Dual decomposition for parsing with non-projective head automata

The MIT Faculty has made this article openly available. Please share how this access benefits you. Your story matters.

Citation

As Published
http://portal.acm.org/citation.cfm?id=1870783

Publisher
Association for Computational Linguistics

Version
Author's final manuscript

Accessed

Citable Link
http://hdl.handle.net/1721.1/67290

Terms of Use
Creative Commons Attribution-Noncommercial-Share Alike 3.0

Detailed Terms
http://creativecommons.org/licenses/by-nc-sa/3.0/
Dual Decomposition for Parsing with Non-Projective Head Automata

Terry Koo  Alexander M. Rush  Michael Collins  Tommi Jaakkola  David Sontag
MIT CSAIL, Cambridge, MA 02139, USA
{maestro, srush, mcollins, tommi, dsontag}@csail.mit.edu

Abstract
This paper introduces algorithms for non-projective parsing based on dual decomposition. We focus on parsing algorithms for non-projective head automata, a generalization of head-automata models to non-projective structures. The dual decomposition algorithms are simple and efficient, relying on standard dynamic programming and minimum spanning tree algorithms. They provably solve an LP relaxation of the non-projective parsing problem. Empirically the LP relaxation is very often tight: for many languages, exact solutions are achieved on over 98% of test sentences. The accuracy of our models is higher than previous work on a broad range of datasets.

1 Introduction
Non-projective dependency parsing is useful for many languages that exhibit non-projective syntactic structures. Unfortunately, the non-projective parsing problem is known to be NP-hard for all but the simplest models (McDonald and Satta, 2007). There has been a long history in combinatorial optimization of methods that exploit structure in complex problems, using methods such as dual decomposition or Lagrangian relaxation (Lemaréchal, 2001). Thus far, however, these methods are not widely used in NLP.

This paper introduces algorithms for non-projective parsing based on dual decomposition. We focus on parsing algorithms for non-projective head automata, a generalization of the head-automata models of Eisner (2000) and Alshawi (1996) to non-projective structures. These models include non-projective dependency parsing models with higher-order (e.g., sibling and/or grandparent) dependency relations as a special case. Although decoding of full parse structures with non-projective head automata is intractable, we leverage the observation that key components of the decoding can be efficiently computed using combinatorial algorithms. In particular,

1. Decoding for individual head-words can be accomplished using dynamic programming.
2. Decoding for arc-factored models can be accomplished using directed minimum-weight spanning tree (MST) algorithms.

The resulting parsing algorithms have the following properties:
- They are efficient and easy to implement, relying on standard dynamic programming and MST algorithms.
- They provably solve a linear programming (LP) relaxation of the original decoding problem.
- Empirically the algorithms very often give an exact solution to the decoding problem, in which case they also provide a certificate of optimality.

In this paper we first give the definition for non-projective head automata, and describe the parsing algorithm. The algorithm can be viewed as an instance of Lagrangian relaxation; we describe this connection, and give convergence guarantees for the method. We describe a generalization to models that include grandparent dependencies. We then introduce a perceptron-driven training algorithm that makes use of point 1 above.

We describe experiments on non-projective parsing for a number of languages, and in particular compare the dual decomposition algorithm to approaches based on general-purpose linear programming (LP) or integer linear programming (ILP) solvers (Martins et al., 2009). The accuracy of our models is higher than previous work on a broad range of datasets. The method gives exact solutions to the decoding problem, together with a certificate of optimality, on over 98% of test examples for many of the test languages, with parsing times ranging between 0.021 seconds/sentence for the most simple languages/models, to 0.295 seconds/sentence for the
most complex settings. The method compares favorably to previous work using LP/ILP formulations, both in terms of efficiency, and also in terms of the percentage of exact solutions returned.

While the focus of the current paper is on non-projective dependency parsing, the approach opens up new ways of thinking about parsing algorithms for lexicalized formalisms such as TAG (Joshi and Schabes, 1997), CCG (Steedman, 2000), and projective head automata.

2 Related Work
McDonald et al. (2005) describe MST-based parsing for non-projective dependency parsing models with arc-factored decompositions; McDonald and Pereira (2006) make use of an approximate (hill-climbing) algorithm for parsing with more complex models. McDonald and Pereira (2006) and McDonald and Satta (2007) describe complexity results for non-projective parsing, showing that parsing for a variety of models is NP-hard. Riedel and Clarke (2006) describe dual decomposition approaches for other models, and the decomposition we use is very different in nature from those used in graphical models, and the decomposition we use is very different in nature from those used in graphical models (Wainwright et al., 2005; Komodakis et al., 2007). However, the non-projective parsing problem has a very different structure from these models, and the decomposition we use is very different in nature from those used in graphical models. Other work has made extensive use of decomposition approaches for efficiently solving LP relaxations for graphical models (e.g., Sontag et al. (2008)). Methods that incorporate combinatorial solvers within loopy belief propagation (LBP) (Duchi et al., 2007; Smith and Eisner, 2008) are also closely related to our approach. Unlike LBP, our method has strong theoretical guarantees, such as guaranteed convergence and the possibility of a certificate of optimality.

