-based method builds a dependency by applying sequence of shift/reduce actions $\mathbf{a}$, and factorizes the score of a tree into the sum of scores of each action in $\mathbf{a}$ (Yamada and Matsumoto, 2003; Nivre, 2003; Zhang and Nivre, 2011):

$$\begin{aligned} Score(\mathbf{x}, \mathbf{d}; \mathbf{w}) &= Score(\mathbf{x}, \mathbf{a} \to \mathbf{d}; \mathbf{w}) \\ &= \sum\nolimits_{i=1}^{|\mathbf{a}|} Score(\mathbf{x}, c_i, a_i; \mathbf{w}) \end{aligned} \quad (6)$$

where $\mathbf{a}_i$ is the action taken at step i and c_i is the configuration status after taking action $a_1 ... a_{i-1}$. Transition-based methods use inexact beam search to find a highest-scoring action sequence.

GN3Par uses a neural network to predict scores of different actions given a state (Chen and Manning, 2014; Andor et al., 2016). First, 48 atomic features are embeded and concatenated as the input layer. Then, two hidden layers are applied to get the scores of all feasible actions. Unlike the traditional perceptron-like training, which only considers the best action sequence in the beam and the gold-standard sequence, their idea of global normalization is to approximately compute the probabilities of all the sequences in the beam to obtain a global CRF-like loss.

LTPar is a traditional discrete feature based model like LLGPar and LGPar, and adopts global perceptron-like training to learn the feature weights $\mathbf{w}$. We build an arc-eager transition-based dependency parser and features described in Zhang and Nivre (2011).

3 Directly training parsers with PA

As described in Li et al. (2014), CRF parsers such as **LLGPar** and **Biaffine** can naturally learn from PA based on the idea of ambiguous labeling, which allows a sentence to have multiple parse trees (forest) as its gold-standard reference (Riezler et al., 2002; Dredze et al., 2009; Täckström et al., 2013). First, a partial tree $\mathbf{d}^p$ is converted into a forest by adding all possible dependencies pointing to remaining words without heads, with the constraint that a newly added dependency does not violate existing ones in $\mathbf{d}^p$. The forest can be formally defined as $\mathcal{F}(\mathbf{x}, \mathbf{d}^p) = \{\mathbf{d} : \mathbf{d} \in \mathcal{Y}(\mathbf{x}), \mathbf{d}^p \subseteq \mathbf{d}\}$, whose conditional probability is the sum of probabilities of all trees that it contains:

什么叫不冲突？

$$p(\mathbf{d}^p|\mathbf{x}; \mathbf{w}) = \sum_{\mathbf{d} \in \mathcal{F}(\mathbf{x}, \mathbf{d}^p)} p(\mathbf{d}|\mathbf{x}; \mathbf{w}) \quad (7)$$

Then, we can define a forest-based training objective function to maximize the likelihood of training data as described in Li et al. (2014).

LGPar can be extended to directly learn from PA based on the idea of constrained decoding, as shown in Algorithm 1, which has been previously applied to Chinese word segmentation with partially labeled sequences (Jiang et al., 2010). The idea is using the best tree $\mathbf{d}^+$ in the constrained search space $\mathcal{Y}(\mathbf{x}_j, \mathbf{d}_j^p)$ (line 7) as a pseudo gold-standard reference for weight update. In traditional perceptron training, $\mathbf{d}^+$ would be a complete parse tree provided in the training data. It is trivial to implement constrained decoding for graph-based parsers, and we only need to disable some illegal combination operations during dynamic programming.

LTPar can also directly learn from PA in a similar way, as shown in Algorithm 1. Constrained decoding is performed to find a pseudo gold-standard reference (line 8). It is more complicate to design constrained decoding for transition-based parsing

	train-1K	train-39K	dev	test
#Sentence	1,000	38,832	1,700	2,416
#Token	24,358	925,670	40,117	56,684

Table 1: Data Statistics. FA is always used for train-1K, whereas PA is simulated for train-39K.

than graph-based parsing. Fortunately, Nivre et al. (2014) propose a constrained decoding procedure for the arc-eager parsing system. We ignore the details due to the space limitation.

