8-2013

Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers

Andre F.T. Martins
Priberam Labs

Miguel B. Almeida
Universidade de Lisboa

Noah A. Smith
Carnegie Mellon University, nasmith@cs.cmu.edu

Published In
Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), 617-622.

Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers

André F. T. Martins[*†] **Miguel B. Almeida**[*†] **Noah A. Smith**[#]

[*]Priberam Labs, Alameda D. Afonso Henriques, 41, 2º, 1000-123 Lisboa, Portugal
[†]Instituto de Telecomunicações, Instituto Superior Técnico, 1049-001 Lisboa, Portugal
[#]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA
{atm,mba}@priberam.pt, nasmith@cs.cmu.edu

Abstract

We present fast, accurate, direct non-projective dependency parsers with third-order features. Our approach uses AD^3, an accelerated dual decomposition algorithm which we extend to handle specialized head automata and sequential head bigram models. Experiments in fourteen languages yield parsing speeds competitive to projective parsers, with state-of-the-art accuracies for the largest datasets (English, Czech, and German).

1 Introduction

Dependency parsing has become a prominent approach to syntax in the last few years, with increasingly fast and accurate models being devised (Kübler et al., 2009; Huang and Sagae, 2010; Zhang and Nivre, 2011; Rush and Petrov, 2012).

In projective parsing, the arcs in the dependency tree are constrained to be nested, and the problem of finding the best tree can be addressed with dynamic programming. This results in cubic-time decoders for arc-factored and sibling second-order models (Eisner, 1996; McDonald and Pereira, 2006), and quartic-time for grandparent models (Carreras, 2007) and third-order models (Koo and Collins, 2010). Recently, Rush and Petrov (2012) trained third-order parsers with vine pruning cascades, achieving runtimes only a small factor slower than first-order systems. Third-order features have also been included in transition systems (Zhang and Nivre, 2011) and graph-based parsers with cube-pruning (Zhang and McDonald, 2012).

Unfortunately, **non-projective dependency parsers** (appropriate for languages with a more flexible word order, such as Czech, Dutch, and German) lag behind these recent advances. The main obstacle is that non-projective parsing is NP-hard beyond arc-factored models (McDonald and Satta, 2007). Approximate parsers have therefore been introduced, based on belief propagation (Smith and Eisner, 2008), dual decomposition (Koo et al., 2010), or multi-commodity flows (Martins et al., 2009, 2011). These are all instances of **turbo parsers**, as shown by Martins et al. (2010): the underlying approximations come from the fact that they run global inference in factor graphs ignoring loop effects. While this line of research has led to accuracy gains, none of these parsers use third-order contexts, and their speeds are well behind those of projective parsers.

This paper bridges the gap above by presenting the following contributions:

- We apply the third-order feature models of Koo and Collins (2010) to **non-projective** parsing.
- This extension is non-trivial since exact dynamic programming is not applicable. Instead, we adapt AD^3, the dual decomposition algorithm proposed by Martins et al. (2011), to handle third-order features, by introducing specialized head automata.
- We make our parser substantially faster than the many-components approach of Martins et al. (2011). While AD^3 requires solving quadratic subproblems as an intermediate step, recent results (Martins et al., 2012) show that they can be addressed with the same oracles used in the subgradient method (Koo et al., 2010). This enables AD^3 to exploit combinatorial subproblems like the the head automata above.

Along with this paper, we provide a free distribution of our parsers, including training code.[1]

2 Dependency Parsing with AD^3

Dual decomposition is a class of optimization techniques that tackle the dual of combinatorial

[1]Released as TurboParser 2.1, and publically available at http://www.ark.cs.cmu.edu/TurboParser.