Finally, in other recent work, Rush et al. (2010) describe dual decomposition approaches for other NLP problems.

3 Sibling Models
This section describes a particular class of models, sibling models; the next section describes a dual-decomposition algorithm for decoding these models.

Consider the dependency parsing problem for a sentence with \( n \) words. We define the index set for dependency parsing to be \( I = \{ (i, j) : i \in \{0 \ldots n \}, j \in \{1 \ldots n \}, i \neq j \} \). A dependency parse is a vector \( y = \{ y(i, j) : (i, j) \in I \} \), where \( y(i, j) = 1 \) if a dependency with head word \( i \) and modifier \( j \) is in the parse, 0 otherwise. We use \( i = 0 \) for the root symbol. We define \( \mathcal{Y} \) to be the set of all well-formed non-projective dependency parses (i.e., the set of directed spanning trees rooted at node 0). Given a function \( f : \mathcal{Y} \mapsto \mathbb{R} \) that assigns scores to parse trees, the optimal parse is

\[
y^* = \arg\max_{y \in \mathcal{Y}} f(y)
\]

A particularly simple definition of \( f(y) \) is \( f(y) = \sum_{(i, j) \in I} y(i, j) \theta(i, j) \) where \( \theta(i, j) \) is the score for dependency \( (i, j) \). Models with this form are often referred to as arc-factored models. In this case the optimal parse tree \( y^* \) can be found efficiently using MST algorithms (McDonald et al., 2005).

This paper describes algorithms that compute \( y^* \) for more complex definitions of \( f(y) \); in this section, we focus on algorithms for models that capture interactions between sibling dependencies. To this end, we will find it convenient to define the following notation. Given a vector \( y \), define

\[
y_i = \{ y(i, j) : j = 1 \ldots n, j \neq i \}
\]

Hence \( y_i \) specifies the set of modifiers to word \( i \); note that the vectors \( y_i \) for \( i = 0 \ldots n \) form a partition of the full set of variables.

We then assume that \( f(y) \) takes the form

\[
f(y) = \sum_{i=0}^{n} f_i(y_i)
\]

Thus \( f(y) \) decomposes into a sum of terms, where each \( f_i \) considers modifiers to the \( i \)'th word alone.

In the general case, finding \( y^* = \arg\max_{y \in \mathcal{Y}} f(y) \) under this definition of \( f(y) \) is an NP-hard problem. However for certain
of the variables suitably powerful functions contain longer cycles; or it could contain words that if: 1) $f$ is a good approximation to $y$ in the dual decomposition parsing algorithm. Note that $Y \subseteq Z$, and in all but trivial cases, $Y$ is a strict subset of $Z$. For example, a structure $z \in Z$ could have $z(i, j) = z(j, i) = 1$ for some $(i, j)$; it could contain longer cycles; or it could contain words that do not modify exactly one head. Nevertheless, with suitably powerful functions $f_i$—for example functions based on discriminative models—$z^*$ may be a good approximation to $y^*$. Later we will see that dual decomposition can effectively use MST inference to rule out ill-formed structures.

We now give the main assumption underlying sibling models:

**Assumption 1 (Sibling Decompositions)** A model $f(y)$ satisfies the sibling-decomposition assumption if: 1) $f(y) = \sum_{i=0}^{n} f_i(y_{i_j})$ for some set of functions $f_0 \ldots f_n$. 2) For any $i \in \{0 \ldots n\}$, for any value of the variables $u(i, j) \in \mathbb{R}$ for $j = 1 \ldots n$, it is possible to compute

$$\arg\max_{y_{i_j} \in Z_i} \left( f_i(y_{i_j}) - \sum_j u(i, j)y(i, j) \right)$$

in polynomial time.

The second condition includes additional terms involving $u(i, j)$ variables that modify the scores of individual dependencies. These terms are benign for most definitions of $f_i$, in that they do not alter decoding complexity. They will be of direct use in the dual decomposition parsing algorithm.

**Example 1: Bigram Sibling Models.** Recall that $y_{i_j}$ is a binary vector specifying which words are modifiers to the head-word $i$. Define $l_1 \ldots l_p$ to be the sequence of left modifiers to word $i$ under $y_{i_j}$, and $r_1 \ldots r_q$ to be the set of right modifiers (e.g., consider the case where $n = 5$, $i = 3$, and we have $y(3, 1) = y(3, 5) = 0$, and $y(3, 2) = y(3, 4) = 1$; in this case $p = 1$, $l_1 = 2$, and $q = 1$, $r_1 = 4$). In bigram sibling models, we have

$$f_i(y_{i_j}) = \sum_{k=1}^{p+1} g_L(i, l_{k-1}, l_k) + \sum_{k=1}^{q+1} g_R(i, r_{k-1}, r_k)$$

where $l_0 = r_0 = \text{START}$ is the initial state, and $l_{p+1} = r_{q+1} = \text{END}$ is the end state. The functions $g_L$ and $g_R$ assign scores to bigram dependencies to the left and right of the head. Under this model calculating $\arg\max_{y_{i_j} \in Z_i} \left( f_i(y_{i_j}) - \sum_j u(i, j)y(i, j) \right)$ takes $O(n^2)$ time using dynamic programming, hence the model satisfies Assumption 1.