GN3Par learns from PA in a similar manner with LTPar. The difference is that for each sentence, all the sequences in beam are used as negative examples in Line 6, and $\mathbf{a}^+$ obtained in Line 8 as gold-standard. Then, the global loss is computed in the same way with the case of FA.[1] Meanwhile, since GN3Par uses the arc-standard transition system, we also develop a constrained decoding procedure for the arc-standard system, which will be released as supporting documents.

4 Experiments

Data. We conduct experiments on Penn Treebank (PTB), and follow the standard data splitting (sec 2-21 as training, sec 22 as development, and sec 23 as test). Original bracketed structures are converted into dependency structures using Penn2Malt with default head-finding rules. We build a CRF-based bigram part-of-speech (POS) tagger to produce automatic POS tags for all train/dev/test data (10-way jackknifing on training data), with tagging accuracy 97.3% on test data. As suggested by an earlier anonymous reviewer, we further split the training data into two parts. We assume that the first $1K$ training sentences are provided as a small-scale data with FA, which can be obtained by a small amount of manual annotation or through cross-lingual projection methods. We simulate PA for the remaining $39K$ sentences. Table 1 shows the data statistics.

Parameter settings. We implement all five parsers from scratch using C++, which will be released publicly in the future. For those who are immediately interested, please contact us. We train LLGPar with stochastic gradient descent (Finkel et al., 2008). For LTPar and GN3Par, the beam size is 64 and the standard early update is adopted during training (Collins, 2002). For LGPar and LTPar, averaged perceptron is adopted (Collins, 2002).

For Biaffine, we directly adopt most hyperparameters of the released code of Dozat and Manning (2017), only removing the components related with dependency labels, since we focus on unlabeled dependency parsing in this work. The LSTM (two forward plus two backward) layers all use 300-dimension hidden cells. Dropout with ratio of 0.75 is applied to most layers before output. The two MLPs both have 100-dimension outputs without hidden layer. Adam optimization is adopted with $\alpha_1 = \alpha_2 = 0.9$.

For GN3Par, we follow Daniel et al. (2016), and use two 1024×1024 hidden layers, and adopt momentum (ratio of 0.9) Adam optimization.

For both Biaffine and GN3Par, we set the embedding dimension of both word/tag to 100, and use the GloVe pretrained word embedding for initialization[2], and randomly initialize embeddings of POS tags.

Since we have two sets of training data, we adopt the simple corpus-weighting strategy of Li et al. (2014). In each iteration, we merge train-1K and a subset of random $10K$ sentences from train-39K, shuffle them, and then use them for training. For all parsers, training terminates when the peak parsing accuracy on dev data does not improve in 30 consecutive iterations.

For **evaluation metrics**, we use the standard unlabeled attachment score (UAS) excluding punctuation marks.

4.1 Three settings for simulating PA on train-39K

In order to simulate PA for each sentence in train-39K, we only keep α% gold-standard dependencies (not considering punctuation marks), and remove all other dependencies. We experiment with three simulation settings to fully investigate the capability of different approaches in learning from PA.

Random (30% or 15%):[3] For each sentence in train-39K, we randomly select α% words, and only keep dependencies linking to these words.

[1] We have also tried to use all sequences in the beam in Line 8 as gold-standard, instead of the best $\mathbf{a}^+$, considering that there may be many gold-standard references in the case of PA. However, the accuracies become lower.

[2] https://nlp.stanford.edu/projects/glove/

[3] We choose 15% since the parsers achieve about 85% UAS when trained on train-1K (see Table 4). Then 30% aim to find the effect of different levels of supervision.