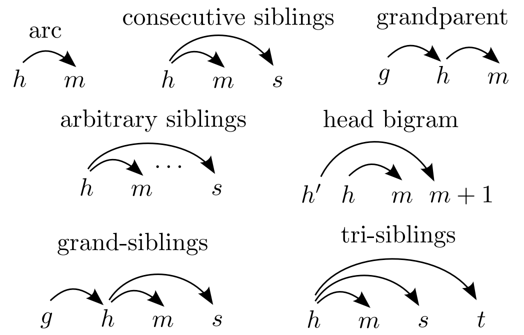

Figure 1: Parts considered in this paper. First-order models factor over arcs (Eisner, 1996; McDonald et al., 2005), and second-order models include also consecutive siblings and grandparents (Carreras, 2007). Our parsers add also *arbitrary* siblings (not necessarily consecutive) and head bigrams, as in Martins et al. (2011), in addition to third-order features for grand- and tri-siblings (Koo and Collins, 2010).

problems in a modular and extensible manner (Komodakis et al., 2007; Rush et al., 2010). In this paper, we employ **alternating directions dual decomposition** (AD^3; Martins et al., 2011). Like the subgradient algorithm of Rush et al. (2010), AD^3 splits the original problem into **local subproblems**, and seeks an agreement on the overlapping variables. The difference is that the AD^3 subproblems have an additional *quadratic* term to accelerate consensus. Recent analysis (Martins et al., 2012) has shown that: (i) AD^3 converges at a faster rate,[2] and (ii) the quadratic subproblems can be solved using the same combinatorial machinery that is used in the subgradient algorithm. This opens the door for larger subproblems (such as the combination of trees and head automata in Koo et al., 2010) instead of a many-components approach (Martins et al., 2011), while still enjoying faster convergence.

2.1 Our Setup

Given a sentence with L words, to which we prepend a root symbol \$, let $A := \{\langle h, m\rangle \mid h \in \{0, \ldots, L\},\ m \in \{1, \ldots, L\},\ h \neq m\}$ be the set of possible dependency arcs. We parameterize a dependency tree via an indicator vector $\boldsymbol{u} := \langle u_a \rangle_{a \in A}$, where u_a is 1 if the arc a is in the tree, and 0 otherwise, and we denote by $\mathcal{Y} \subseteq \mathbb{R}^{|A|}$ the set of such vectors that are indicators of well-formed trees. Let $\{A_s\}_{s=1}^{S}$ be a cover of A, where each $A_s \subseteq A$. We assume that the score of a parse tree $\boldsymbol{u} \in \mathcal{Y}$ decomposes as $f(\boldsymbol{u}) := \sum_{s=1}^{S} f_s(\boldsymbol{z}_s)$, where each $\boldsymbol{z}_s := \langle z_{s,a} \rangle_{a \in A_s}$ is a "partial view" of $\boldsymbol{u}$, and each local score function f_s comes from a feature-based linear model.

Past work in dependency parsing considered either (i) a few "large" components, such as **trees** and **head automata** (Smith and Eisner, 2008; Koo et al., 2010), or (ii) many "small" components, coming from a multi-commodity flow formulation (Martins et al., 2009, 2011). Let $\mathcal{Y}_s \subseteq \mathbb{R}^{|A_s|}$ denote the set of feasible realizations of $\boldsymbol{z}_s$, *i.e.*, those that are partial views of an actual parse tree. A tuple of views $\langle \boldsymbol{z}_1, \ldots, \boldsymbol{z}_S \rangle \in \prod_{s=1}^{S} \mathcal{Y}_s$ is said to be *globally consistent* if $z_{s,a} = z_{s',a}$ holds for every a, s and s' such that $a \in A_s \cap A_{s'}$. We assume each parse $\boldsymbol{u} \in \mathcal{Y}$ corresponds uniquely to a globally consistent tuple of views, and vice-versa. Following Martins et al. (2011), the problem of obtaining the best-scored tree can be written as follows:

$$
\begin{aligned}
&\text{maximize } \textstyle\sum_{s=1}^{S} f_s(\boldsymbol{z}_s)\\
&\quad \text{w.r.t. } \boldsymbol{u} \in \mathbb{R}^{|A|},\ \boldsymbol{z}_s \in \mathcal{Y}_s,\ \forall s\\
&\quad\quad \text{s.t. } z_{s,a} = u_a,\ \forall s,\ \forall a \in A_s, \qquad (1)
\end{aligned}
$$

where the equality constraint ensures that the partial views "glue" together to form a coherent parse tree.[3]