**Example 2: Head Automata** Head-automata models constitute a second important model type that satisfy the sibling-decomposition assumption (bigram sibling models are a special case of head automata). These models make use of functions $g_R(i, s, s', r)$ where $s \in S$, $s' \in S$ are variables in a set of possible states $S$, and $r$ is an index of a word in the sentence such that $i < r \leq n$. The function $g_R$ returns a cost for taking word $r$ as the next dependency, and transitioning from state $s$ to $s'$. A similar function $g_L$ is defined for left modifiers. We define

$$f_i(y_{i_j}, s_0 \ldots s_q, t_0 \ldots t_p) = \sum_{k=1}^{q} g_R(i, s_{k-1}, s_k, r_k) + \sum_{k=1}^{p} g_L(i, t_{k-1}, t_k, l_k)$$

and it follows that $\arg\max_{y_{i_j} \in Z_i} f_i(y_{i_j})$ can be computed in $O(n|S|^2)$ time using a variant of the Viterbi algorithm, hence the model satisfies the sibling-decomposition assumption.

## 4 The Parsing Algorithm

We now describe the dual decomposition parsing algorithm for models that satisfy Assumption 1. Consider the following generalization of the decoding
variables. In the next section we will show that these constraints enforce agreement between the decoding dynamic programming and MST, respectively.

Thus, it is these constraints that complicate the optimization problem from Eq. 1, where \( f(y) = \sum_i f_i(y_i) \), \( h(y) = \sum_{(i,j) \in \mathcal{I}} \gamma(i,j)y(i,j) \), and \( \gamma(i,j) \in \mathbb{R} \) for all \((i,j)\):

\[
\text{argmax}_{z \in \mathbb{Z}, y \in \mathcal{Y}} f(z) + h(y) \tag{4}
\]

such that \( z(i,j) = y(i,j) \) for all \((i,j) \in \mathcal{I} \) (5)

Although the maximization w.r.t. \( z \) is taken over the set \( \mathcal{Z} \), the constraints in Eq. 5 ensure that \( z = y \) for some \( y \in \mathcal{Y} \), and hence that \( z \in \mathcal{Y} \).

Without the \( z(i,j) = y(i,j) \) constraints, the objective would decompose into the separate maximizations \( z^* = \text{argmax}_{z \in \mathcal{Z}} f(z) \), and \( y^* = \text{argmax}_{y \in \mathcal{Y}} h(y) \), which can be easily solved using dynamic programming and MST, respectively. Thus, it is these constraints that complicate the optimization. Our approach gets around this difficulty by introducing new variables, \( u(i,j) \), that serve to enforce agreement between the \( y(i,j) \) and \( z(i,j) \) variables. In the next section we will show that these \( u(i,j) \) variables are actually Lagrange multipliers for the \( z(i,j) = y(i,j) \) constraints.

Our parsing algorithm is shown in Figure 1. At each iteration \( k \), the algorithm finds \( y^{(k)} \in \mathcal{Y} \) using an MST algorithm, and \( z^{(k)} \in \mathcal{Z} \) through separate decoding of the \((n+1)\) sibling models. The \( u^{(k)} \) variables are updated if \( y^{(k)}(i,j) \neq z^{(k)}(i,j) \) for some \((i,j)\); these updates modify the objective functions for the two decoding steps, and intuitively encourage the \( y^{(k)} \) and \( z^{(k)} \) variables to be equal.

### 4.1 Lagrangian Relaxation

Recall that the main difficulty in solving Eq. 4 was the \( z = y \) constraints. We deal with these constraints using Lagrangian relaxation (Lemaréchal, 2001). We first introduce Lagrange multipliers \( u = \{u(i,j) : (i,j) \in \mathcal{I}\} \), and define the Lagrangian

\[
L(u, y, z) = \int f(z) + h(y) + \sum_{(i,j) \in \mathcal{I}} u(i,j) \gamma(i,j) y(i,j) - z(i,j) \tag{6}
\]

If \( L^* \) is the optimal value of Eq. 4 subject to the constraints in Eq. 5, then for any value of \( u \),

\[
L^* = \max_{z \in \mathbb{Z}, y \in \mathcal{Y}, y = z} L(u, y, z) \tag{7}
\]

This follows because if \( y = z \), the right term in Eq. 6 is zero for any value of \( u \). The dual objective \( L(u) \) is obtained by omitting the \( y = z \) constraint:

\[
L(u) = \max_{z \in \mathbb{Z}, y \in \mathcal{Y}} L(u, y, z) = \max_{z \in \mathbb{Z}} \left( f(z) - \sum_{i,j} u(i,j) \gamma(i,j) z(i,j) \right) + \max_{y \in \mathcal{Y}} \left( h(y) + \sum_{i,j} u(i,j) y(i,j) \right) \tag{8}
\]