	Biaffine	LLGPar	LGPar	GN3Par	LTPar	Berkeley	Turbo	Mate-tool	ZPar
on Dev	**94.37**	93.16	93.00	93.32	92.77	92.84	92.86	92.58	92.42
on Test	**94.18**	92.42	92.43	93.26	92.01	92.85	92.63	92.48	92.12

Table 2: UAS of different parsers trained on all training data ($40K$)

	FA(random)			PA(random)		PA(uncertain)		PA(divergence)
	100%	30%	15%	30%	15%	30%	15%	21.33%
Biaffine	**94.37**	**93.06** (-1.31)	**92.10** (-2.27)	**92.84** (-1.53)	**91.92** (-2.45)	**93.63** (-0.74)	**92.83** (-1.54)	**93.58** (-0.79)
LLGPar	93.16	91.93 **(-1.23)**	91.15 **(-2.01)**	92.39 **(-0.77)**	91.66 **(-1.50)**	93.02 **(-0.14)**	92.44 **(-0.72)**	92.83 **(-0.33)**
LGPar	93.00	91.76 (-1.24)	90.80 (-2.20)	91.63 (-1.37)	90.62 (-2.38)	92.46 (-0.54)	91.64 (-1.36)	92.42 (-0.58)
GN3Par	93.32	91.99 (-1.33)	91.17 (-2.15)	91.43 *(-1.89)*	90.34 *(-2.98)*	92.40 (-0.92)	91.80 (-1.52)	92.60 (-0.72)
LTPar	*92.77*	*91.22 (-1.55)*	*90.35 (-2.42)*	*91.12* (-1.65)	*90.12* (-2.65)	*91.35 (-1.42)*	*90.99 (-1.78)*	*91.04 (-1.73)*

Table 3: UAS on dev data: parsers are directly trained on train-$1K$ with FA and train-$39K$ with PA. "FA (random) $\alpha\%$" means randomly selecting $\alpha\%$ sentences with FA from train-$39K$ for training. Numbers in parenthesis are the accuracy gap from the second column "FA (100%)".

With this setting, we aim to purely study the issue without biasing to certain structures. This setting may be best fit the scenario automatic syntax projection based on bitext, where the projected dependencies tend to be arbitrary (and noisy) due to the errors in automatic source-language parses and word alignments and non-isomorphism syntax between languages.

Uncertain (30% or 15%): In their work of active learning with PA, Li et al. (2016) show that the marginal probabilities from LLGPar is the most effective uncertainty measurement for selecting the most informative words to be annotated. Following their work, we first train LLGPar on train-$1K$ with FA, and then use LLGPar to parse train-$39K$ and select $\alpha\%$ most uncertain words. We use the gold-standard heads of the selected words as PAs for model training.

Following Li et al. (2016), we measure the uncertainty of a word w_i according to the *marginal probability gap* between its two most likely heads h_i^0 and h_i^1.

$$\mathit{Uncertainty}(\mathbf{x}, i) = p(h_i^0 \curvearrowright i|\mathbf{x}) - p(h_i^1 \curvearrowright i|\mathbf{x}) \quad (8)$$

This setting fits the scenario of active learning, which aims to save annotation effort by only annotating the most useful structures.

Divergence (21.33%): We train all five parsers on train-$1K$, and use them to parse train-$39K$. If their output trees do not assign the same head to a word, then we keep the gold-standard dependency pointing to the word, leading to 21.33% remaining dependencies. This setting fits to the tri-training scenario investigated in Li et al. (2014).

4.2 Results of different parsers trained on FA

We train the five parsers on all the training data with FA. We also employ four publicly available parsers with their default settings. BerkeleyParser (v1.7) is a constituent-structure parser, whose results are converted into dependency structures (Petrov and Klein, 2007). TurboParser (v2.1.0) is a linear graph-based dependency parser using linear programming for inference (Martins et al., 2013). Mate-tool (v3.3) is a linear graph-based dependency parser very similar to our implemented LGPar (Bohnet, 2010). ZPar (v0.6) is a linear transition-based dependency parser very similar to our implemented LTPar (Zhang and Clark, 2011). The results are shown in Table 2.

