

Concise Integer Linear Programming Formulations for Dependency Parsing

André F. T. Martins^{*†} Noah A. Smith^{*} Eric P. Xing^{*}

^{*}School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

[†]Instituto de Telecomunicações, Instituto Superior Técnico, Lisboa, Portugal

{afm, nasmith, epxing}@cs.cmu.edu

Abstract

We formulate the problem of non-projective dependency parsing as a polynomial-sized integer linear program. Our formulation is able to handle non-local output features in an efficient manner; not only is it compatible with prior knowledge encoded as hard constraints, it can also learn soft constraints from data. In particular, our model is able to learn correlations among neighboring arcs (siblings and grandparents), word valency, and tendencies toward nearly-projective parses. The model parameters are learned in a max-margin framework by employing a linear programming relaxation. We evaluate the performance of our parser on data in several natural languages, achieving improvements over existing state-of-the-art methods.

1 Introduction

Much attention has recently been devoted to integer linear programming (ILP) formulations of NLP problems, with interesting results in applications like semantic role labeling (Roth and Yih, 2005; Punyakanok et al., 2004), dependency parsing (Riedel and Clarke, 2006), word alignment for machine translation (Lacoste-Julien et al., 2006), summarization (Clarke and Lapata, 2008), and coreference resolution (Denis and Baldrige, 2007), among others. In general, the rationale for the development of ILP formulations is to incorporate non-local features or global constraints, which are often difficult to handle with traditional algorithms; this way, ILP formulations focus more on the modeling of problems, rather than in algorithm design. While solving an ILP is NP-hard in general, fast solvers are available today that make this a practical solution for many NLP problems.

This paper presents new, concise ILP formulations for projective and non-projective dependency parsing. We believe that our formulations can pave the way for efficient exploitation of global features and constraints in parsing applications, leading to more powerful models. While casting dependency parsing as an ILP has also been attempted by Riedel and Clarke (2006), *efficient* formulations remain an open problem. Our formulations offer, with respect to the latter, the following advantages:

- They are polynomial-sized in the number of variables and constraints, as opposed to requiring exponentially many constraints, eliminating the need for incremental procedures like the cutting-plane algorithm;
- LP relaxations permit fast online discriminative training of the constrained model;
- Soft constraints may be automatically learned from data. In particular, our formulations handle higher-order arc interactions (like siblings and grandparents), model word valency, and can learn to favor nearly-projective parses.

We evaluate the performance of the new parsers on standard parsing tasks in seven languages. The techniques that we present are also compatible with scenarios where expert knowledge is available, for example in the form of first-order logic hard or soft constraints (Richardson and Domingos, 2006; Chang et al., 2008).

2 Dependency Parsing

2.1 Preliminaries

A dependency tree is a lightweight syntactic representation that attempts to capture functional relationships between words. Lately, this formalism has been used as an alternative to phrase-based parsing for a variety of tasks, ranging from machine translation (Ding and Palmer, 2005) to rela-

tion extraction (Culotta and Sorensen, 2004) and question answering (Wang et al., 2007).

Let us first describe formally the set of legal dependency parse trees. Consider a sentence $x = \langle w_0, \dots, w_n \rangle$, where w_i denotes the word at the i -th position, and $w_0 = \$$ is a wall symbol. We form the (complete¹) directed graph $D = \langle V, A \rangle$, with vertices in $V = \{0, \dots, n\}$ (the i -th vertex corresponding to the i -th word) and arcs in $A = V^2$. Using terminology from graph theory, we say that $B \subseteq A$ is an r -**arborescence**² of the directed graph D if $\langle V, B \rangle$ is a (directed) tree rooted at r . We define the set of legal dependency parse trees of x (denoted $\mathcal{Y}(x)$) as the set of 0-arborescences of D , i.e., we admit each arborescence as a potential dependency tree.

Let $y \in \mathcal{Y}(x)$ be a legal dependency tree for x ; if the arc $a = \langle i, j \rangle \in y$, we refer to i as the parent of j (denoted $i = \pi(j)$) and j as a child of i . We also say that a is **projective** (in the sense of Kahane et al., 1998) if any vertex k in the span of a is reachable from i (in other words, if for any k satisfying $\min(i, j) < k < \max(i, j)$, there is a directed path in y from i to k). A dependency tree is called projective if it only contains projective arcs. Fig. 1 illustrates this concept.³

The formulation to be introduced in §3 makes use of the notion of the *incidence vector* associated with a dependency tree $y \in \mathcal{Y}(x)$. This is the binary vector $\mathbf{z} \triangleq \langle z_a \rangle_{a \in A}$ with each component defined as $z_a = \mathbb{I}(a \in y)$ (here, $\mathbb{I}(\cdot)$ denotes the indicator function). Considering simultaneously all incidence vectors of legal dependency trees and taking the convex hull, we obtain a polyhedron that we call the **arborescence polytope**, denoted by $\mathcal{Z}(x)$. Each vertex of $\mathcal{Z}(x)$ can be identified with a dependency tree in $\mathcal{Y}(x)$. The Minkowski-Weyl theorem (Rockafellar, 1970) ensures that $\mathcal{Z}(x)$ has a representation of the form $\mathcal{Z}(x) = \{\mathbf{z} \in \mathbb{R}^{|A|} \mid \mathbf{A}\mathbf{z} \leq \mathbf{b}\}$, for some p -by- $|A|$ matrix \mathbf{A} and some vector \mathbf{b} in \mathbb{R}^p . However, it is not easy to obtain a *compact* representation (where p grows polynomially with the number of words n). In §3, we will provide a compact represen-

¹The general case where $A \subseteq V^2$ is also of interest; it arises whenever a constraint or a lexicon forbids some arcs from appearing in dependency tree. It may also arise as a consequence of a first-stage pruning step where some candidate arcs are eliminated; this will be further discussed in §4.