Since \( L(u) \) maximizes over a larger space (\( y \) may not equal \( z \)), we have that \( L^* \leq L(u) \) (compare this to Eq. 7). The dual problem, which our algorithm optimizes, is to obtain the tightest such upper bound,

\[
\text{(Dual problem)} \min_{u \in \mathbb{R}^{|\mathcal{Z}|}} L(u). \tag{8}
\]

The dual objective \( L(u) \) is convex, but not differentiable. However, we can use a subgradient method to derive an algorithm that is similar to gradient descent, and which minimizes \( L(u) \). A subgradient of a convex function \( L(u) \) at \( u \) is a vector \( d_u \) such that for all \( v \in \mathbb{R}^{|\mathcal{Z}|} \), \( L(v) \geq L(u) + d_u \cdot (v - u) \). By standard results,

\[
d_{u^{(k)}} = y^{(k)} - z^{(k)}
\]

is a subgradient for \( L(u) \) at \( u = u^{(k)} \), where \( z^{(k)} = \text{argmax}_{z \in \mathcal{Z}} f(z) - \sum_{i,j} u^{(k)}(i,j) z(i,j) \) and \( y^{(k)} = \text{argmax}_{y \in \mathcal{Y}} h(y) + \sum_{i,j} u^{(k)}(i,j) y(i,j) \).
argmax_{y \in Y} h(y) + \sum_{i,j} u^{(k)}(i, j) y(i, j). Subgradient optimization methods are iterative algorithms with updates that are similar to gradient descent:
\[ u^{(k+1)} = u^{(k)} - \alpha_k d_u(u) = u^{(k)} - \alpha_k (y^{(k)} - z^{(k)}), \]
where \( \alpha_k \) is a step size. It is easily verified that the algorithm in Figure 1 uses precisely these updates.

4.2 Formal Guarantees

With an appropriate choice of the step sizes \( \alpha_k \), the subgradient method can be shown to solve the dual problem, i.e.
\[ \lim_{k \to \infty} L(u^{(k)}) = \min_u L(u). \]

See Korte and Vygen (2008), page 120, for details.

As mentioned before, the dual provides an upper bound on the optimum of the primal problem (Eq. 4),
\[ \max_{z \in \mathcal{Z}, y \in \mathcal{Y}, y=z} f(z) + h(y) \leq \min_{u \in \mathbb{R}^{|I|}} L(u). \quad (9) \]

However, we do not necessarily have strong duality—i.e., equality in the above equation—because the sets \( \mathcal{Z} \) and \( \mathcal{Y} \) are discrete sets. That said, for some functions \( h(y) \) and \( f(z) \) strong duality does hold, as stated in the following:

**Theorem 1** If for some \( k \in \{1 \ldots K\} \) in the algorithm in Figure 1, \( y^{(k)}(i, j) = z^{(k)}(i, j) \) for all \((i, j) \in \mathcal{I}\), then \((y^{(k)}, z^{(k)})\) is a solution to the maximization problem in Eq. 4.

**Proof.** We have that \( f(z^{(k)}) + h(y^{(k)}) = L(u^{(k)}, z^{(k)}, y^{(k)}) = L(u^{(k)}) \), where the last equality is because \( y^{(k)}, z^{(k)} \) are defined as the respective argmax’s. Thus, the inequality in Eq. 9 is tight, and \((y^{(k)}, z^{(k)})\) and \( u^{(k)} \) are primal and dual optimal. \( \square \)

Although the algorithm is not guaranteed to satisfy \( y^{(k)} = z^{(k)} \) for some \( k \), by Theorem 1 if it does reach such a state, then we have the guarantee of an exact solution to Eq. 4, with the dual solution \( u \) providing a certificate of optimality. We show in the experiments that this occurs very frequently, in spite of the parsing problem being NP-hard.

It can be shown that Eq. 8 is the dual of an LP relaxation of the original problem. When the conditions of Theorem 1 are satisfied, it means that the LP relaxation is tight for this instance. For brevity we omit the details, except to note that when the LP relaxation is not tight, the optimal primal solution to the LP relaxation could be recovered by averaging methods (Nedić and Ozdaglar, 2009).

5 Grandparent Dependency Models

In this section we extend the approach to consider grandparent relations. In grandparent models each parse tree \( y \) is represented as a vector
\[ y = \{y(i, j) : (i, j) \in \mathcal{I}\} \cup \{y_t(i, j) : (i, j) \in \mathcal{I}\} \]
where we have added a second set of duplicate variables, \( y_t(i, j) \) for all \((i, j) \in \mathcal{I}\). The set of all valid parse trees is then defined as
\[ \mathcal{Y} = \{y : y(i, j) \text{ variables form a directed tree,} \}
\[ y_t(i, j) = y(i, j) \text{ for all } (i, j) \in \mathcal{I}\} \]

We again partition the variables into \( n + 1 \) subsets, \( y_0 \ldots y_n \), by (re)defining
\[ y_{i} = \left\{ y(i, j) : j = 1 \ldots n, j \neq i \right\} \cup \{y_t(k, i) : k = 0 \ldots n, k \neq i \} \]

So as before \( y_i \) contains variables \( y(i, j) \) which indicate which words modify the \( i \)’th word. In addition, \( y_i \) includes \( y_t(k, i) \) variables that indicate the word that \( i \) itself modifies.

