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 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

1Released as TurboParser 2.1, and publicly available at http://www.ark.cs.cmu.edu/TurboParser.
problems in a modular and extensible manner (Komedakis et al., 2007; Rush et al., 2010). In this paper, we employ alternating directions dual decomposition (AD³; Martins et al., 2011). Like the subgradient algorithm of Rush et al. (2010), AD³ splits the original problem into local subproblems, and seeks an agreement on the overlapping variables. The difference is that the AD³ subproblems have an additional quadratic term to accelerate consensus. Recent analysis (Martins et al., 2012) has shown that: (i) AD³ converges at a faster rate, 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 \( A := \{ (h, m) \mid h \in \{0, \ldots, L\}, m \in \{1, \ldots, L\}, h \neq m \} \) be the set of possible dependency arcs. We parameterize a dependency tree via an indicator vector \( u := \{ u_a \}_{a \in A} \), where \( u_a = 1 \) if the arc \( a \) is in the tree, and 0 otherwise, and we denote by \( Y_s \subseteq [A] \) 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 \( u \in Y \) decomposes as \( f(u) := \sum_{s=1}^S f_s(z_s) \), where each \( z_s := (z_{s,a})_{a \in A_s} \) is a “partial view” of \( u \), 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 \( Y_s \subseteq [A] \) denote the set of feasible realizations of \( z_s \), i.e., those that are partial views of an actual parse tree. A tuple of views \( z_1, \ldots, z_S \in \prod_{s=1}^S Y_s \) 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 \( u \in Y \) 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:

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

where the equality constraint ensures that the partial views “glue” together to form a coherent parse tree.³

2.2 Dual Decomposition and AD³

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 \( Y_s \) are replaced by their convex hulls \( Z_s := \text{conv}(Y_s) \).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:

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

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

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

⁴Let \( \Delta^{[Y_s]} := \{ \alpha \in \mathbb{R}^{[Y_s]} \mid \alpha \geq 0, \sum_{a \in A_s} \alpha_{y,a} = 1 \} \) be the probability simplex. The convex hull of \( Y_s \) is the set \( \text{conv}(Y_s) := \{ \sum_{a \in A_s} \alpha_{y,a} y_a \mid \alpha \in \Delta^{[Y_s]} \} \). 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:
  \[
z_s^{(t+1)} := \arg \max_{z_s \in Y_s} f_s(z_s) + \sum_{a \in A_s} \lambda_{s,a}^{(t)} z_{s,a} - \frac{\rho}{2} \sum_{a \in A_s} (z_{s,a} - u_a^{(t)})^2. \tag{3}
\]
  Above, \( \rho \) is a constant and the quadratic term penalizes deviations from the current global solution (stored in \( u_a^{(t)} \)).\(^5\) 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:
  \[
u_a^{(t+1)} := \frac{1}{|\{s : a \in A_s\}|} \sum_{s : a \in A_s} z_s^{(t+1)}.
\]

- **\( \lambda \)-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 in general is 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 \( W \subseteq Y_s \) with cardinality at most \( |W| \leq O(|A_s|) \). In other words, there is a distribution \( \alpha \) with support in \( W \) such that \( z_s^* = \sum y_s \in W \alpha y_s. \)

Prop. 2 has motivated an active set algorithm (Martins et al., 2012) that maintains an estimate of \( W \) by iteratively adding and removing elements computed through the oracle in Eq. 2.\(^7\) Typically, very few iterations are necessary and great speed-ups are achieved by warm-starting \( W \) 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 \S 4 (see Fig. 2).

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

\(^4\)Note that \(|Y_s| = O(2^{4|A_s|})\) in general. What Prop. 2 tells us is that the solution of Eq. 3 can be represented as a distribution over \( Y_s \), with a very sparse support.

\(^5\)The algorithm is a specialization of Nocedal and Wright (1999), \S 16.4, which effectively exploits the sparse representation of \( z_s \). For details, see Martins et al. (2012).
maximized in time $O(L^3)$ with dynamic programming, yielding $O(L^3)$ 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):

$$f_{TSIB}(z_{A_{out}}) = \sum_{k=2}^{p+1} \sigma_{TSIB}(h, m_{k-2}, m_{k-1}, m_k).$$

Again, each of these functions can be maximized in time $O(L^3)$, yielding $O(L^4)$ 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 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 $f_{ASIB}(z_{A_{out}}) = \sigma_{ASIB}(h, m, s)$. 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. Further speed-ups are possible with more pruning: by limiting the number of possible modifiers to a constant $J$, the runtime would reduce to $O(L \log L)$.

---

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.

| TREE | No pruning | $|A_{out}^m| \leq K$ | same, $|A_{out}^m| \leq J$ |
|------|------------|----------------------|-----------------------------|
| TREE | $O(L^4)$   | $O(KL + L \log L)$   | $O(KL + L \log L)$         |
| GSIB | $O(L^4)$   | $O(K^2 L^2)$          | $O((JL)K^2 L)$              |
| TSIB | $O(L^4)$   | $O(KL^3)$             | $O(J^2KL)$                  |
| SEQ  | $O(L^4)$   | $O(K^2L^2)$           | $O(K^2L)$                   |
| ASIB | $O(L^4)$   | $O(2KL^2)$            | $O(JKL)$                    |

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).

### 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).\textsuperscript{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.

---

\textsuperscript{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 $(h, m)$ with posterior probability less than 0.0001 times the probability of the most likely head.

\textsuperscript{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 90.9\%, 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 publicly 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


