Efficient Inner-to-outer Greedy Algorithm for Higher-order Labeled Dependency Parsing

Xuezhe Ma and **Eduard Hovy**
Language Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
xuezhem@cs.cmu.edu, ehovy@andrew.cmu.edu

Abstract

Many NLP systems use dependency parsers as critical components. Jonit learning parsers usually achieve better parsing accuracies than two-stage methods. However, classical joint parsing algorithms significantly increase computational complexity, which makes joint learning impractical. In this paper, we proposed an efficient dependency parsing algorithm that is capable of capturing multiple edge-label features, while maintaining low computational complexity. We evaluate our parser on 14 different languages. Our parser consistently obtains more accurate results than three baseline systems and three popular, off-the-shelf parsers.

1 Introduction

Natural language processing (NLP) systems, like machine translation (Xie et al., 2011), resource-low languages processing (McDonald et al., 2013; Ma and Xia, 2014), word sense disambiguation (Fauceglia et al., 2015) , and entity coreference resolution (Durrett and Klein, 2013), are becoming more sophisticated, in part because of utilizing syntacitc knowledges such as dependency parsing trees.

Dependency parsers predict dependency structures and dependency type labels on each edge. However, most graph-based dependency parsing algorithms only produce unlabeled dependency trees, particularly when higher-order factorizations are used (Koo and Collins, 2010; Ma and Zhao, 2012b; Martins et al., 2013; Ma and Zhao, 2012a). A two-stage method (McDonald, 2006) is often used because the complexity of some joint learning models is unacceptably high. On the other hand, joint learning models can benefit from edge-label information that has proven to be important to provide more accurate tree structures and labels (Nivre and Scholz, 2004).

Previous studies explored the trade-off between computational costs and parsing performance. Some work (McDonald, 2006; Carreras, 2007) simplified labeled information to only single label features. Other work (Johansson and Nugues, 2008; Bohnet, 2010) used richer label features but increased systems' complexities significantly, while achieving better parsing accuracy. Yet, there are no previous work addressing the problem of good balance between parsing accuracy and computational costs for joint parsing models.

In this paper, we propose a new dependency parsing algorithm that can utilize edge-label information of more than one edge, while simultaneously maintaining low computational complexity. The component needed to solve this dilemma is an inner-to-outer greedy approximation to avoid an exhaustive search. The contributions of this work are (i) showing the effectiveness of edge-label information on both UAS and LAS. (ii) proposing a joint learning parsing model which achieves both effectiveness and efficience. (iii) giving empirical evaluations of this parser on different treebanks over 14 languages.

2 Joint Parsing Algorithm

2.1 Basic Notations

In the following, x represents a generic input sentence, and y represents a generic dependency tree. Formally, for a sentence x, dependency parsing is the task of finding the dependency tree y with the highest-score for x:

$$y^*(x) = \underset{y \in \mathcal{Y}(x)}{\operatorname{argmax}} \operatorname{Score}(x, y). \tag{1}$$

Here $\mathcal{Y}(x)$ denotes the set of possible dependency trees for sentence x.

In this paper, we adopt the second-order sibling factorization (Eisner, 1996; McDonald and

Pereira, 2006), in which each sibling part consists of a tuple of indices (h, m, c) where (h, m) and (h, c) are a pair of adjacent edges to the same side of the head h. By adding labele information to this factorization, $\mathrm{Score}(x, y)$ can be rewritten as:

$$\begin{aligned}\mathrm{Score}(x, y) &= \sum_{(h,m,c,l_1,l_2)\in y} \mathrm{S_{sib}}(h, m, c, l_1, l_2) \\ &= \sum_{(h,m,c,l_1,l_2)\in y} \lambda^T \mathrm{f}(h, m, c, l_1, l_2)\end{aligned} \tag{2}$$

where $\mathrm{S_{sib}}(h, m, c, l_1, l_2)$ is the score function for the sibling part (h, m, c) with l_1 and l_2 being the labels of edge (h, m) and (h, c), respectively. **f** are feature functions and λ is the parameters of parsing model.