The set of all possible values of \( y_i \) is now
\[ \mathcal{Z}_i = \{ y_{i} : y(i, j) \in \{0, 1\} \text{ for } j = 1 \ldots n, j \neq i; \]
\[ y_t(k, i) \in \{0, 1\} \text{ for } k = 0 \ldots n, k \neq i; \]
\[ \sum_k y_t(k, i) = 1 \}

Hence the \( y(i, j) \) variables can take any values, but only one of the \( y_t(k, i) \) variables can be equal to 1 (as only one word can be a parent of word \( i \)). As before, we define \( \mathcal{Z} = \{y : y_i \in \mathcal{Z}_i \text{ for } i = 0 \ldots n\} \).

We introduce the following assumption:

**Assumption 2 (GS Decompositions)**

A model \( f(y) \) satisfies the grandparent/sibling-decomposition (GSD) assumption if: 1) \( f(z) = \sum_{i=0}^n f_i(z_{i}) \) for some set of functions \( f_0 \ldots f_n \). 2) For any \( i \in \{0 \ldots n\} \), for any value of the variables \( u(i, j) \in \mathbb{R} \) for \( j = 1 \ldots n \), and \( v(k, i) \in \mathbb{R} \) for \( k = 0 \ldots n \), it is possible to compute
\[ \arg\max_{z_{i} \in \mathcal{Z}_i} (f_i(z_{i}) - \sum_j u(i, j)z(i, j) - \sum_k v(k, i)z_t(k, i)) \]
in polynomial time.
Again, it follows that we can approximate \( y^* = \arg \max_{y \in Y} \sum_{i=0}^{n} f_i(y_i) \) by \( z^* = \arg \max_{z \in Z} \sum_{i=0}^{n} f_i(z_i) \), by defining \( z_i^* = \arg \max_{z_i \in Z, f_i(z_i)} \) for \( i = 0 \ldots n \). The resulting vector \( z^* \) may be deficient in two respects. First, the variables \( z^*(i,j) \) may not form a well-formed directed spanning tree. Second, we may have \( z^*_i(i,j) \neq z^*(i,j) \) for some values of \( (i,j) \).

**Example 3: Grandparent/Sibling Models**

An important class of models that satisfy Assumption 2 are defined as follows. Again, for a vector \( y_i \) define \( l_1 \ldots l_p \) to be the sequence of left modifiers to word \( i \) under \( y_i \), and \( r_1 \ldots r_q \) to be the set of right modifiers. Define \( k^* \) to the value for \( k \) such that \( y^*_i(k,i) = 1 \). Then the model is defined as follows:

\[
\begin{align*}
  f_i(y_i) &= \sum_{j=1}^{p+1} g_L(i, k^*, l_j-1, l_j) + \sum_{j=1}^{q+1} g_R(i, k^*, r_{j-1}, r_j) \\
  &\text{This is very similar to the bigram-sibling model, but with the modification that the } g_l \text{ and } g_R \text{ functions depend in addition on the value for } k^*. \text{ This allows these functions to model grandparent dependencies such as } (k^*, i, l_j) \text{ and sibling dependencies such as } (i, l_j-1, l_j). \text{ Finding } z^*_i \text{ under the definition can be accomplished in } O(n^3) \text{ time, by decoding the model using dynamic programming separately for each of the } O(n) \text{ possible values of } k^*, \text{ and picking the value for } k^* \text{ that gives the maximum value under these rescodings.} \\
\end{align*}
\]

A dual-decomposition algorithm for models that satisfy the GSD assumption is shown in Figure 2. The algorithm can be justified as an instance of Lagrangian relaxation applied to the problem

\[
\begin{align*}
  \arg \max_{z \in Z, y \in Y} & f(z) + h(y) \\
  \text{with constraints} & \\
  z(i,j) &= y(i,j) \text{ for all } (i,j) \in \mathcal{I} \\
  z^*_i(i,j) &= y^*(i,j) \text{ for all } (i,j) \in \mathcal{I}
\end{align*}
\]

The algorithm employs two sets of Lagrange multipliers, \( u(i,j) \) and \( v(i,j) \), corresponding to constraints in Eqs. 11 and 12. As in Theorem 1, if at any point in the algorithm \( z(k) = y(k) \), then \( (z(k), y(k)) \) is an exact solution to the problem in Eq. 10.

Figure 2: The parsing algorithm for grandparent/sibling-decomposable models.