2.2 Dual Decomposition and AD^3

Dual decomposition methods dualize out the equality constraint in Eq. 1 by introducing **Lagrange multipliers** $\lambda_{s,a}$. In doing so, they solve a relaxation where the combinatorial sets $\mathcal{Y}_s$ are replaced by their convex hulls $\mathcal{Z}_s := \text{conv}(\mathcal{Y}_s)$.[4] All that is necessary is the following assumption:

Assumption 1 (Local-Max Oracle). *Every $s \in \{1, \ldots, S\}$ has an oracle that solves efficiently any instance of the following subproblem:*

$$
\begin{aligned}
&\text{maximize } f_s(\boldsymbol{z}_s) + \textstyle\sum_{a \in A_s} \lambda_{s,a} z_{s,a}\\
&\quad \text{w.r.t. } \boldsymbol{z}_s \in \mathcal{Y}_s. \qquad (2)
\end{aligned}
$$

Typically, Assumption 1 is met whenever the maximization of f_s over $\mathcal{Y}_s$ is tractable, since the objective in Eq. 2 just adds a linear function to f_s.

[2] Concretely, AD^3 needs $O(1/\epsilon)$ iterations to converge to a ϵ-accurate solution, while subgradient needs $O(1/\epsilon^2)$.

[3] Note that any tuple $\langle \boldsymbol{z}_1, \ldots, \boldsymbol{z}_S \rangle \in \prod_{s=1}^{S} \mathcal{Y}_s$ satisfying the equality constraints will be globally consistent; this fact, due the assumptions above, will imply $\boldsymbol{u} \in \mathcal{Y}$.

[4] Let $\Delta^{|\mathcal{Y}_s|} := \{\boldsymbol{\alpha} \in \mathbb{R}^{|\mathcal{Y}_s|} \mid \boldsymbol{\alpha} \geq \mathbf{0},\ \sum_{\boldsymbol{y}_s \in \mathcal{Y}_s} \alpha_{\boldsymbol{y}_s} = 1\}$ be the probability simplex. The convex hull of $\mathcal{Y}_s$ is the set $\text{conv}(\mathcal{Y}_s) := \{\sum_{\boldsymbol{y}_s \in \mathcal{Y}_s} \alpha_{\boldsymbol{y}_s} \boldsymbol{y}_s \mid \boldsymbol{\alpha} \in \Delta^{|\mathcal{Y}_s|}\}$. Its members represent marginal probabilities over the arcs in A_s.

The AD^3 algorithm (Martins et al., 2011) alternates among the following iterative updates:

- **z-updates**, which decouple over $s = 1, \ldots, S$, and solve a penalized version of Eq. 2:

$$z_s^{(t+1)} := \arg\max_{z_s \in \mathcal{Z}_s} f_s(z_s) + \sum_{a \in A_s} \lambda_{s,a}^{(t)} z_{s,a} - \frac{\rho}{2} \sum_{a \in A_s} (z_{s,a} - u_a^{(t)})^2. \quad (3)$$

  Above, ρ is a constant and the quadratic term penalizes deviations from the current global solution (stored in $u^{(t)}$).[5] We will see (Prop. 2) that this problem can be solved iteratively using only the Local-Max Oracle (Eq. 2).

- **u-updates**, a simple averaging operation:

$$u_a^{(t+1)} := \frac{1}{|\{s : a \in A_s\}|} \sum_{s : a \in A_s} z_{s,a}^{(t+1)}. \quad (4)$$

- **λ-updates**, where the Lagrange multipliers are adjusted to penalize disagreements:

$$\lambda_{s,a}^{(t+1)} := \lambda_{s,a}^{(t)} - \rho(z_{s,a}^{(t+1)} - u_a^{(t+1)}). \quad (5)$$

In sum, the only difference between AD^3 and the subgradient method is in the z-updates, which in AD^3 require solving a quadratic problem. While closed-form solutions have been developed for some specialized components (Martins et al., 2011), this problem is in general more difficult than the one arising in the subgradient algorithm. However, the following result, proved in Martins et al. (2012), allows to expand the scope of AD^3 to any problem which satisfies Assumption 1.