We can see that the five parsers that we adopt achieve competitive parsing accuracy and serve as strong baselines. Especially, the recently proposed neural network Biaffine outperforms other parser by more than 1%.

4.3 Results of the directly-train approaches

The five parsers are directly trained on train-$1K$ with FA and train-$39K$ with PA based on the methods described in Section 3. Table 3 shows the results.

Comparing the five parsers, we have several clear findings. (1) LLGPar is the most effective in directly learning from PA since its accuracy drop is the smallest over all PA settings compared with FA (100%). (2) Although Biaffine achieves best

Parser for completion	No constraints	PA (random)		PA (uncertain)		PA (divergence)
	0%	30%	15%	30%	15%	21.33%
Biaffine-1K	**87.08**	92.10 (+5.02)	89.79 (+2.71)	96.78 (+9.70)	93.47 (+6.39)	96.76 (+9.68)
LLGPar-1K	86.67	**92.65** (+5.98)	**90.02** (+3.35)	**97.43** (+10.76)	**94.43** (+7.76)	**97.07** (+10.40)
LGPar-1K	86.05	92.16 (+6.11)	89.48 (+3.43)	97.30 (+11.25)	94.11 **(+8.06)**	96.99 (+10.94)
GN3Par-1K	85.86	92.34 **(+6.48)**	89.54 **(+3.68)**	97.02 (+11.16)	93.69 (+7.83)	96.56 (+10.70)
LTPar-1K	85.38	91.76 (+6.38)	88.89 (+3.51)	96.90 **(+11.52)**	93.35 (+7.97)	96.72 **(+11.34)**
LLGPar-1K+39K	–	95.55	93.37	**98.30**	**96.22**	97.69
Biaffine-1K+39K	–	**95.77**	**93.52**	98.27	96.17	**97.73**

Table 4: UAS of full trees in train-39K completed via constrained decoding.

accuracy over all settings, thanks to its strong performance under the basic FA setting, we find that the accuracy gap between LLGPar and Biaffine becomes much smaller with PA than with FA. This also indicates that LLGPar is more effective in directly learning from PA. (3) LTPar achieves the lowest accuracy over all settings, especially on PA under uncertain (30%, 15%) and divergence. It is also clear that the accuracy declines the largest on these three settings, compared with FA (100%).

FA (random) vs. PA (random):[4] from the results in the two major columns, we can see that LLGPar achieves higher accuracy by about 0.5% when trained on sentences with $\alpha\%$ random dependencies than when trained on $\alpha\%$ random sentences with FA. This is reasonable and can be explained under the assumption that LLGPar can make full use of PA in model training. In fact, in both cases, the training data contains approximately the same number of annotated dependencies. However, from the perspective of model training, given some dependencies in the case of PA, more information about the syntactic structure can be derived.[5]

Taking Figure 1 as an example, "I_1" can only modify "saw_2" due to the single-root and single-head constraints; similarly, "$Sarah_3$" can only modify either "saw_2" or "$with_2$"; and so on. Moreover, since LLGPar is a second-order model, the presence of certain dependencies can directly affect the choice of other dependencies through the scores of adjacent siblings. Therefore, given the same amount of annotated dependencies, random PA contains more syntactic information than random FA, which explains why LLGPar performs better with PA than FA.

In contrast, all other four parsers achieve lower accuracy with PA than with FA. Biaffine differs from LLGPar in being a first-order model, and thus cannot fully utilize PA by considering sibling scores. The problem of LGPar may lie in the perceptron training with constrained decoding, which only considers a single best tree that complies with the given PA as gold-standard (Line 7 in Algorithm 1), unlike the forest-based objective of LLGPar that consider all trees weighted with probabilities. Both GN3Par and LTPar suffer from the inexact search problem. In other words, the approximate beam search can cause the correct tree drops off the beam too soon due to lower scores for earlier actions, and thus return a bad $\mathbf{a}^+$ that causes the model be updated to bias to wrong structures (Line 8 in Algorithm 1).