6 **The Training Algorithm**

In our experiments we make use of discriminative linear models, where for an input sentence \( x \), the score for a parse \( y \) is \( f(y) = w \cdot \phi(x, y) \) where \( w \in \mathbb{R}^d \) is a parameter vector, and \( \phi(x, y) \in \mathbb{R}^d \) is a feature-vector representing parse tree \( y \) in conjunction with sentence \( x \). We will assume that the features decompose in the same way as the sibling-decomposable or grandparent/sibling-decomposable models, that is \( \phi(x, y) = \sum_{i=0}^{n} \phi(x, y_i) \) for some feature vector definition \( \phi(x, y_i) \). In the bigram sibling models in our experiments, we assume that

\[
\phi(x, y_i) = \sum_{k=1}^{p+1} \phi_L(x, i, l_k-1, l_k) + \sum_{k=1}^{q+1} \phi_R(x, i, r_{k-1}, r_k)
\]

where as before \( l_1 \ldots l_p \) and \( r_1 \ldots r_q \) are left and right modifiers under \( y_i \), and where \( \phi_L \) and \( \phi_R \) are feature vector definitions. In the grandparent models in our experiments, we use a similar definition with feature vectors \( \phi_L(x, i, k^*, l_k-1, l_k) \) and \( \phi_R(x, i, k^*, r_{k-1}, r_k) \), where \( k^* \) is the parent for word \( i \) under \( y_i \).

We train the model using the averaged perceptron for structured problems (Collins, 2002). Given the \( i \)'th example in the training set, \( (x^{(i)}, y^{(i)}) \), the perceptron updates are as follows:

- \( z^* = \arg \max_{z \in Z} w \cdot \phi(x^{(i)}, y) \)
- \( w = w + \phi(x^{(i)}, y^{(i)}) - \phi(x^{(i)}, z^*) \)
The first step involves inference over the set \( \mathcal{Z} \), rather than \( \mathcal{Y} \) as would be standard in the perceptron. Thus, decoding during training can be achieved by dynamic programming over head automata alone, which is very efficient.

Our training approach is closely related to local training methods (Punyakanok et al., 2005). We have found this method to be effective, very likely because \( \mathcal{Z} \) is a superset of \( \mathcal{Y} \). Our training algorithm is also related to recent work on training using outer bounds (see, e.g., (Taskar et al., 2003; Finley and Joachims, 2008; Kulesza and Pereira, 2008; Martins et al., 2009)). Note, however, that the LP relaxation optimized by dual decomposition is significantly tighter than \( \mathcal{Z} \). Thus, an alternative approach would be to use the dual decomposition algorithm for inference during training.

7 Experiments

We report results on a number of data sets. For comparison to Martins et al. (2009), we perform experiments for Danish, Dutch, Portuguese, Slovene, Swedish and Turkish data from the CoNLL-X shared task (Buchholz and Marsi, 2006), and English data from the CoNLL-2008 shared task (Surdeanu et al., 2008). We use the official training/test splits for these data sets, and the same evaluation methodology as Martins et al. (2009). For comparison to Smith and Eisner (2008), we also report results on Danish and Dutch using their alternate training/test split. Finally, we report results on the English WSJ treebank, and the Prague treebank. We use feature sets that are very similar to those described in Carreras (2007). We use marginal-based pruning, using marginals calculated from an arc-factored spanning tree model using the matrix-tree theorem (McDonald and Satta, 2007; Smith and Smith, 2007; Koo et al., 2007).

In all of our experiments we set the value \( K \), the maximum number of iterations of dual decomposition in Figures 1 and 2, to be 5,000. If the algorithm does not terminate—i.e., it does not return \((y^{(k)}, z^{(k)})\) within 5,000 iterations—we simply take the parse \( y^{(k)} \) with the maximum value of \( f(y^{(k)}) \) as the output from the algorithm. At first sight 5,000 might appear to be a large number, but decoding is still fast—see Sections 7.3 and 7.4 for discussion.\(^2\)

The strategy for choosing step sizes \( \alpha_k \) is described in Appendix A, along with other details.

We first discuss performance in terms of accuracy, success in recovering an exact solution, and parsing speed. We then describe additional experiments examining various aspects of the algorithm.

7.1 Accuracy

Table 1 shows results for previous work on the various data sets, and results for an arc-factored model with pure MST decoding with our features. (We use the acronym UAS (unlabeled attachment score) for dependency accuracy.) We also show results for the bigram-sibling and grandparent/sibling (G+S) models under dual decomposition. Both the bigram-sibling and G+S models show large improvements over the arc-factored approach; they also compare favorably to previous work—for example the G+S model gives better results than all results reported in the CoNLL-X shared task, on all languages. Note that we use different feature sets from both Martins et al. (2009) and Smith and Eisner (2008).

7.2 Success in Recovering Exact Solutions

Next, we consider how often our algorithms return an exact solution to the original optimization problem, with a certificate—i.e., how often the algorithms in Figures 1 and 2 terminate with \( y^{(k)} = z^{(k)} \) for some value of \( k < 5000 \) (and are thus optimal, by Theorem 1). The CertS and CertG columns in Table 1 give the results for the sibling and G+S models respectively. For all but one setting\(^3\) over 95% of the test sentences are decoded exactly, with 99% exactness in many cases.

For comparison, we also ran both the single-commodity flow and multiple-commodity flow LP relaxations of Martins et al. (2009) with our models and features. We measure how often these relaxations terminate with an exact solution. The results in Table 2 show that our method gives exact solutions more often than both of these relaxations.\(^4\)

In computing the accuracy figures for Martins et al.

---

\(^2\)Note also that the feature vectors \( \phi \) and inner products \( w \cdot \phi \) only need to be computed once, thus saving computation.