²Or “directed spanning tree with designated root r .”

³In this paper, we consider *unlabeled* dependency parsing, where only the backbone structure (i.e., the arcs without the labels depicted in Fig. 1) is to be predicted.

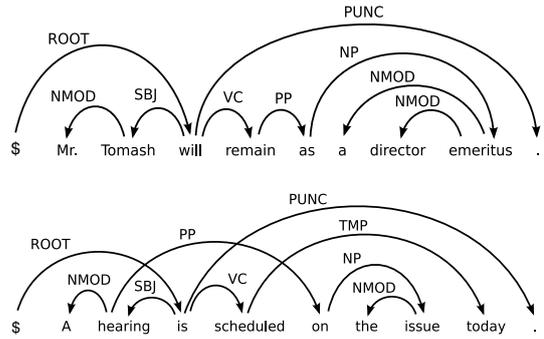


Figure 1: A projective dependency parse (top), and a non-projective dependency parse (bottom) for two English sentences; examples from (McDonald and Satta, 2007).

tation of an outer polytope $\bar{\mathcal{Z}}(x) \supseteq \mathcal{Z}(x)$ whose *integer vertices* correspond to dependency trees. Hence, the problem of finding the dependency tree that maximizes some linear function of the incidence vectors can be cast as an ILP. A similar idea was applied to word alignment by Lacoste-Julien et al. (2006), where permutations (rather than arborescences) were the combinatorial structure being requiring representation.

Letting \mathcal{X} denote the set of possible sentences, define $\mathcal{Y} \triangleq \bigcup_{x \in \mathcal{X}} \mathcal{Y}(x)$. Given a labeled dataset $\mathcal{L} \triangleq \{\langle x_1, y_1 \rangle, \dots, \langle x_m, y_m \rangle\} \subseteq \mathcal{X} \times \mathcal{Y}$, we aim to learn a parser, i.e., a function $h : \mathcal{X} \rightarrow \mathcal{Y}$ that given $x \in \mathcal{X}$ outputs a legal dependency parse $y \in \mathcal{Y}(x)$. The fact that there are exponentially many candidates in $\mathcal{Y}(x)$ makes dependency parsing a structured classification problem.

2.2 Arc Factorization and Locality

There has been much recent work on dependency parsing using graph-based, transition-based, and hybrid methods; see Nivre and McDonald (2008) for an overview. Typical graph-based methods consider linear classifiers of the form

$$h_{\mathbf{w}}(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \mathbf{w}^\top \mathbf{f}(x, y), \quad (1)$$

where $\mathbf{f}(x, y)$ is a vector of features and \mathbf{w} is the corresponding weight vector. One wants $h_{\mathbf{w}}$ to have small expected loss; the typical loss function is the Hamming loss, $\ell(y'; y) \triangleq |\{\langle i, j \rangle \in y' : \langle i, j \rangle \notin y\}|$. Tractability is usually ensured by strong factorization assumptions, like the one underlying the **arc-factored model** (Eisner, 1996; McDonald et al., 2005), which forbids any feature that depends on two or more arcs. This induces a decomposition of the feature vector $\mathbf{f}(x, y)$ as:

$$\mathbf{f}(x, y) = \sum_{a \in y} \mathbf{f}_a(x). \quad (2)$$

Under this decomposition, each arc receives a score; parsing amounts to choosing the configuration that maximizes the overall score, which, as shown by McDonald et al. (2005), is an instance of the **maximal arborescence problem**. Combinatorial algorithms (Chu and Liu, 1965; Edmonds, 1967) can solve this problem in cubic time.⁴ If the dependency parse trees are restricted to be projective, cubic-time algorithms are available via dynamic programming (Eisner, 1996). While in the projective case, the arc-factored assumption can be weakened in certain ways while maintaining polynomial parser runtime (Eisner and Satta, 1999), the same does not happen in the nonprojective case, where finding the highest-scoring tree becomes NP-hard (McDonald and Satta, 2007). Approximate algorithms have been employed to handle models that are not arc-factored (although features are still fairly local): McDonald and Pereira (2006) adopted an approximation based on $O(n^3)$ projective parsing followed by a hill-climbing algorithm to rearrange arcs, and Smith and Eisner (2008) proposed an algorithm based on loopy belief propagation.

3 Dependency Parsing as an ILP

Our approach will build a graph-based parser without the drawback of a restriction to local features. By formulating inference as an ILP, non-local features can be easily accommodated in our model; furthermore, by using a relaxation technique we can still make learning tractable. The impact of LP-relaxed inference in the learning problem was studied elsewhere (Martins et al., 2009).