Proposition 2. *The problem in Eq. 3 admits a solution z_s^* which is spanned by a sparse basis $\mathcal{W} \subseteq \mathcal{Y}_s$ with cardinality at most $|\mathcal{W}| \le O(|A_s|)$. In other words, there is a distribution α with support in $\mathcal{W}$ such that $z_s^* = \sum_{y_s \in \mathcal{W}} \alpha_{y_s} y_s$.*[6]

Prop. 2 has motivated an active set algorithm (Martins et al., 2012) that maintains an estimate of $\mathcal{W}$ by iteratively adding and removing elements computed through the oracle in Eq. 2.[7] Typically, very few iterations are necessary and great speedups are achieved by warm-starting $\mathcal{W}$ with the active set computed in the previous AD^3 iteration. This has a huge impact in practice and is crucial to obtain the fast runtimes in §4 (see Fig. 2).

[5] In our experiments (§4), we set $\rho = 0.05$.

[6] Note that $|\mathcal{Y}_s| = O(2^{|A_s|})$ in general. What Prop. 2 tells us is that the solution of Eq. 3 can be represented as a distribution over $\mathcal{Y}_s$ with a very sparse support.

[7] The algorithm is a specialization of Nocedal and Wright (1999), §16.4, which effectively exploits the sparse representation of z_s^*. For details, see Martins et al. (2012).

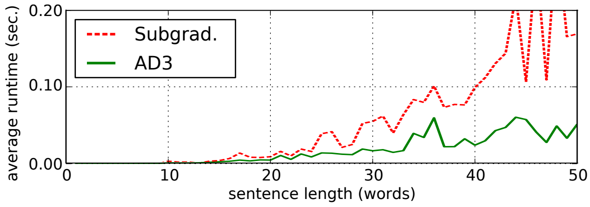

Figure 2: Comparison between AD^3 and subgradient. We show averaged runtimes in PTB §22 as a function of the sentence length. For subgradient, we chose for each sentence the most favorable stepsize in $\{0.001, 0.01, 0.1, 1\}$.

3 Solving the Subproblems

We next describe the actual components used in our third-order parsers.

Tree component. We use an arc-factored score function (McDonald et al., 2005): $f^{\text{TREE}}(z) = \sum_{m=1}^{L} \sigma_{\text{ARC}}(\pi(m), m)$, where $\pi(m)$ is the parent of the mth word according to the parse tree z, and $\sigma_{\text{ARC}}(h, m)$ is the score of an individual arc. The parse tree that maximizes this function can be found in time $O(L^3)$ via the Chu-Liu-Edmonds' algorithm (Chu and Liu, 1965; Edmonds, 1967).[8]

Grand-sibling head automata. Let A_h^{in} and A_h^{out} denote respectively the sets of *incoming* and *outgoing* candidate arcs for the hth word, where the latter subdivides into arcs pointing to the right, $A_{h,\rightarrow}^{\text{out}}$, and to the left, $A_{h,\leftarrow}^{\text{out}}$. Define the sets $A_{h,\rightarrow}^{\text{GSIB}} = A_h^{\text{in}} \cup A_{h,\rightarrow}^{\text{out}}$ and $A_{h,\leftarrow}^{\text{GSIB}} = A_h^{\text{in}} \cup A_{h,\leftarrow}^{\text{out}}$. We describe right-side grand-sibling head automata; their left-side counterparts are analogous. For each head word h in the parse tree z, define $g := \pi(h)$, and let $\langle m_0, m_1, \ldots, m_{p+1} \rangle$ be the sequence of right modifiers of h, with $m_0 = \text{START}$ and $m_{p+1} = \text{END}$. Then, we have the following grand-sibling component:

$$f_{h,\rightarrow}^{\text{GSIB}}(z|_{A_{h,\rightarrow}^{\text{GSIB}}}) = \sum_{k=1}^{p+1} \big(\sigma_{\text{SIB}}(h, m_{k-1}, m_k) \\ \sigma_{\text{GP}}(g, h, m_k) + \sigma_{\text{GSIB}}(g, h, m_{k-1}, m_k)\big),$$

where we use the shorthand $z|_B$ to denote the subvector of z indexed by the arcs in $B \subseteq A$. Note that this score function absorbs grandparent and consecutive sibling scores, in addition to the grand-sibling scores.[9] For each h, $f_{h,\rightarrow}^{\text{GSIB}}$ can be

[8] In fact, there is an asymptotically faster $O(L^2)$ algorithm (Tarjan, 1977). Moreover, if the set of possible arcs is reduced to a subset $B \subseteq A$ (via pruning), then the fastest known algorithm (Gabow et al., 1986) runs in $O(|B| + L \log L)$ time.