PA (random) vs. PA (uncertain):[6] we can see that all five parsers achieve much higher accuracy in the latter case.[7] The annotated dependencies in PA (uncertain) are most uncertain ones for current statistical parser (i.e., LLGPar), and thus are more helpful for training the models than those in PA (random). Another phenomenon is that, in the case of PA (uncertain), increasing $\alpha\% = 15\%$ to

[4] These two settings should give the clearest evidence whether a parser can effectively learn from PAs. Under the same $\alpha\%$, although containing approximately the same number of dependencies, PA certainly provide more syntactic information than FA, since 1) it is more expensive to annotate PA than FA in the terms of annotation time per dependency; 2) in PA, partially annotated dependencies can provide strong constraints on the remaining undecided dependencies. Therefore, we assume that a parser is effectively in learning from PA if it can achieve at least higher accuracy under PA.

[5] Also, as suggested in the work of Li et al. (2016), annotating PA is more time-consuming than annotating FA in terms of averaged time for each dependency, since dependencies in the same sentence are correlated and earlier annotated dependencies usually make later annotation easier.

[6] From the idea of active learning, we know that annotating the most informative dependencies as more training data can help models best. So, we select the most uncertain dependencies and compare the result on the setting with randomly-selected dependencies.

[7] The only exception is LTPar with 30% PA, the accuracy increases by only $91.35 - 91.12 = 0.23\%$, which may be caused by the ineffectiveness of LTPar in learning from PA.

30% actually doubles the number of annotated dependencies, but only boost accuracy of LLGPar by $93.02 - 92.44 = 0.58\%$, which indicates that newly added 15% dependencies are much less useful since the model can already well handle these low-uncertainty dependencies.

PA (uncertain, 30%) vs. PA (divergence):[8] we can see that the all five parsers achieve similar parsing accuracies under the two settings. This indicates that the divergence strategy can find very useful dependencies for all parsers, whereas uncertainty measurement based on LLGPar might be biased towards itself to a certain extent.

In summary, we can conclude from the results that *LLGPar is the most effective in directly learning from PA among all five parsers, due to both the second-order modeling and the forest-based training objective.*

4.4 Results of the complete-then-train methods

The most straight-forward method for learning from PA is the complete-then-learn method (Mirroshandel and Nasr, 2011). The idea is first using an existing parser to complete partial trees in train-39K into full trees based on constrained decoding, and then training the target parser on train-1K with FA and train-39K with completed FA.

Results of completing via constrained decoding: Table 4 reports UAS of the completed trees on train-39K using two different strategies for completion. "No constraints (0%)" means that train-39K has no annotated dependencies and normal decoding without constraints is used. In the remaining columns, each parser performs constrained decoding on PA where $\alpha\%$ dependencies are provided in each sentence.

- **Coarsely-trained-self for completion**: We complete PA into FA using corresponding parsers coarsely trained on only train-1K with FA. We call these parsers *Biaffine*-1K, *LLGPar*-1K, *LGPar*-1K, *GN3Par*-1K, *LTPar*-1K respectively.

- **Fine-trained-LLGPar for completion**: We complete PA into FA using LLGPar fine trained on both train-1K with FA and train-39K with PA. We call this LLGPar as *LLGPar*-1K+39K. Please note that *LLGPar*-1K+39K actually performs *closed test* in this setting, meaning that it parses its training data. For example, *LLGPar*-1K+39K trained on random (30%) is employed to complete the same data by filling the remaining 70% dependencies.

- **Fine-trained-Biaffine for completion**: This is the same with the case of "Fine-trained-LLGPar", except that we replace LLGParser with Biaffine. We call the resulting parser as *Biaffine*-1K+39K.