A **linear program** (LP) is an optimization problem of the form

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^d} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}. \end{aligned} \quad (3)$$

If the problem is feasible, the optimum is attained at a vertex of the polyhedron that defines the constraint space. If we add the constraint $\mathbf{x} \in \mathbb{Z}^d$, then the above is called an **integer linear program** (ILP). For some special parameter settings—e.g., when \mathbf{b} is an integer vector and \mathbf{A} is totally unimodular⁵—all vertices of the constraining polyhedron are integer points; in these cases, the integer constraint may be suppressed and (3) is guaranteed to have integer solutions (Schrijver, 2003).

⁴There is also a quadratic algorithm due to Tarjan (1977).

⁵A matrix is called totally unimodular if the determinants of each square submatrix belong to $\{0, 1, -1\}$.

Of course, this need not happen: solving a general ILP is an NP-complete problem. Despite this fact, fast solvers are available today that make this a practical solution for many problems. Their performance depends on the dimensions and degree of sparsity of the constraint matrix \mathbf{A} .

Riedel and Clarke (2006) proposed an ILP formulation for dependency parsing which refines the arc-factored model by imposing linguistically motivated “hard” constraints that forbid some arc configurations. Their formulation includes an exponential number of constraints—one for each possible cycle. Since it is intractable to throw in all constraints at once, they propose a cutting-plane algorithm, where the cycle constraints are only invoked when violated by the current solution. The resulting algorithm is still slow, and an arc-factored model is used as a surrogate during training (i.e., the hard constraints are only used at test time), which implies a discrepancy between the model that is optimized and the one that is actually going to be used.

Here, we propose ILP formulations that eliminate the need for cycle constraints; in fact, they require only a *polynomial* number of constraints. Not only does our model allow expert knowledge to be injected in the form of constraints, it is also capable of *learning* soft versions of those constraints from data; indeed, it can handle features that are not arc-factored (correlating, for example, siblings and grandparents, modeling valency, or preferring nearly projective parses). While, as pointed out by McDonald and Satta (2007), the inclusion of these features makes inference NP-hard, by *relaxing* the integer constraints we obtain approximate algorithms that are very efficient and competitive with state-of-the-art methods. In this paper, we focus on unlabeled dependency parsing, for clarity of exposition. If it is extended to labeled parsing (a straightforward extension), our formulation fully subsumes that of Riedel and Clarke (2006), since it allows using the same hard constraints and features while keeping the ILP polynomial in size.

3.1 The Arborescence Polytope

We start by describing our *constraint space*. Our formulations rely on a concise polyhedral representation of the set of candidate dependency parse trees, as sketched in §2.1. This will be accomplished through the use of *flow constraints*, by

drawing an analogy with a network flow problem.

Let $D = \langle V, A \rangle$ be the complete directed graph associated with a sentence $x \in \mathcal{X}$, as stated in §2. A subgraph $y = \langle V, B \rangle$ is a legal dependency tree (i.e., $y \in \mathcal{Y}(x)$) if and only if the following conditions are met:

1. Each vertex in $V \setminus \{0\}$ must have exactly one incoming arc in B ,
2. 0 has no incoming arcs in B ,
3. B does not contain cycles.

For each vertex $v \in V$, let $\delta^-(v) \triangleq \{\langle i, j \rangle \in A \mid j = v\}$ denote its set of incoming arcs, and $\delta^+(v) \triangleq \{\langle i, j \rangle \in A \mid i = v\}$ denote its set of outgoing arcs. While the two first conditions can be easily expressed by linear constraints on the incidence vector \mathbf{z} (see Eqs. 7–8 below), condition 3 is somewhat harder to express. Rather than adding exponentially many constraints, one for each potential cycle (like Riedel and Clarke, 2006), we equivalently replace condition 3 by

3'. B is connected.

Note that conditions 1-2-3 are equivalent to 1-2-3', in the sense that both define the same set $\mathcal{Y}(x)$; indeed, the fact that B is connected (3') and each word except the root has a parent (1-2) implies that B is free of cycles (3). However, as we will see, the latter set of conditions is more convenient. Connectedness of graphs can be imposed via flow constraints (by requiring that, for any $v \in V \setminus \{0\}$, there is a directed path in B connecting 0 to v). We adapt the **single commodity flow** formulation for the (undirected) minimum spanning tree problem, due to Magnanti and Wolsey (1994), that requires $O(n^2)$ variables and constraints. Under this model, the root node must send one unit of flow to every other node. By making use of extra variables, $\phi \triangleq \langle \phi_a \rangle_{a \in A}$, to denote the flow of commodities through each arc, we are led to the following constraints (we denote $\mathbb{U} \triangleq [0, 1]$, and $\mathbb{B} \triangleq \{0, 1\} = \mathbb{U} \cap \mathbb{Z}$):

- Root sends flow n :

$$\sum_{a \in \delta^+(0)} \phi_a = n \quad (4)$$

- Each node consumes one unit of flow:

$$\sum_{a \in \delta^-(j)} \phi_a - \sum_{a \in \delta^+(j)} \phi_a = 1, \quad j \in V \setminus \{0\} \quad (5)$$

