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

Follow this and additional works at: http://repository.cmu.edu/lti
Part of the Computer Sciences Commons

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 Henriedes, 41, 2°, 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 AD³, 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 AD³, 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 AD³ 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 AD³ to exploit combinatorial subproblems like the head automata above.

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

2 Dependency Parsing with AD³

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

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

Koo et al., 2010) instead of a many-components problem can be solved using the same combinatorial machinery that is necessary and sufficient to accelerate consensus. Recent analysis (Martins et al., 2011), the problem of obtaining 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:

\[
\begin{align*}
\text{maximize} & \quad \sum_{s=1}^{S} f_s(z_s) \\
\text{w.r.t.} & \quad u \in \mathbb{R}^{|A|}, \quad z_s \in \mathbb{Y}_s, \quad \forall s \\
\text{s.t.} & \quad z_{s,a} = u_a, \quad \forall s, \quad \forall a \in A_s, \\
& \quad (1)
\end{align*}
\]

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 \(\mathbb{Y}_s\) are replaced by their convex hulls \(\mathbb{Z}_s := \text{conv}(\mathbb{Y}_s)\). 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:

\[
\begin{align*}
\text{maximize} & \quad f_s(z_s) + \sum_{a \in A_s} \lambda_{s,a} z_{s,a} \\
\text{w.r.t.} & \quad z_s \in \mathbb{Z}_s. \\
& \quad (2)
\end{align*}
\]

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

\(^3\)Note that any tuple \(\langle z_1, \ldots, z_S \rangle \in \prod_{s=1}^{S} \mathbb{Y}_s\) satisfying the equality constraints will be globally consistent; this fact, due the assumptions above, will imply \(u \in \mathbb{Y}\).

\(^4\)Let \(\Delta[|\mathbb{Y}_s|] := \{\alpha \in \mathbb{R}^{|\mathbb{Y}_s|} \mid \alpha \geq 0, \sum_{s} \alpha_{y_s} = 1\}\) be the probability simplex. The convex hull of \(\mathbb{Y}_s\) is the set \(\text{conv}(\mathbb{Y}_s) := \{\sum_{s \in \mathbb{Y}_s} \alpha_{y_s} y_s \mid \alpha \in \Delta[|\mathbb{Y}_s|]\}\). Its members represent marginal probabilities over the arcs in \(A_s\).
The AD³ 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:
  \[
  z_s^{(t+1)} := \arg \max_{z_s \in \mathcal{Z}_s} \left( f_s(z_s) + \sum_{a \in A_s} \lambda_{s,a} z_{s,a} - \frac{\rho}{2} \sum_{a \in A_s} (z_{s,a} - u_a^{(t)})^2 \right). \tag{3}
  \]
  Above, \( \rho \) is a constant and the quadratic term penalizes deviations from the current global solution (stored in \( u_a^{(t)} \)). 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:
  \[
  u_a^{(t+1)} := \frac{1}{|\{s : a \in A_s\}|} \sum_{s : a \in A_s} z_s^{(t+1)} \tag{4}
  \]

- **λ-updates**, where the Lagrange multipliers are adjusted to penalize disagreements:
  \[
  \lambda_{s,a}^{(t+1)} := \lambda_{s,a}^{(t)} - \rho(z_{s,a}^{(t+1)} - u_a^{(t+1)}). \tag{5}
  \]

In sum, the only difference between AD³ and the subgradient method is in the \( z \)-updates, which in AD³ 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 AD³ 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 \( |\mathcal{W}| \leq O(|A_s|) \). In other words, there is a distribution \( \alpha \) with support in \( \mathcal{W} \) such that \( z_s^* = \sum_{y_s \in \mathcal{W}} \alpha_y y_s \).

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. Typically, very few iterations are necessary and great speed-ups are achieved by warm-starting \( \mathcal{W} \) with the active set computed in the previous AD³ iteration. This has a huge impact in practice and is crucial to obtain the fast runtimes in §4 (see Fig. 2).

---

3In our experiments (§4), we set \( \rho = 0.05 \).

4Note 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.

5The 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 AD³ 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):

\[
  f^{\text{TREE}}(z) = \sum_{m=1}^{M} \sigma_{\text{ARC}}(\pi(m), m), \tag{6}
  \]

where \( \pi(m) \) is the parent of the \( m \)-th 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).

**Grand-sibling head automata.** Let \( A^\text{in}_h \) and \( A^\text{out}_h \) denote respectively the sets of incoming and outgoing candidate arcs for the \( h \)-th word, where the latter subdivides into arcs pointing to the right, \( A^\text{out}_h \rightarrow \), and to the left, \( A^\text{out}_h \leftarrow \). Define the sets \( A^\text{GSIB}_h \rightarrow = A^\text{in}_h \cup A^\text{out}_h \rightarrow \), and \( A^\text{GSIB}_h \leftarrow = A^\text{in}_h \cup A^\text{out}_h \leftarrow \). 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:

\[
  f^\text{GSIB}_{h \rightarrow}(z|A^\text{GSIB}_h) = \sum_{k=1}^{p+1} \left( \sigma_{\text{SIB}}(h, m_{k-1}, m_k) + \sigma_{\text{GP}}(g, h, m_k) \right), \tag{7}
  \]

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. For each \( h \), \( f^\text{GSIB}_{h \rightarrow} \) can be

---

8In 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.

9Koo et al. (2010) used an identical automaton for their second-order model, but leaving out the grand-sibling scores.
maximized in time $O(L^3)$ with dynamic program-
ing, yielding $O(L^3)$ total runtime.

**Tri-sibling head automata.** In addition, we de-
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):

$$f_{h,m,s}^{TSIB}(z|_{A_{out}}) = \sum_{k=2}^{P+1} \sigma_{TSIB}(h,m_{k-2},m_{k-1},m_k).$$

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

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

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

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

**Arbitrary siblings.** We handle arbitrary siblings
as in Martins et al. (2011), defining $O(L^3)$ compo-
nent functions of the form $f_{h,m,s}^{ASIB}(z_{h,m}, z_{h,s}) = \sigma_{ASIB}(h,m,s)$. In this case, the quadra-
tic problem in Eq. 3 can be solved directly in constant
time.

Tab. 1 details the time complexities of each sub-
problem 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.

**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 program-
ing. To this end, we converted the Penn Tree-
tank 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).\(^1\) 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 ex-
pressive features are added, the best accuracies be-
ing achieved with the full third-order model; this
comes at the cost of a 6-fold drop in runtime com-
pared 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

---

\(^{10}\)In our experiments, we employed this strategy with $K = 10$, by pruning with a first-order probabilistic model. Follow-
ing 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.

\(^{11}\)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.
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