2.2 Exact Search Parsing

The unlabeled sibling parser introduces three types of dynamic-programming structures: *complete* spans $C_{(s,t)}$, which consist of the headword s and its descendents on one side with the endpoint t, *incomplete* spans $I_{(s,t)}$, which consist of the dependency (s, t) and the region between the head s and the modifier t, and *sibling* spans $S_{(s,t)}$, which represent the region between successive modifiers s and t of some head. To capture label information, we have to extend each incomplete span $I_{s,t}$ to $I_{s,t,l}$ to store the label of dependency edge from s to t. The reason is that there is an edge shared by two adjacent sibling parts (e.g. (h, m, c) and (h, c, c') share the edge (h, c)). So the incomplete span $I_{(s,t)}$ does not only depend on the label of dependency (s, t), but also the label of the dependency (s, r) for each split point r. The dynamic-programming procedure for new incomplete spans[1] are

$$I_{(s,t,l)} = \max_{s<r\le t} S_{(r,t)} + \max_{l'\in L} I_{(s,r,l')} + \mathrm{S_{sib}}(s, r, t, l', l) \tag{3}$$

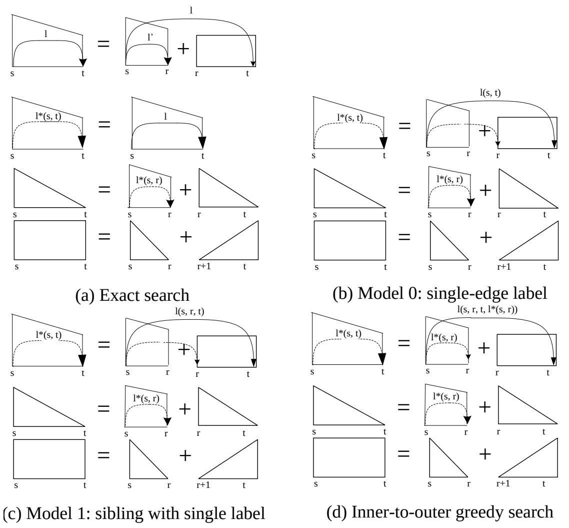
where L is set of all edge labels. Then we have,

$$I_{(s,t)} = \max_{l\in L} I_{(s,t,l)} \tag{4}$$

$$l^*_{(s,t)} = \operatorname*{argmax}_{l\in L} I_{(s,t,l)} \tag{5}$$

The graphical specification of the this parsing algorithm is provided in Figure 1 (a). The computational complexity of the exactly searching algorithm is $O(|L|^2n^3)$ time and $O(|L|n^2)$ space. In practice, $|L|$ is probably large. For English, the number of edge labels in Stanford Basic Dependencies (De Marneffe et al., 2006) is 45, and the number in the treebank of CoNLL-2008 shared task is 70. So it is impractical to perform an exhaustive search for parsing, and more efficient approximating algorithms are needed.

[1]Symmetric right-headed versions are elided for brevity



(a) Exact search (b) Model 0: single-edge label (c) Model 1: sibling with single label (d) Inner-to-outer greedy search

Figure 1: The dynamic-programming structures and derivation of four parsing algorithms. $I_{(s,t,l)}$ and $I_{(s,t)}$ are depicted as trapezoids with solid and dashed lines, respectively. $C_{(s,t)}$ are depicted as triangles and $S_{(s,t)}$ are depicted as boxes. Symmetric right-headed versions are elided for brevity.

2.3 Two Intermediate Models

In this section, we describe two intuitive simplifications of the labeled parsing model presented above. For the two simplified parsing models, efficient algorithms are available.

2.3.1 Model 0: Single-edge Label

In this parsing model, labeled features are restricted to a single edge. Specifically,

$$\mathrm{S_{sib}}(h, m, c, l_1, l_2) = \mathrm{S_{sib}}(h, c, l_2).$$

Then the dynamic-programming derivation for each incomplete span becomes

$$\begin{aligned}I_{(s,t)} &= \max_{s<r\le t}\left\{I_{(s,r)} + S_{(r,t)} + \max_{l\in L}\mathrm{S_{sib}}(s, t, l)\right\} \\ &= \max_{s<r\le t}\left\{I_{(s,r)} + S_{(r,t)} + \mathrm{S_{sib}}(s, t, l(s, t))\right\}\end{aligned}$$

where $l(s, t) = \operatorname*{argmax}_{l\in L} \mathrm{S_{sib}}(s, t, l)$.