Comparing the five parsers trained on train-1K, we can see that constrained decoding has similar effects on all five parsers, and is able to return much more accurate trees. Numbers in parenthesis show the accuracy gap between unconstrained 0% and constrained decoding. This suggests that constrained decoding itself is not responsible for the ineffectiveness of Algorithm 1 for other parsers, especially LTPar.

Comparing the results of LLGPar-1K and LLGPar-1K+39K, it is obvious that the latter produces much better full trees since the fine-trained LLGPar can make extra use of PA in train-39K during training.

LLGPar-1K+39K and Biaffine-1K+39K achieve similar accuracies. We choose to use the former for completion since LLGPar is the most effective in both learning from PA and completing PA, as indicated by the results in Table 3 and 4.

Results of training on completed FA: Table 5 compares performance of the five parsers trained on train-1K with FA and train-39K with completed FA, from which we can draw several clear and interesting findings. First, different from the case of directly training on PA, the accuracy gaps among the five parsers become much more stable when trained on data with completed FA in both completion settings. Second, *using parsers coarsely-trained on train-1K for completion leads to very bad performance*, which is even much worse than those of the directly-train method in Table 3 except for LTPar with uncertain (30%) and divergence. Third, *using the fine-trained LLGPar-1K+39K for completion makes LGPar and LTPar achieve nearly the same accuracies with LLGPar*, which may be because LLGPar provides complementary effects during completion, analogous to the scenario of co-training.

[8] Selecting uncertain dependencies according to LLGPar may cause the resulting data to be biased to LLGPar. Therefore, we consider the divergence among all parsers for selection.

	Completed by self-1K					Completed by LLGPar-1K+39K				
	PA (random)		PA (uncertain)		PA (divergence)	PA (random)		PA (uncertain)		PA (divergence)
	30%	15%	30%	15%	21.33%	30%	15%	30%	15%	21.33%
Biaffine	**90.88**	**89.77**	**92.91**	**91.55**	**92.83**	**93.13**	**92.46**	**93.52**	**93.02**	**93.48**
LLGPar	89.91	88.69	92.05	90.77	92.28	92.29	91.54	92.86	92.33	92.76
LGPar	89.42	88.32	91.85	90.66	92.07	92.17	91.59	92.84	92.21	92.79
GN3Par	89.77	88.38	92.07	90.71	92.07	92.43	91.83	92.82	92.45	92.66
LTPar	89.17	87.72	91.59	90.12	91.67	92.05	91.37	92.42	92.10	92.40

Table 5: UAS on dev data: parsers trained on train-1K with FA and train-39K with completed FA.

	Directly train on train-39K with PA					Train-39K with FA completed by LLGPar-1K+39K				
	PA (random)		PA (uncertain)		PA (divergence)	PA (random)		PA (uncertain)		PA (divergence)
	30%	15%	30%	15%	21.33%	30%	15%	30%	15%	21.33%
Biaffine	**92.76**	**91.66**	**93.44**	**92.82**	**93.43**	**92.82**	**92.00**	**93.20**	**92.88**	**93.30**
LLGPar	91.73	91.02	92.34	91.83	92.34	91.46	90.99	92.20	91.59	92.18
LGPar	91.17	90.36	91.99	91.28	91.74	91.55	90.96	91.98	91.57	92.01
GN3Par	91.15	89.86	92.12	91.91	92.50	92.12	91.44	92.65	92.27	92.56
LTPar	90.79	89.89	90.47	90.37	90.75	91.48	90.78	91.80	91.45	91.87

Table 6: UAS on test data: comparison of the directly-train and complete-then-train methods.

4.5 Results on test data: directly-train vs. complete-then-train

Table 6 reports UAS on the test data of parsers directly trained on train-1K with FA and train-39K with PA, and of those trained on train-1K with FA and train-39K with FA completed by fine-trained LLGPar-1K+39K. The results are consistent with the those on dev data in Table 3 and 5. Comparing the two settings, we can draw two interesting findings. First, LLGPar performs slightly better with the directly-train method. Second, the two settings lead to very similar performance on Biaffine, without a clear trend. Third, LGPar performs slightly better with the complete-then-train method in most cases except for uncertain (30%). Four, GN3Par and LTPar perform much better with the complete-then-train method.