\(^3\)The exception is Slovene, which has the smallest training set at only 1534 sentences.

\(^4\)Note, however, that it is possible that the Martins et al. relaxations would have given a higher proportion of integral solutions if their relaxation was used during training.
(2009), we project fractional solutions to a well-formed spanning tree, as described in that paper.

Finally, to better compare the tightness of our LP relaxation to that of earlier work, we consider randomly-generated instances. Table 2 gives results for our model and the LP relaxations of Martins et al. (2009) with randomly generated scores on automata transitions. We again recover exact solutions more often than the Martins et al. relaxations. Note that with random parameters the percentage of exact solutions is significantly lower, suggesting that the exactness of decoding of the trained models is a special case. We speculate that this is due to the high performance of approximate decoding with $Z$ in place of $Y$ under the trained models for $f_i$; the training algorithm described in section 6 may have the tendency to make the LP relaxation tight.

### 7.3 Speed

Table 1, columns TimeS and TimeG, shows decoding times for the dual decomposition algorithms. Table 2 gives speed comparisons to Martins et al. (2009). Our method gives significant speed-ups over the Martins et al. (2009) method, presumably because it leverages the underlying structure of the problem, rather than using a generic solver.

### 7.4 Lazy Decoding

Here we describe an important optimization in the dual decomposition algorithms. Consider the algorithm in Figure 1. At each iteration we must find

$$z_{ij}^{(k)} = \arg\max_{z_{ij} \in Z_i} (f_i(z_{ij}) - \sum_j u^{(k)}(i, j)z(i, j))$$

Figure 3: The average percentage of head automata that must be recomputed on each iteration of dual decomposition on the PTB validation set.

Table 1: A comparison of non-projective automaton-based parsers with results from previous work. MST: Our first-order baseline. Sib/G+S: Non-projective head automata with sibling or grandparent/sibling interactions, decoded via dual decomposition. Ma09: The best UAS of the LP/ILP-based parsers introduced in Martins et al. (2009). Sm08: The best UAS of any LBP-based parser in Smith and Eisner (2008). Mc06: The best UAS reported by McDonald and Pereira (2006). Best: For the CoNLL-X languages only, the best UAS for any parser in the original shared task (Buchholz and Marsi, 2006) or in any column of Martins et al. (2009, Table 1); note that the latter includes McDonald and Pereira (2006), Nivre and McDonald (2008), and Martins et al. (2008). CertS/CertG: Percent of test examples for which dual decomposition produced a certificate of optimality, for Sib/G+S. TimeS/TimeG: Seconds/sentence for test decoding, for Sib/G+S. TrainS/TrainG: Seconds/sentence during training, for Sib/G+S. For consistency of timing, and Pereira (2006), Nivre and McDonald (2008), and Martins et al. (2008). CertS/CertG: Percent of test examples for which dual decomposition produced a certificate of optimality, for Sib/G+S. TimeS/TimeG: Seconds/sentence for test decoding, for Sib/G+S. TrainS/TrainG: Seconds/sentence during training, for Sib/G+S. For consistency of timing, test decoding was carried out on identical machines with zero additional load; however, training was conducted on machines with varying hardware and load. We ran two tests on the CoNLL-08 corpus. Eng1: UAS when testing on the CoNLL-08 validation set, following Martins et al. (2009). Eng2: UAS when testing on the CoNLL-08 test set.
for $i = 0 \ldots n$. However, if for some $i$, $u^{(k)}(i,j) = u^{(k-1)}(i,j)$ for all $j$, then $z^{(k)} = z^{(k-1)}$. In lazy decoding we immediately set $z^{(k)}_{i} = z^{(k-1)}_{i}$ if $u^{(k)}(i,j) = u^{(k-1)}(i,j)$ for all $j$; this check takes $O(n)$ time, and saves us from decoding with the $i$'th automaton. In practice, the updates to $u$ are very sparse, and this condition occurs very often in practice. Figure 3 demonstrates the utility of this method for both sibling automata and G+S automata.

### 7.5 Early Stopping

We also ran experiments varying the value of $K$—the maximum number of iterations—in the dual decomposition algorithms. As before, if we do not find $y^{(k)} = z^{(k)}$ for some value of $k \leq K$, we choose the $y^{(k)}$ with optimal value for $f(y^{(k)})$ as the final solution. Figure 4 shows three graphs: 1) the accuracy of the parser on PTB validation data versus the value for $K$; 2) the percentage of examples where $y^{(k)} = z^{(k)}$ at some point during the algorithm, hence the algorithm returns a certificate of optimality; 3) the percentage of examples where the solution returned is the same as the solution for the algorithm with $K = 5000$ (our original setting). It can be seen for $K$ as small as 250 we get very similar accuracy to $K = 5000$ (see Table 2). In fact, for this setting the algorithm returns the same solution as for $K = 5000$ on 99.59% of the examples. However only 89.29% of these solutions are produced with a certificate of optimality ($y^{(k)} = z^{(k)}$).