- Flow is zero on disabled arcs:

$$\phi_a \leq n z_a, \quad a \in A \quad (6)$$

- Each node (except the root) has one incoming arc:

$$\sum_{a \in \delta^-(j)} z_a = 1, \quad j \in V \setminus \{0\} \quad (7)$$

- Root has no incoming arcs:

$$\sum_{a \in \delta^-(0)} z_a = 0 \quad (8)$$

- Each arc indicator lies in the unit interval:

$$z_a \in \mathbb{U}, \quad a \in A. \quad (9)$$

These constraints project an outer bound of the arborescence polytope, i.e.,

$$\begin{aligned} \bar{\mathcal{Z}}(x) &\triangleq \{\mathbf{z} \in \mathbb{R}^{|A|} \mid (\mathbf{z}, \phi) \text{ satisfy (4–9)}\} \\ &\supseteq \mathcal{Z}(x). \end{aligned} \quad (10)$$

Furthermore, the integer points of $\bar{\mathcal{Z}}(x)$ are precisely the incidence vectors of dependency trees in $\mathcal{Y}(x)$; these are obtained by replacing Eq. 9 by

$$z_a \in \mathbb{B}, \quad a \in A. \quad (11)$$

3.2 Arc-Factored Model

Given our polyhedral representation of (an outer bound of) the arborescence polytope, we can now formulate dependency parsing with an arc-factored model as an ILP. By storing the arc-local feature vectors into the columns of a matrix $\mathbf{F}(x) \triangleq [\mathbf{f}_a(x)]_{a \in A}$, and defining the score vector $\mathbf{s} \triangleq \mathbf{F}(x)^\top \mathbf{w}$ (each entry is an arc score) the inference problem can be written as

$$\begin{aligned} \max_{y \in \mathcal{Y}(x)} \mathbf{w}^\top \mathbf{f}(x, y) &= \max_{\mathbf{z} \in \bar{\mathcal{Z}}(x)} \mathbf{w}^\top \mathbf{F}(x) \mathbf{z} \\ &= \max_{\mathbf{z}, \phi} \mathbf{s}^\top \mathbf{z} \\ \text{s.t.} \quad &\mathbf{A} \begin{bmatrix} \mathbf{z} \\ \phi \end{bmatrix} \leq \mathbf{b} \\ &\mathbf{z} \in \mathbb{B} \end{aligned} \quad (12)$$

where \mathbf{A} is a sparse constraint matrix (with $O(|A|)$ non-zero elements), and \mathbf{b} is the constraint vector; \mathbf{A} and \mathbf{b} encode the constraints (4–9). This is an ILP with $O(|A|)$ variables and constraints (hence, quadratic in n); if we drop the integer constraint the problem becomes the LP relaxation. As is, this formulation is no more attractive than solving the problem with the existing combinatorial algorithms discussed in §2.2; however, we can now start adding non-local features to build a more powerful model.

3.3 Sibling and Grandparent Features

To cope with higher-order features of the form $\mathbf{f}_{a_1, \dots, a_K}(x)$ (i.e., features whose values depend on the simultaneous inclusion of arcs a_1, \dots, a_K on a candidate dependency tree), we employ a linearization trick (Boros and Hammer, 2002), defining extra variables $z_{a_1 \dots a_K} \triangleq z_{a_1} \wedge \dots \wedge z_{a_K}$. This logical relation can be expressed by the following $O(K)$ agreement constraints:⁶

$$\begin{aligned} z_{a_1 \dots a_K} &\leq z_{a_i}, \quad i = 1, \dots, K \\ z_{a_1 \dots a_K} &\geq \sum_{i=1}^K z_{a_i} - K + 1. \end{aligned} \quad (13)$$

As shown by McDonald and Pereira (2006) and Carreras (2007), the inclusion of features that correlate sibling and grandparent arcs may be highly beneficial, even if doing so requires resorting to approximate algorithms.⁷ Define $\mathcal{R}^{\text{sibl}} \triangleq \{\langle i, j, k \rangle \mid \langle i, j \rangle \in A, \langle i, k \rangle \in A\}$ and $\mathcal{R}^{\text{grand}} \triangleq \{\langle i, j, k \rangle \mid \langle i, j \rangle \in A, \langle j, k \rangle \in A\}$. To include such features in our formulation, we need to add extra variables $\mathbf{z}^{\text{sibl}} \triangleq \langle z_r \rangle_{r \in \mathcal{R}^{\text{sibl}}}$ and $\mathbf{z}^{\text{grand}} \triangleq \langle z_r \rangle_{r \in \mathcal{R}^{\text{grand}}}$ that indicate the presence of sibling and grandparent arcs. Observe that these indicator variables are *conjunctions* of arc indicator variables, i.e., $z_{ijk}^{\text{sibl}} = z_{ij} \wedge z_{ik}$ and $z_{ijk}^{\text{grand}} = z_{ij} \wedge z_{jk}$. Hence, these features can be handled in our formulation by adding the following $O(|A| \cdot |V|)$ variables and constraints:

$$\begin{aligned} z_{ijk}^{\text{sibl}} &\leq z_{ij} \\ z_{ijk}^{\text{sibl}} &\leq z_{ik} \\ z_{ijk}^{\text{sibl}} &\geq z_{ij} + z_{ik} - 1 \end{aligned} \quad (14)$$

for all triples $\langle i, j, k \rangle \in \mathcal{R}^{\text{sibl}}$, and

$$\begin{aligned} z_{ijk}^{\text{grand}} &\leq z_{ij} \\ z_{ijk}^{\text{grand}} &\leq z_{jk} \\ z_{ijk}^{\text{grand}} &\geq z_{ij} + z_{jk} - 1 \end{aligned} \quad (15)$$

for all triples $\langle i, j, k \rangle \in \mathcal{R}^{\text{grand}}$. Let $\mathcal{R} \triangleq A \cup \mathcal{R}^{\text{sibl}} \cup \mathcal{R}^{\text{grand}}$; by redefining $\mathbf{z} \triangleq \langle z_r \rangle_{r \in \mathcal{R}}$ and $\mathbf{F}(x) \triangleq [\mathbf{f}_r(x)]_{r \in \mathcal{R}}$, we may express our inference problem as in Eq. 12, with $O(|A| \cdot |V|)$ variables and constraints.