In this case, therefore, we do not have to extend incomplete spans. The computational cost to calculate $l(s, t)$ is $O(|L|n^2)$ time, so the computational complexity of this algorithm is $O(n^3 + |L|n^2)$ time and $O(n^2)$ space.

2.3.2 Model 1: Sibling with Single Label

As remarked in McDonald (2006), Model 0 can be slightly enriched to include single label features associated with a sibling part. Formally,

$$\mathrm{S_{sib}}(h, m, c, l_1, l_2) = \mathrm{S_{sib}}(h, m, c, l_2).$$

Now, the dynamic-programming derivation is

$$I_{(s,t)} = \max_{s<r\leq t} \left\{ I_{(s,r)} + S_{(r,t)} + \mathrm{S_{sib}}(s, r, t, l(s, r, t)) \right\}$$

where $l(s, r, t) = \underset{l \in L}{\operatorname{argmax}} \mathrm{S_{sib}}(s, r, t, l)$.

The additional algorithm to calculate the best edge label $l(s, r, t)$ takes $O(|L|n^3)$ time. Therefore, this algorithm requires $O(|L|n^3)$ time and $O(n^2)$ space[2]. Figure 1 (b) and Figure 1 (c) provide the graphical specifications for Model 0 and Model 1, respectively.

2.4 Inner-to-outer Greedy Search

Though the two intermediate parsing models, model 0 and model 1, encode edge-label information and have efficient parsing algorithms, the labeled features they are able to capture are relatively limited due to restricting their labeled feature functions to a single label. Our experimental results show that utilizing these edge-label information yields a slight improvement of parsing accuracy (see Section 3 for details). In this section, we describe our new labeled parsing model that can exploit labeled features involving two edge-labels in a sibling part. To achieve efficient search, we adopt an method characterized by inferring labels of outer parts from the labels of inner ones.

Formally, consider the maximization problem in Eq 3. It can be treated as a two-layer maximization: first fixes a split point r and maximizes over all edge-label l, then maximizes over all possible split points. Our approach approximates the maximization in the first layer:

$$\begin{aligned} & \max_{l' \in L} I_{(s,r,l')} + \mathrm{S_{sib}}(s, r, t, l', l) \\ \approx \quad & I_{(s,r)} + \mathrm{S_{sib}}(s, r, t, l^*_{(s,r)}, l) \end{aligned} \tag{6}$$

Then the dynamic-programming derivation for each incomplete span is

$$I_{(s,t)} = \max_{s<r\leq t} I_{(s,r)} + S_{(r,t)} + \max_{l \in L} \mathrm{S_{sib}}(s, r, t, l^*_{(s,r)}, l) \tag{7}$$

To compute $I_{(s,t)}$, we need to calculate

$$l(s, r, t, l^*_{(s,r)}) = \underset{l \in L}{\operatorname{argmax}} \mathrm{S_{sib}}(s, r, t, l^*_{(s,r)}, l), \tag{8}$$

[2]We do not have to store $l(s, r, t)$, as each $l(s, r, t)$ will be calculated exactly once.

which is similar to the calculation of $l(s, r, t)$ in Model 1. The only difference between them is $l^*_{(s,r)}$ that can be calculated in previous derivations. Thus, their computation costs are almost the same.