5 Related work

In parsing community, most previous works adopt ad-hoc methods to learn from PA. Sassano and Kurohashi (2010), Jiang et al. (2010), and Flannery and Mori (2015) convert partially annotated instances into local dependency/non-dependency classification instances, which may suffer from the lack of non-local correlation between dependencies in a tree.

Mirroshandel and Nasr (2011) and Majidi and Crane (2013) adopt the complete-then-learn method. They use parsers coarsely trained on existing data with FA for completion via constrained decoding. However, our experiments show that this leads to dramatic decrease in parsing accuracy.

Nivre et al. (2014) present a constrained decoding procedure for arc-eager transition-based parsers. However, their work focuses on allowing their parser to effectively exploit external constraints during the evaluation phase. In this work, we directly employ their method and show that constrained decoding is effective for LTPar and thus irresponsible for its ineffectiveness in learning PA.

Directly learning from PA based on constrained decoding is previously proposed by Jiang et al. (2013) for Chinese word segmentation, which is treated as a character-level sequence labeling problem. In this work, we first apply the idea to LGPar and LTPar for directly learning from PA.

Directly learning from PA based on a forest-based objective in LLGPar is first proposed by Li et al. (2014), inspired by the idea of *ambiguous labeling*. Similar ideas have been extensively explored recently in sequence labeling tasks (Liu et al., 2014; Yang and Vozila, 2014; Marcheggiani and Artières, 2014).

Hwa (1999) pioneers the idea of exploring PA for constituent grammar induction based on a variant Inside-Outside re-estimation algorithm (Pereira and Schabes, 1992). Clark and Curran (2006) propose to train a Combinatorial Categorial Grammar parser using partially labeled data

only containing predicate-argument dependencies. Mielens et al. (2015) propose to impute missing dependencies based on Gibbs sampling in order to enable traditional parsers to learn from partial trees.

6 Conclusions

This paper investigates the problem of dependency parsing with partially labeled data. Particularly, we focus on the realistic scenario where we have a small-scale training dataset with FA and a large-scale training dataset with PA. We experiment with three settings for simulating PA and compare several directly-train and complete-then-train approaches with five mainstream parsers, i.e., Biaffine, LLGPar, LGPar, GN3Par and LTPar.

Based on this work, we may draw the following conclusions.

- For the complete-then-train approach, using parsers coarsely trained on small-scale data with FA for completion leads to unsatisfactory results.
- LLGPar is the most effective in directly learning from PA due to both its second-order modeling and probabilistic forest-based training objective.
- All other four parsers are less effective in directly learning from PA, but can achieve their best performance with the complete-then-train approach where PAs are completed into FAs by LLGPar fine-trained on all FA+PA data.

However, as our reviewers kindly point out, more extensive experiments and systematic analysis are needed to really understand this interesting issue and provide stronger findings, which we leave for future work.

Acknowledgments

The authors would like to thank the anonymous reviewers for the helpful comments. We are greatly grateful to Jiayuan Chao for her earlier-stage experiments on this work, and to Wenliang Chen for the helpful discussions. This work was supported by National Natural Science Foundation of China (Grant No. 61525205, 61373095, 61502325).

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of ACL*, pages 2442–2452.

Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of COLING*, pages 89–97.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of EMNLP/CoNLL*, pages 141–150.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Conference on Empirical Methods in Natural Language Processing*, pages 740–750.

Stephen Clark and James Curran. 2006. Partial training for a lexicalized-grammar parser. In *Proceedings of the Human Language Technology Conference of the NAACL*, pages 144–151.

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

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

Mark Dredze, Partha Pratim Talukdar, and Koby Crammer. 2009. Sequence learning from data with multiple labels. In *ECML/PKDD Workshop on Learning from Multi-Label Data*.