⁶Actually, any logical condition can be encoded with linear constraints involving binary variables; see e.g. Clarke and Lapata (2008) for an overview.

⁷By *sibling features* we mean features that depend on pairs of sibling arcs (i.e., of the form $\langle i, j \rangle$ and $\langle i, k \rangle$); by *grandparent features* we mean features that depend on pairs of grandparent arcs (of the form $\langle i, j \rangle$ and $\langle j, k \rangle$).

Notice that the strategy just described to handle sibling features is not fully compatible with the features proposed by Eisner (1996) for projective parsing, as the latter correlate only *consecutive* siblings and are also able to place special features on the *first* child of a given word. The ability to handle such “ordered” features is intimately associated with Eisner’s dynamic programming parsing algorithm and with the Markovian assumptions made explicitly by his generative model. We next show how similar features can be incorporated in our model by adding “dynamic” constraints to our ILP. Define

$$\begin{aligned} z_{ijk}^{\text{next sibl}} &\triangleq \begin{cases} 1, & \text{if } \langle i, j \rangle \text{ and } \langle i, k \rangle \text{ are} \\ & \text{consecutive siblings,} \\ 0, & \text{otherwise.} \end{cases} \\ z_{ij}^{\text{first child}} &\triangleq \begin{cases} 1, & \text{if } j \text{ is the first child of } i, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \quad (16)$$

Suppose (without loss of generality) that $i < j < k \leq n$. We could naively compose the constraints (14) with additional linear constraints that encode the logical relation

$$z_{ijk}^{\text{next sibl}} = z_{ijk}^{\text{sibl}} \wedge \bigwedge_{j < l < k} \neg z_{il},$$

but this would yield a constraint matrix with $O(n^4)$ non-zero elements. Instead, we define auxiliary variables β_{jk} and γ_{ij} :

$$\begin{aligned} \beta_{jk} &= \begin{cases} 1, & \text{if } \exists l \text{ s.t. } \pi(l) = \pi(j) < j < l < k \\ 0, & \text{otherwise} \end{cases} \\ \gamma_{ij} &= \begin{cases} 1, & \text{if } \exists k \text{ s.t. } i < k < j \text{ and } \langle i, k \rangle \in y \\ 0, & \text{otherwise} \end{cases} \end{aligned} \quad (17)$$

Then, we have that $z_{ijk}^{\text{next sibl}} = z_{ijk}^{\text{sibl}} \wedge (\neg \beta_{jk})$ and $z_{ij}^{\text{first child}} = z_{ij} \wedge (\neg \gamma_{ij})$, which can be encoded via

$$\begin{aligned} z_{ijk}^{\text{next sibl}} &\leq z_{ijk}^{\text{sibl}} \\ z_{ijk}^{\text{next sibl}} &\leq 1 - \beta_{jk} \\ z_{ijk}^{\text{next sibl}} &\geq z_{ijk}^{\text{sibl}} - \beta_{jk} \\ z_{ij}^{\text{first child}} &\leq z_{ij} \\ z_{ij}^{\text{first child}} &\leq 1 - \gamma_{ij} \\ z_{ij}^{\text{first child}} &\geq z_{ij} - \gamma_{ij} \end{aligned} \quad (18)$$

The following “dynamic” constraints encode the

logical relations for the auxiliary variables (17):

$$\beta_{j(j+1)} = 0 \quad (19)$$

$$\beta_{j(k+1)} \geq \beta_{jk}$$

$$\beta_{j(k+1)} \geq \sum_{i < j} z_{ijk}^{\text{sibl}}$$

$$\beta_{j(k+1)} \leq \beta_{jk} + \sum_{i < j} z_{ijk}^{\text{sibl}}$$

$$\gamma_{i(i+1)} = 0 \quad (20)$$

$$\gamma_{i(j+1)} \geq \gamma_{ij}$$

$$\gamma_{i(j+1)} \geq z_{ij}$$

$$\gamma_{i(j+1)} \leq \gamma_{ij} + z_{ij}$$

Auxiliary variables and constraints are defined analogously for the case $n \geq i > j > k$. This results in a sparser constraint matrix, with only $O(n^3)$ non-zero elements.