[9] Koo et al. (2010) used an identical automaton for their second-order model, but leaving out the grand-sibling scores.

	No pruning	$\|A_m^{\text{in}}\| \leq K$	same, + $\|A_h^{\text{out}}\| \leq J$
TREE	$O(L^2)$	$O(KL + L \log L)$	$O(KL + L \log L)$
GSIB	$O(L^4)$	$O(K^2L^2)$	$O(JK^2L)$
TSIB	$O(L^4)$	$O(KL^3)$	$O(J^2KL)$
SEQ	$O(L^3)$	$O(K^2L)$	$O(K^2L)$
ASIB	$O(L^3)$	$O(KL^2)$	$O(JKL)$

Table 1: Theoretical runtimes of each subproblem without pruning, limiting the number of candidate heads, and limiting (in addition) the number of modifiers. Note the $O(L \log L)$ total runtime per AD^3 iteration in the latter case.

maximized in time $O(L^3)$ with dynamic programming, yielding $O(L^4)$ total runtime.

Tri-sibling head automata. In addition, we define left and right-side tri-sibling head automata that remember the previous two modifiers of a head word. This corresponds to the following component function (for the right-side case):

$$f^{\text{TSIB}}_{h,\rightarrow}(\boldsymbol{z}|_{A^{\text{out}}_{h,\rightarrow}}) = \sum_{k=2}^{p+1} \sigma_{\text{TSIB}}(h, m_{k-2}, m_{k-1}, m_k).$$

Again, each of these functions can be maximized in time $O(L^3)$, yielding $O(L^4)$ runtime.

Sequential head bigram model. Head bigrams can be captured with a simple sequence model:

$$f^{\text{SEQ}}(\boldsymbol{z}) = \sum_{m=2}^{L} \sigma_{\text{HB}}(m, \pi(m), \pi(m-1)).$$

Each score $\sigma_{\text{HB}}(m, h, h')$ is obtained via features that look at the heads of consecutive words (as in Martins et al. (2011)). This function can be maximized in time $O(L^3)$ with the Viterbi algorithm.

Arbitrary siblings. We handle arbitrary siblings as in Martins et al. (2011), defining $O(L^3)$ component functions of the form $f^{\text{ASIB}}_{h,m,s}(z_{\langle h,m\rangle}, z_{\langle h,s\rangle}) = \sigma_{\text{ASIB}}(h, m, s)$. In this case, the quadratic problem in Eq. 3 can be solved directly in constant time.

Tab. 1 details the time complexities of each subproblem. Without pruning, each iteration of AD^3 has $O(L^4)$ runtime. With a simple strategy that limits the number of candidate heads per word to a constant K, this drops to cubic time.[10] Further speed-ups are possible with more pruning: by limiting the number of possible modifiers to a constant J, the runtime would reduce to $O(L \log L)$.

[10]In our experiments, we employed this strategy with $K = 10$, by pruning with a first-order probabilistic model. Following Koo and Collins (2010), for each word m, we also pruned away incoming arcs $\langle h, m\rangle$ with posterior probability less than 0.0001 times the probability of the most likely head.

	UAS	Tok/sec
PTB-YM §22, 1st ord	91.38	4,063
PTB-YM §22, 2nd ord	93.15	1,338
PTB-YM §22, 2nd ord, +ASIB, +HB	93.28	1,018
PTB-YM §22, 3rd ord	93.29	709
PTB-YM §22, 3rd ord, gold tags	94.01	722
This work (PTB-YM §23, 3rd ord)	**93.07**	735
Koo et al. (2010)	92.46	112†
Huang and Sagae (2010)	92.1–	587†
Zhang and Nivre (2011)	92.9–	680†
Martins et al. (2011)	92.53	66†
Zhang and McDonald (2012)	93.06	220
This work (PTB-S §23, 3rd ord)	**92.82**	604
Rush and Petrov (2012)	92.7–	4,460

Table 2: Results for the projective English dataset. We report unlabeled attachment scores (UAS) ignoring punctuation, and parsing speeds in tokens per second. Our speeds include the time necessary for pruning, evaluating features, and decoding, as measured on a Intel Core i7 processor @3.4 GHz. The others are speeds reported in the cited papers; those marked with † were converted from times per sentence.