The procedure of our algorithm to derivate incomplete spans can be regarded as two steps. At the first step, the algorithm goes through all possible split points (Eq 7). Then at the second step, at each split point r, it calculate the label $l(s, r, t, l^*_{(s,r)})$ (Eq 8) based on the sibling part (s, r, t) and the label $l^*_{(s,r)}$ which is the "best" label for dependency edge (s, r) based on incomplete span $I_{(s,r)}$. The key insight of this algorithm is the inner-to-outer dynamic-programming structure: inner modifiers (r) of a head (s) and their "best" labels $(l^*_{(s,r)})$ are generated before outer ones (t). Thus, using already computed "best" labels of inner dependency edges makes us get rid of maximizing over two labels, l' and l. Moreover, we do not have to extend each incomplete span by the augmentation with a "label" index. This makes the space complexity remains $O(n^2)$, which is important in practice. The graphical specification is provided in Figure 1 (d).

3 Experiments

3.1 Setup

We conduct our experiments on 14 languages, including the English treebank from CoNLL-2008 shared task (Surdeanu et al., 2008) and all 13 treebanks from CoNLL-2006 shared task (Buchholz and Marsi, 2006). We train our parser using The k-best version of the Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003; Crammer et al., 2006; McDonald, 2006). In our experiments, we set $k = 1$ and fix the number of iteration to 10, instead of tuning these parameters on development sets. Following previous work, all experiments are evaluated on the metrics of unlabeled attachment score (UAS) and Labeled attachment score (LAS), using the official scorer[3] of CoNLL-2006 shared task.

3.2 Non-Projective Parsing

The parsing algorithms described in the paper fall into the category of *projective* dependency parsers, which exclude crossing dependency edges. Since the treebanks from CoNLL shared tasks contain non-projective edges, we use the "mountain-

[3]`http://ilk.uvt.nl/conll/software.html`

	Two-stage		Model 0		Model 1		Our Model		Best in CoNLL		
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	System
ar	78.52	64.69	79.38	66.91	78.72	66.67	**79.60**	**67.09**	79.34	66.91	MD06
bg	91.98	86.75	92.34	87.47	92.12	87.41	**92.68**	**87.79**	92.04	87.57	MD06
zh	91.25	86.50	91.81	87.53	92.27	88.23	92.58	88.51	**93.18**	**89.96**	RD06
cs	87.32	76.56	87.36	78.42	87.72	78.90	**88.01**	79.31	87.30	**80.18**	MD06
da	90.96	84.63	91.24	85.47	91.50	85.79	**91.44**	**85.55**	90.58	84.79	MD06
nl	83.79	78.81	84.25	80.49	84.27	80.41	**84.45**	**80.31**	83.57	79.19	MD06
en	91.92	88.09	92.10	88.96	92.19	89.18	**92.45**	89.43	92.38	**90.13**	JN08
de	90.52	87.46	90.52	87.12	90.40	87.30	**90.79**	**87.74**	90.38	87.34	MD06
ja	93.14	90.95	93.32	91.29	93.52	91.80	**93.54**	**91.80**	93.10	91.65	NV06
pt	**91.60**	86.05	91.04	86.46	91.02	87.30	91.54	**87.68**	91.22	87.60	NV06
sl	83.03	70.80	83.23	72.88	83.93	73.38	**84.39**	**73.74**	83.17	73.44	MD06
es	85.61	80.95	86.05	82.83	86.42	**83.59**	**86.44**	83.29	86.05	82.25	MD06
sv	89.07	81.88	89.74	82.77	89.82	83.13	**89.94**	83.09	89.50	**84.58**	NV06
tr	75.02	57.78	**75.40**	60.25	74.75	59.73	75.32	60.39	**75.82**	**65.68**	NV06
av	87.41	80.14	87.70	81.35	87.76	81.63	**88.08**	81.84	87.69	**82.23**	– –

Table 1: UAS and LAS of non-projective versions of our parsing algorithms on 14 treebanks from CoNLL shared tasks, together with three baseline systems and the best systems for each language reported in CoNLL shared tasks. MD06 is McDonald et al. (2006), RD06 is Riedel et al. (2006), JN08 is Johansson and Nugues (2008), and NV06 is Nivre et al. (2006) Bold indicates the best result for a language. Red values represent statistically significant improvements over two-stage baseline system on the corresponding metrics with $p < 0.01$, using McNemar's test. Blue values indicate statistically significant improvements with $p < 0.05$.

climbing" non-projective parsing algorithm proposed in McDonald and Pereira (2006). This approximating algorithm first searches the highest scoring projective parse tree and then it rearranges edges in the tree until the rearrangements do not increase the score for the tree anymore [4].