3.4 Valency Features

A crucial fact about dependency grammars is that words have preferences about the number and arrangement of arguments and modifiers they accept. Therefore, it is desirable to include features that indicate, for a candidate arborescence, how many outgoing arcs depart from each vertex; denote these quantities by $v_i \triangleq \sum_{a \in \delta^+(i)} z_a$, for each $i \in V$. We call v_i the **valency** of the i th vertex. We add valency indicators $z_{ik}^{\text{val}} \triangleq \mathbb{I}(v_i = k)$ for $i \in V$ and $k = 0, \dots, n-1$. This way, we are able to penalize candidate dependency trees that assign unusual valencies to some of their vertices, by specifying a individual cost for each possible value of valency. The following $O(|V|^2)$ constraints encode the agreement between valency indicators and the other variables:

$$\sum_{k=0}^{n-1} k z_{ik}^{\text{val}} = \sum_{a \in \delta^+(i)} z_a, \quad i \in V \quad (21)$$

$$\sum_{k=0}^{n-1} z_{ik}^{\text{val}} = 1, \quad i \in V$$

$$z_{ik}^{\text{val}} \geq 0, \quad i \in V, k \in \{0, \dots, n-1\}$$

3.5 Projectivity Features

For most languages, dependency parse trees tend to be nearly projective (cf. Buchholz and Marsi, 2006). We wish to make our model capable of learning to prefer “nearly” projective parses whenever that behavior is observed in the data.

The **multicommodity directed flow** model of Magnanti and Wolsey (1994) is a refinement of the model described in §3.1 which offers a compact and elegant way to indicate nonprojective arcs, requiring $O(n^3)$ variables and constraints. In this model, every node $k \neq 0$ defines a commodity:

one unit of commodity k originates at the root node and must be delivered to node k ; the variable ϕ_{ij}^k denotes the flow of commodity k in arc $\langle i, j \rangle$. We first replace (4–9) by (22–26):

- The root sends one unit of commodity to each node:

$$\sum_{a \in \delta^-(0)} \phi_a^k - \sum_{a \in \delta^+(0)} \phi_a^k = -1, \quad k \in V \setminus \{0\} \quad (22)$$

- Any node consumes its own commodity and no other:

$$\sum_{a \in \delta^-(j)} \phi_a^k - \sum_{a \in \delta^+(j)} \phi_a^k = \delta_j^k, \quad j, k \in V \setminus \{0\} \quad (23)$$

where $\delta_j^k \triangleq \mathbb{I}(j = k)$ is the Kronecker delta.

- Disabled arcs do not carry any flow:

$$\phi_a^k \leq z_a, \quad a \in A, k \in V \quad (24)$$

- There are exactly n enabled arcs:

$$\sum_{a \in A} z_a = n \quad (25)$$

- All variables lie in the unit interval:

$$z_a \in \mathbb{U}, \quad \phi_a^k \in \mathbb{U}, \quad a \in A, k \in V \quad (26)$$

We next define auxiliary variables ψ_{jk} that indicate if there is a path from j to k . Since each vertex except the root has only one incoming arc, the following linear equalities are enough to describe these new variables:

$$\begin{aligned} \psi_{jk} &= \sum_{a \in \delta^-(j)} \phi_a^k, \quad j, k \in V \setminus \{0\} \\ \psi_{0k} &= 1, \quad k \in V \setminus \{0\}. \end{aligned} \quad (27)$$

Now, define indicators $z^{\text{np}} \triangleq \langle z_a^{\text{np}} \rangle_{a \in A}$, where

$$z_a^{\text{np}} \triangleq \mathbb{I}(a \in y \text{ and } a \text{ is nonprojective}).$$

From the definition of projective arc in §2.1, we have that $z_a^{\text{np}} = 1$ if and only if the arc is active ($z_a = 1$) and there is some vertex k in the span of $a = \langle i, j \rangle$ such that $\psi_{ik} = 0$. We are led to the following $O(|A| \cdot |V|)$ constraints for $\langle i, j \rangle \in A$:

$$\begin{aligned} z_{ij}^{\text{np}} &\leq z_{ij} \\ z_{ij}^{\text{np}} &\geq z_{ij} - \psi_{ik}, \quad \min(i, j) \leq k \leq \max(i, j) \\ z_{ij}^{\text{np}} &\leq -\sum_{k=\min(i, j)+1}^{\max(i, j)-1} \psi_{ik} + |j - i| - 1 \end{aligned}$$

There are other ways to introduce nonprojectivity indicators and alternative definitions of “non-projective arc.” For example, by using dynamic constraints of the same kind as those in §3.3, we can indicate arcs that “cross” other arcs with $O(n^3)$ variables and constraints, and a cubic number of non-zero elements in the constraint matrix (omitted for space).

3.6 Projective Parsing

It would be straightforward to adapt the constraints in §3.5 to allow only projective parse trees: simply force $z_a^{\text{pp}} = 0$ for any $a \in A$. But there are more efficient ways of accomplish this. While it is difficult to impose projectivity constraints *or* cycle constraints individually, there is a simpler way of imposing *both*. Consider 3 (or 3') from §3.1.

Proposition 1 *Replace condition 3 (or 3') with*

3''. *If $\langle i, j \rangle \in B$, then, for any $k = 1, \dots, n$ such that $k \neq j$, the parent of k must satisfy (defining $i' \triangleq \min(i, j)$ and $j' \triangleq \max(i, j)$):*

$$\left\{ \begin{array}{ll} i' \leq \pi(k) \leq j', & \text{if } i' < k < j', \\ \pi(k) < i' \vee \pi(k) > j', & \text{if } k < i' \text{ or } k > j' \\ & \text{or } k = i. \end{array} \right.$$

*Then, $\mathcal{Y}(x)$ will be redefined as the set of **projective** dependency parse trees.*

We omit the proof for space. Conditions 1, 2, and 3'' can be encoded with $O(n^2)$ constraints.