#### 7.6 How Good is the Approximation $z^*$?

We ran experiments measuring the quality of $z^* = \arg\max_{z \in \mathcal{Z}} f(z)$, where $f(z)$ is given by the perceptron-trained bigram-sibling model. Because $z^*$ may not be a well-formed tree with $n$ dependencies, we report precision and recall rather than conventional dependency accuracy. Results on the PTB validation set were 91.11%/88.95% precision/recall, which is accurate considering the unconstrained nature of the predictions. Thus the $z^*$ approximation is clearly a good one; we suspect that this is one reason for the good convergence results for the method.

### 7.7 Importance of Non-Projective Decoding

It is simple to adapt the dual-decomposition algorithms in figures 1 and 2 to give projective dependency structures: the set $\mathcal{Y}$ is redefined to be the set:
of all projective structures, with the \( \arg \max \) over \( \mathcal{Y} \) being calculated using a projective first-order parser (Eisner, 2000). Table 3 shows results for projective and non-projective parsing using the dual decomposition approach. For Czech data, where non-projective structures are common, non-projective decoding has clear benefits. In contrast, there is little difference in accuracy between projective and non-projective decoding on English.

8 Conclusions

We have described dual decomposition algorithms for non-projective parsing, which leverage existing dynamic programming and MST algorithms. There are a number of possible areas for future work. As described in section 7.7, the algorithms can be easily modified to consider projective structures by replacing \( \mathcal{Y} \) with the set of projective trees, and then using first-order dependency parsing algorithms in place of MST decoding. This method could be used to derive parsing algorithms that include higher-order features, as an alternative to specialized dynamic programming algorithms. Eisner (2000) describes extensions of head automata to include word senses; we have not discussed this issue in the current paper, but it is simple to develop dual decomposition algorithms for this case, using similar methods to those used for the grandparent models. The general approach should be applicable to other lexicalized syntactic formalisms, and potentially also to decoding in syntax-driven translation. In addition, our dual decomposition approach is well-suited to parallelization. For example, each of the head-automata could be optimized independently in a multi-core or GPU architecture. Finally, our approach could be used with other structured learning algorithms, e.g. Meshi et al. (2010).

A Implementation Details

This appendix describes details of the algorithm, specifically choice of the step sizes \( \alpha_k \), and use of the \( \gamma(i, j) \) parameters.

A.1 Choice of Step Sizes

We have found the following method to be effective. First, define \( \delta = f(z(1)) - f(y(1)) \), where \((z(1), y(1))\) is the output of the algorithm on the first iteration (note that we always have \( \delta \geq 0 \) since \( f(z(1)) = L(u(1)) \)). Then define \( \alpha_k = \delta/(1 + \eta_k) \), where \( \eta_k \) is the number of times that \( L(u(k')) > L(u(k'-1)) \) for \( k' \leq k \). Hence the learning rate drops at a rate of \( 1/(1 + t) \), where \( t \) is the number of times that the dual increases from one iteration to the next.

A.2 Use of the \( \gamma(i, j) \) Parameters

The parsing algorithms both consider a generalized problem that includes \( \gamma(i, j) \) parameters. We now describe how these can be useful. Recall that the optimization problem is to solve \( \arg \max_{z \in \mathcal{Z}, y \in \mathcal{Y}} f(z) + h(y) \) subject to a set of agreement constraints. In our models, \( f(z) \) can be written as \( f'(z) + \sum_{i,j} \alpha(i,j)z(i,j) \) where \( f'(z) \) includes only terms depending on higher-order (non arc-factored) features, and \( \alpha(i,j) \) are weights that consider the dependency between \( i \) and \( j \) alone. For any value of \( 0 \leq \beta \leq 1 \), the problem \( \arg \max_{z \in \mathcal{Z}, y \in \mathcal{Y}} f_2(z) + h_2(y) \) is equivalent to the original problem, if \( f_2(z) = f'(z) + (1 - \beta) \sum_{i,j} \alpha(i,j)z(i,j) \) and \( h_2(y) = \beta \sum_{i,j} \alpha(i,j)y(i,j) \). We have simply shifted the \( \alpha(i,j) \) weights from one model to the other. While the optimization problem remains the same, the algorithms in Figure 1 and 2 will converge at different rates depending on the value for \( \beta \). In our experiments we set \( \beta = 0.001 \), which puts almost all the weight in the head-automata models, but allows weights on spanning tree edges to break ties in MST inference in a sensible way. We suspect this is important in early iterations of the algorithm, when many values for \( u(i,j) \) or \( v(i,j) \) will be zero, and where with \( \beta = 0 \) many spanning tree solutions \( y(k) \) would be essentially random, leading to very noisy updates to the \( u(i,j) \) and \( v(i,j) \) values. We have not tested other values for \( \beta \).

Acknowledgments  MIT gratefully acknowledges the support of Defense Advanced Research Projects Agency (DARPA) Machine Reading Program under Air Force Research Laboratory (AFRL) prime contract no. FA8750-09-C-0181. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of the DARPA, AFRL, or the US government. A. Rush was supported by the GALE program of the DARPA, Contract No. HR0011-06-C-0022. D. Sontag was supported by a Google PhD Fellowship.
References