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

Daniel Flannery and Shinsuke Mori. 2015. Combining active learning and partial annotation for domain adaptation of a japanese dependency parser. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 11–19.

Kuzman Ganchev, Jennifer Gillenwater, and Ben Taskar. 2009. Dependency grammar induction via bitext projection constraints. In *Proceedings of ACL-IJCNLP 2009*, pages 369–377.

Rebecca Hwa. 1999. Supervised grammar induction using training data with limited constituent information. In *Proceedings of ACL*, pages 73–79.

Wenbin Jiang, , and Qun Liu. 2010. Dependency parsing and projection based on word-pair classification. In *ACL*, pages 897–904.

Wenbin Jiang, Meng Sun, Yajuan Lü, Yating Yang, and Qun Liu. 2013. Discriminative learning with natural annotations: Word segmentation as a case study. In *Proceedings of ACL*, pages 761–769.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *ACL*, pages 1–11.

Shoushan Li, Guodong Zhou, and Chu-Ren Huang. 2012. Active learning for Chinese word segmentation. In *Proceedings of COLING 2012: Posters*, pages 683–692.

Zhenghua Li, Min Zhang, and Wenliang Chen. 2014. Soft cross-lingual syntax projection for dependency parsing. In *COLING*, pages 783–793.

Zhenghua Li, Min Zhang, Yue Zhang, Zhanyi Liu, Wenliang Chen, Hua Wu, and Haifeng Wang. 2016. Active learning for dependency parsing with partial annotation. In *ACL*.

Yijia Liu, Yue Zhang, Wanxiang Che, Ting Liu, and Fan Wu. 2014. Domain adaptation for CRF-based Chinese word segmentation using free annotations. In *Proceedings of EMNLP*, pages 864–874.

Xuezhe Ma and Eduard Hovy. 2017. Neural probabilistic model for non-projective mst parsing. In *arxiv:https://arxiv.org/abs/1701.00874*.

Saeed Majidi and Gregory Crane. 2013. Active learning for dependency parsing by a committee of parsers. In *Proceedings of IWPT*, pages 98–105.

Diego Marcheggiani and Thierry Artières. 2014. An experimental comparison of active learning strategies for partially labeled sequences. In *Proceedings of EMNLP*, pages 898–906.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of ACL*, pages 617–622.

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

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

Jason Mielens, Liang Sun, and Jason Baldridge. 2015. Parse imputation for dependency annotations. In *Proceedings of ACL-IJCNLP*, pages 1385–1394.

Seyed Abolghasem Mirroshandel and Alexis Nasr. 2011. Active learning for dependency parsing using partially annotated sentences. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 140–149.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of IWPT*, pages 149–160.

Joakim Nivre, Yoav Goldberg, and Ryan McDonald. 2014. Constrained arc-eager dependency parsing. In *Computational Linguistics*, volume 40, pages 249–258.

Fernando Pereira and Yves Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the Workshop on Speech and Natural Language (HLT)*, pages 122–127.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of NAACL*.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. III Maxwell, and Mark Johnson. 2002. Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of ACL*, pages 271–278.

Manabu Sassano and Sadao Kurohashi. 2010. Using smaller constituents rather than sentences in active learning for japanese dependency parsing. In *Proceedings of ACL*, pages 356–365.

Kathrin Spreyer and Jonas Kuhn. 2009. Data-driven dependency parsing of new languages using incomplete and noisy training data. In *CoNLL*, pages 12–20.

Oscar Täckström, Ryan McDonald, and Joakim Nivre. 2013. Target language adaptation of discriminative transfer parsers. In *Proceedings of NAACL*, pages 1061–1071.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, pages 195–206.

Fan Yang and Paul Vozila. 2014. Semi-supervised Chinese word segmentation using partial-label learning with conditional random fields. In *Proceedings of EMNLP*, pages 90–98.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of ACL*, pages 188–193.