3.3 Results and Comparison

Table 1 illustrates the parsing results our parser with non-projective parsing algorithm, together with three baseline systems—the two-stage system (McDonald, 2006) and the two intermediate models, Model 0 and Model 1—and the best systems reported in CoNLL shared tasks for each language. Our parser achieves better parsing performance on both UAS and LAS than all the three baseline systems for 12 languages. The two exceptions are Portuguese and Turkish, on which our parser achieves better LAS and comparable UAS.

Comparing with the best systems from CoNLL, our parser achieves better performance on both UAS and LAS for 9 languages. Moreover, the average UAS of our parser over the 14 languages is better than that of the best systems in CoNLL. It should be noted that the best results for 14 languages in CoNLL are not from one single system, but different systems that achieved best results for different languages. The system of McDonald et al. (2006) achieved the best average parsing performance over 13 languages (excluding English) in CoNLL-2006 shared tasks. Its average UAS and LAS are 87.03% and 80.83%, respectively, while our average UAS and LAS excluding English are 87.79% and 81.29%. So our parser shows significant improvement over the single best system reported in CoNLL-2006 shared task.

System	UAS	LAS
MaltParser	89.3	86.9
MSTParser	90.7	87.6
DNNParser	91.8	89.6
This Paper	**92.4**	**89.9**

Table 2: Parsing performance on PTB. The results for MaltParser, MSTParser and DNNParser are from table 5 of Chen and Manning (2014).

3.4 Experiments on PTB

To make a thorough empirical comparison with previous studies, we also evaluate our system on the English Penn Treebanks (Marcus et al., 1993) with Stanford Basic Dependencies (De Marneffe et al., 2006). We compare our parser with three off-the-shelf parsers: MaltParser (Nivre and Scholz, 2004; Zhang and Clark, 2008; Zhang and Nivre, 2011), MSTParser (McDonald et al., 2005), and the parser using Neural Networks

[4] Additional care is required in the non-projective approximation since a change of one edge could result in a label change for multiple edges

		F1 (UAS)			F1 (LAS)		
Label	Description	TST	Ours	diff	TST	Ours	diff
MNR	Adverbial of manner	54.01	66.10	12.10	52.23	62.15	9.92
OPRD	Predicative complement of raising/control verb	86.61	96.92	10.31	86.61	89.89	3.28
APPO	Apposition	77.40	82.62	5.22	72.74	77.03	4.29
ADV	General adverbial	73.33	77.65	4.32	69.86	73.14	3.28
AMOD	Modifier of adjective or adverbial	75.49	79.58	4.09	72.60	76.91	4.30
TMP	Temporal adverbial or nominal modifier	73.47	76.76	3.29	64.50	68.59	4.09
DIR	Adverbial of direction	62.42	65.02	2.60	62.42	64.61	2.19
LOC	Locative adverbial or nominal modifier	75.78	78.35	2.57	63.11	65.83	2.72
OBJ	Object	91.69	94.08	2.39	90.62	93.23	2.61

Table 3: Top 10 dependency labels on which our algorithm achieves most improvements on the F1 score of UAS, together with the corresponding improvements of LAS. “TST” indicates the two-stage system. The first column is the label name in the treebank. The second column is the label’s description from Surdeanu et al. (2008).

(DNNParser) (Chen and Manning, 2014). The results are listed in Table 2. Clearly, our parser is superior in terms of both UAS and LAS.

3.5 Analysis

To better understand the performance of our parser, we analyze the distribution of our parser’s UAS and LAS over different dependency labels on the English CoNLL treebank, compared with the ones of the two-stage model. Table 3 lists the top 10 dependency labels on which our algorithm achieves most improvements on the F1 score of UAS, together with the corresponding improvements of LAS.