4 Experiments

We first evaluated our non-projective parser in a *projective* English dataset, to see how its speed and accuracy compares with recent projective parsers, which can take advantage of dynamic programming. To this end, we converted the Penn Treebank to dependencies through (i) the head rules of Yamada and Matsumoto (2003) (PTB-YM) and (ii) basic dependencies from the Stanford parser 2.0.5 (PTB-S).[11] We trained by running 10 epochs of cost-augmented MIRA (Crammer et al., 2006). To ensure valid parse trees at test time, we rounded fractional solutions as in Martins et al. (2009)—yet, solutions were integral $\approx 95\%$ of the time.

Tab. 2 shows the results in the dev-set (top block) and in the test-set (two bottom blocks). In the dev-set, we see consistent gains when more expressive features are added, the best accuracies being achieved with the full third-order model; this comes at the cost of a 6-fold drop in runtime compared with a first-order model. By looking at the two bottom blocks, we observe that our parser has slightly better accuracies than recent projective parsers, with comparable speed levels (with the exception of the highly optimized vine cascade approach of Rush and Petrov, 2012).

[11]We train on sections §02–21, use §22 as validation data, and test on §23. We trained a simple 2nd-order tagger with 10-fold jackknifing to obtain automatic part-of-speech tags for §22–23, with accuracies 97.2% and 96.9%, respectively.

	First Ord.		Sec. Ord.		Third Ord.		Best published UAS			RP12		ZM12
	UAS	Tok/sec	UAS	Tok/sec	UAS	Tok/sec	UAS	Tok/sec		UAS	Tok/sec	UAS
Arabic	77.23	2,481	78.50	388	79.64	197	81.12	-	Ma11	-	-	-
Bulgarian	91.76	5,678	92.82	2,049	93.10	1,273	93.50	-	Ma11	91.9	3,980	93.08
Chinese	88.49	18,094	90.14	4,284	89.98	2,592	91.89	-	Ma10	90.9	7,800	-
Czech	87.66	1,840	90.00	751	**90.32**	501	89.46	-	Ma11	-	-	-
Danish	89.42	4,110	91.20	1,053	91.48	650	91.86	-	Ma11	-	-	
Dutch	83.61	3,884	86.37	1,294	**86.19**	599	85.81	121	Ko10	-	-	-
German	90.52	5,331	91.85	1,788	**92.41**	965	91.89	-	Ma11	90.8	2,880	91.35
English	91.21	3,127	93.03	1,317	**93.22**	785	92.68	-	Ma11	-	-	-
Japanese	92.78	23,895	93.14	5,660	93.52	2,996	93.72	-	Ma11	92.3	8,600	93.24
Portuguese	91.14	4,273	92.71	1,316	92.69	740	93.03	79	Ko10	91.5	2,900	91.69
Slovene	82.81	4,315	85.21	722	86.01	366	86.95	-	Ma11	-	-	-
Spanish	83.61	4,347	84.97	623	85.59	318	87.48	-	ZM12	-	-	**87.48**
Swedish	89.36	5,622	90.98	1,387	91.14	684	91.44	-	ZM12	90.1	5,320	**91.44**
Turkish	75.98	6,418	76.50	1,721	76.90	793	77.55	258	Ko10	-	-	-

Table 3: Results for the CoNLL-2006 datasets and the *non-projective* English dataset of CoNLL-2008. "Best Published UAS" includes the most accurate parsers among Nivre et al. (2006), McDonald et al. (2006), Martins et al. (2010, 2011), Koo et al. (2010), Rush and Petrov (2012), Zhang and McDonald (2012). The last two are shown separately in the rightmost columns.