4 Experiments

We report experiments on seven languages, six (Danish, Dutch, Portuguese, Slovene, Swedish and Turkish) from the CoNLL-X shared task (Buchholz and Marsi, 2006), and one (English) from the CoNLL-2008 shared task (Surdeanu et al., 2008).⁸ All experiments are evaluated using the *unlabeled attachment score* (UAS), using the default settings.⁹ We used the same arc-factored features as McDonald et al. (2005) (included in the MSTParser toolkit¹⁰); for the higher-order models described in §3.3–3.5, we employed simple higher order features that look at the word, part-of-speech

⁸We used the provided train/test splits except for English, for which we tested on the development partition. For training, sentences longer than 80 words were discarded. For testing, all sentences were kept (the longer one being 118 words long).

⁹<http://nextens.uvt.nl/~conll/software.html>.

¹⁰<http://sourceforge.net/projects/mstparser>.

tag, and (if available) morphological information of the words being correlated through the indicator variables. For scalability (and noting that some of the models require $O(|V| \cdot |A|)$ constraints and variables, which, when $A = V^2$, grows cubically with the number of words), we first prune the base graph by running a simple algorithm that ranks the k -best candidate parents for each word in the sentence; this reduces the number of candidate arcs to $|A| = kn$.¹¹ This strategy is similar to the one employed by Carreras et al. (2008) to prune the search space of the actual parser. In our experiments, we set $k = 10$.¹² The ranker is a local model trained using a max-margin criterion; it is arc-factored and not subject to *any* structural constraints, so it is fast.

The actual parser was trained via the online structured passive-aggressive algorithm of Crammer et al. (2006); it differs from the 1-best MIRA algorithm of McDonald et al. (2005) by solving a sequence of *loss-augmented* inference problems.¹³ The number of iterations was set to 10.

The results are summarized in Table 1; for the sake of comparison, we reproduced three strong baselines, all of them state-of-the-art parsers based on non-arc-factored models: the second order model of McDonald and Pereira (2006), the hybrid model of Nivre and McDonald (2008), which combines a (labeled) transition-based and a graph-based parser, and a refinement of the latter, due to Martins et al. (2008), which attempts to approximate non-local features.¹⁴ We did not reproduce the model of Riedel and Clarke (2006) since the latter is tailored for *labeled* dependency parsing; however, experiments reported in that paper for Dutch (and extended to other languages in the CoNLL-X task) suggest that their model performs worse than our three baselines.

By looking at the middle four columns, we can see that adding non-arc-factored features makes

¹¹Note that, unlike reranking approaches, there are still exponentially many candidate parse trees after pruning.

¹²The oracle constrained to pick parents from these lists achieves $> 98\%$ in every case.

¹³The loss-augmented inference problem can also be expressed as an LP for Hamming loss functions that factor over arcs; we refer to Martins et al. (2009) for further details.

¹⁴Unlike our model, the hybrid models used here as baselines make use of the dependency labels at training time; indeed, the transition-based parser is trained to predict a *labeled* dependency parse tree, and the graph-based parser use these predicted labels as input features. Our model ignores this information at training time; therefore, this comparison is slightly unfair to us.

	[MP06]	[NM08]	[MDSX08]	ARC-FACTORED	+SIBL/GRANDP.	+VALENCY	+PROJ. (FULL)	FULL, RELAXED
DANISH	90.60	91.30	91.54	89.80	91.06	90.98	91.18	91.04 (-0.14)
DUTCH	84.11	84.19	84.79	83.55	84.65	84.93	85.57	85.41 (-0.16)
PORTUGUESE	91.40	91.81	92.11	90.66	92.11	92.01	91.42	91.44 (+0.02)
SLOVENE	83.67	85.09	85.13	83.93	85.13	85.45	85.61	85.41 (-0.20)
SWEDISH	89.05	90.54	90.50	89.09	90.50	90.34	90.60	90.52 (-0.08)
TURKISH	75.30	75.68	76.36	75.16	76.20	76.08	76.34	76.32 (-0.02)
ENGLISH	90.85	–	–	90.15	91.13	91.12	91.16	91.14 (-0.02)

Table 1: Results for nonprojective dependency parsing (unlabeled attachment scores). The three baselines are the second order model of McDonald and Pereira (2006), and the hybrid models of Nivre and McDonald (2008) and Martins et al. (2008). The four middle columns show the performance of our model using exact (ILP) inference at test time, for increasing sets of features (see §3.2–§3.5). The rightmost column shows the results obtained with the full set of features using relaxed LP inference followed by projection onto the feasible set. Differences are with respect to exact inference for the same set of features. Bold indicates the best result for a language. As for overall performance, both the exact and relaxed full model outperform the arc-factored model and the second order model of McDonald and Pereira (2006) with statistical significance ($p < 0.01$) according to Dan Bikel’s randomized method (<http://www.cis.upenn.edu/~dbikel/software.html>).

the models more accurate, for all languages. With the exception of Portuguese, the best results are achieved with the full set of features. We can also observe that, for some languages, the valency features do not seem to help. Merely modeling the *number* of dependents of a word may not be as valuable as knowing what *kinds* of dependents they are (for example, distinguishing among arguments and adjuncts).