From Table 3 we can see among the 10 labels, there are 5 labels — “MNR”, “ADV”, “TMP”, “DIR”, “LOC” — which are a specific kind of adverbials. This illustrates that our parser performs well on the recognition of different kinds of adverbials. Moreover, the label “OPRD” and “OBJ” indicate dependency relations between verbs and their modifiers, too. In addition, our parser also significantly improves the accuracy of appositional relations (“APPO”).

4 Conclusion

We proposed a new dependency parsing algorithm which can jointly learn dependency structures and edge labels. Our parser is able to use multiple edge-label features, while maintaining low computational complexity. Experimental results on 14 languages show that our parser significantly improves the accuracy of both dependency structures (UAS) and edge labels (LAS), over three baseline systems and three off-the-shelf parsers. This demonstrates that jointly learning dependency structures and edge labels can benefit both performance of tree structures and labeling accuracy. Moreover, our parser outperforms the best systems of different languages reported in CoNLL shared task for 9 languages.

In future, we are interested in extending our parser to higher-order factorization by increasing horizontal context (e.g., from siblings to “tri-siblings”) and vertical context (e.g., from siblings to “grand-siblings”) and validating its effectiveness via a wide range of NLP applications.

Acknowledgements

This research was supported in part by DARPA grant FA8750-12-2-0342 funded under the DEFT program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of Coling-2010*, pages 89–97, Beijing, China, August.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceeding of CoNLL-2006*, pages 149–164, New York, NY.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CONLL*, pages 957–961.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP-2014*, pages 740–750, Doha, Qatar, October.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learining Research*, 3:951–991.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Jornal of Machine Learning Research*, 7:551–585.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC-2006*, pages 449–454.

Greg Durrett and Dan Klein. 2013. Easy victories and uphill battles in coreference resolution. In *Proceedings of EMNLP-2013*, pages 1971–1982, Seattle, Washington, USA, October.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING-1996*, pages 340–345.

Nicolas R Fauceglia, Yiu-Chang Lin, Xuezhe Ma, and Eduard Hovy. 2015. Word sense disambiguation via propstore and ontonotes for event mention detection. In *Proceedings of the The 3rd Workshop on EVENTS: Definition, Detection, Coreference, and Representation*, pages 11–15, Denver, Colorado, June.

Richard Johansson and Pierre Nugues. 2008. Dependency-based syntactic-semantic analysis with propbank and nombank. In *Proceedings of the CoNLL-2008*, pages 183–187.

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

Xuezhe Ma and Fei Xia. 2014. Unsupervised dependency parsing with transferring distribution via parallel guidance and entropy regularization. In *Proceedings of ACL-2014*, pages 1337–1348, Baltimore, Maryland, June.

Xuezhe Ma and Hai Zhao. 2012a. Fourth-order dependency parsing. In *Proceedings of COLING 2012: Posters*, pages 785–796, Mumbai, India, December.

Xuezhe Ma and Hai Zhao. 2012b. Probabilistic models for high-order projective dependency parsing. *Technical Report, arXiv:1502.04174.*

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of ACL-2013 (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August.

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

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT/EMNLP-2005*, pages 523–530, Vancouver, Canada, October.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 216–220, New York City, June.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of ACL-2013*, pages 92–97, Sofia, Bulgaria, August.

Ryan McDonald. 2006. *Discriminative learning spanning tree algorithm for dependency parsing*. Ph.D. thesis, University of Pennsylvania.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of COLING-2004*, pages 64–70, Geneva, Switzerland, August 23-27.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225, New York City, June.

Sebastian Riedel, Ruket Çakıcı, and Ivan Meza-Ruiz. 2006. Multi-lingual dependency parsing with incremental integer linear programming. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 226–230, New York City, June.

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 *Proceedings of CoNLL-2008*, pages 159–177.

Jun Xie, Haitao Mi, and Qun Liu. 2011. A novel dependency-to-string model for statistical machine translation. In *Proceedings of EMNLP-2011*, pages 216–226, Edinburgh, Scotland, UK., July.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam search. In *Proceedings of EMNLP*, pages 562–571.

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