In our second experiment (Tab. 3), we used 14 datasets, most of which are non-projective, from the CoNLL 2006 and 2008 shared tasks (Buchholz and Marsi, 2006; Surdeanu et al., 2008). Our third-order model achieved the best reported scores for English, Czech, German, and Dutch—which includes the three largest datasets and the ones with the most non-projective dependencies—and is on par with the state of the art for the remaining languages. To our knowledge, the speeds are the highest reported among higher-order non-projective parsers, and only about 3–4 times slower than the vine parser of Rush and Petrov (2012), which has lower accuracies.

5 Conclusions

We presented new third-order non-projective parsers which are both fast and accurate. We decoded with AD^3, an accelerated dual decomposition algorithm which we adapted to handle large components, including specialized head automata for the third-order features, and a sequence model for head bigrams. Results are above the state of the art for large datasets and non-projective languages. In the hope that other researchers may find our implementation useful or are willing to contribute with further improvements, we made our parsers publically available as open source software.

Acknowledgments

We thank all reviewers for their insightful comments and Lingpeng Kong for help in converting the Penn Treebank to Stanford dependencies. This work was partially supported by the EU/FEDER programme, QREN/POR Lisboa (Portugal), under the Intelligo project (contract 2012/24803), by a FCT grant PTDC/EEI-SII/2312/2012, and by NSF grant IIS-1054319.

References

S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *International Conference on Natural Language Learning*.

X. Carreras. 2007. Experiments with a higher-order projective dependency parser. In *International Conference on Natural Language Learning*.

Y. J. Chu and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

J. Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

J. M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of International Conference on Computational Linguistics*, pages 340–345.

H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. 1986. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122.

L. Huang and K. Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proc. of Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086.

N. Komodakis, N. Paragios, and G. Tziritas. 2007. MRF optimization via dual decomposition: Message-passing revisited. In *Proc. of International Conference on Computer Vision*.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In *Proc. of Annual Meeting of the Association for Computational Linguistics*, pages 1–11.

T. Koo, A. M. Rush, M. Collins, T. Jaakkola, and D. Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proc. of Empirical Methods for Natural Language Processing*.

S. Kübler, R. McDonald, and J. Nivre. 2009. *Dependency parsing*. Morgan & Claypool Publishers.

A. F. T. Martins, N. A. Smith, and E. P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

A. F. T. Martins, N. A. Smith, E. P. Xing, M. A. T. Figueiredo, and P. M. Q. Aguiar. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proc. of Empirical Methods for Natural Language Processing*.

A. F. T. Martins, N. A. Smith, P. M. Q. Aguiar, and M. A. T. Figueiredo. 2011. Dual decomposition with many overlapping components. In *Proc. of Empirical Methods for Natural Language Processing*.

A. F. T. Martins, M. A. T. Figueiredo, P. M. Q. Aguiar, N. A. Smith, and E. P. Xing. 2012. Alternating directions dual decomposition. Arxiv preprint arXiv:1212.6550.

R. T. McDonald and F. C. N. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. of Annual Meeting of the European Chapter of the Association for Computational Linguistics*.

R. McDonald and G. Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proc. of International Conference on Parsing Technologies*.

R. T. McDonald, F. Pereira, K. Ribarov, and J. Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of Empirical Methods for Natural Language Processing*.

R. McDonald, K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proc. of International Conference on Natural Language Learning*.

J. Nivre, J. Hall, J. Nilsson, G. Eryiğit, and S. Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Procs. of International Conference on Natural Language Learning*.

J. Nocedal and S. J. Wright. 1999. *Numerical optimization*. Springer-Verlag.

Alexander M Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proc. of Conference of the North American Chapter of the Association for Computational Linguistics*.

A. Rush, D. Sontag, M. Collins, and T. Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proc. of Empirical Methods for Natural Language Processing*.

D. Smith and J. Eisner. 2008. Dependency parsing by belief propagation. In *Proc. of Empirical Methods for Natural Language Processing*.

M. Surdeanu, R. Johansson, A. Meyers, L. Màrquez, and J. Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. *Proc. of International Conference on Natural Language Learning*.

R.E. Tarjan. 1977. Finding optimum branchings. *Networks*, 7(1):25–36.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. of International Conference on Parsing Technologies*.

H. Zhang and R. McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proc. of Empirical Methods in Natural Language Processing*.

Y. Zhang and J. Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.