Comparing with the baselines, we observe that our full model outperforms that of McDonald and Pereira (2006), and is in line with the most accurate dependency parsers (Nivre and McDonald, 2008; Martins et al., 2008), obtained by combining transition-based and graph-based parsers. Notice that our model, compared with these hybrid parsers, has the advantage of not requiring an ensemble configuration (eliminating, for example, the need to tune two parsers). Unlike the ensembles, it directly handles non-local output features by optimizing a single global objective. Perhaps more importantly, it makes it possible to exploit expert knowledge through the form of *hard global constraints*. Although not pursued here, the same kind of constraints employed by Riedel and Clarke (2006) can straightforwardly fit into our model, after extending it to perform labeled dependency parsing. We believe that a careful design of features and constraints can lead to further improvements on accuracy.

We now turn to a different issue: scalability. In previous work (Martins et al., 2009), we showed that training the model via LP-relaxed inference (as we do here) makes it learn to avoid fractional solutions; as a consequence, ILP solvers will converge faster to the optimum (on average). Yet, in the worst-case setting, the problem is still NP-

complete, and may not scale well with the sentence length. Merely considering the LP-relaxed version of the problem at test time is unsatisfactory, as it may lead to a fractional solution (i.e., a solution whose components indexed by arcs, $\tilde{z} = \langle z_a \rangle_{a \in A}$, are not all integer), which does not correspond to a valid dependency tree. We propose the following approximate algorithm to obtain an actual parse: first, solve the LP relaxation (which can be done in polynomial time with interior-point methods); then, if the solution is fractional, project it onto the feasible set $\mathcal{Y}(x)$. Fortunately, the Euclidean projection can be computed in a straightforward way by finding a maximal arborescence in the directed graph whose weights are defined by \tilde{z} (we omit the proof for space); as we saw in §2.2, the Chu-Liu-Edmonds algorithm can do this in polynomial time. The overall parsing runtime becomes polynomial with respect to the length of the sentence.

The last column of Table 1 compares the accuracy of this approximate method with the exact one. We observe that there is not a substantial drop in accuracy; on the other hand, we observed a considerable speed-up with respect to exact inference, particularly for long sentences. The average runtime (across all languages) is 0.632 seconds per sentence, which is in line with existing higher-order parsers and is much faster than the runtimes reported by Riedel and Clarke (2006).

5 Conclusions

We presented new dependency parsers based on concise ILP formulations. We have shown how non-local output features can be incorporated, while keeping only a polynomial number of constraints. These features can act as soft constraints

whose penalty values are automatically learned from data; in addition, our model is also compatible with expert knowledge in the form of hard constraints. Learning through a max-margin framework is made effective by the means of a LP-relaxation. Experimental results on seven languages show that our rich-featured parsers outperform arc-factored and approximate higher-order parsers, and are in line with stacked parsers, having with respect to the latter the advantage of not requiring an ensemble configuration.

Acknowledgments

The authors thank the reviewers for their comments. A.M. was supported by a grant from FCT/ICTI through the CMU-Portugal Program, and also by Priberam Informática. N.S. was supported by NSF IIS-0836431 and an IBM Faculty Award. E.X. was supported by NSF DBI-0546594, DBI-0640543, IIS-0713379, and an Alfred Sloan Foundation Fellowship in Computer Science.

References

- E. Boros and P.L. Hammer. 2002. Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225.
- S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. of CoNLL*.
- X. Carreras, M. Collins, and T. Koo. 2008. TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-rich Parsing. In *Proc. of CONLL*.
- X. Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proc. of CoNLL*.
- M. Chang, L. Ratnoff, and D. Roth. 2008. Constraints as prior knowledge. In *ICML Workshop on Prior Knowledge for Text and Language Processing*, July.
- Y. J. Chu and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.
- J. Clarke and M. Lapata. 2008. Global Inference for Sentence Compression An Integer Linear Programming Approach. *JAIR*, 31:399–429.
- K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. 2006. Online Passive-Aggressive Algorithms. *JMLR*, 7:551–585.
- A. Culotta and J. Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proc. of ACL*.
- P. Denis and J. Baldridge. 2007. Joint Determination of Anaphoricity and Coreference Resolution using Integer Programming. In *Proc. of HLT-NAACL*.
- Y. Ding and M. Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammar. In *Proc. of ACL*.
- J. Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.
- J. Eisner and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proc. of ACL*.
- J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of COLING*.
- S. Kahane, A. Nasr, and O. Rambow. 1998. Pseudo-projectivity: a polynomially parsable non-projective dependency grammar. In *Proc. of COLING*. Association for Computational Linguistics Morristown, NJ, USA.
- S. Lacoste-Julien, B. Taskar, D. Klein, and M.I. Jordan. 2006. Word alignment via quadratic assignment. In *Proc. of HLT-NAACL*.
- T.L. Magnanti and L.A. Wolsey. 1994. Optimal Trees. Technical Report 290-94, Massachusetts Institute of Technology, Operations Research Center.
- A.F.T. Martins, D. Das, N.A. Smith, and E.P. Xing. 2008. Stacking dependency parsers. In *Proc. of EMNLP*.
- A. F. T. Martins, N. A. Smith, and E. P. Xing. 2009. Polyhedral outer approximations with application to natural language parsing. In *Proc. of ICML*.
- R. T. McDonald and F. C. N. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. of EACL*.
- R. McDonald and G. Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proc. of IWPT*.
- R. T. McDonald, F. Pereira, K. Ribarov, and J. Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of HLT-EMNLP*.
- J. Nivre and R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proc. of ACL-HLT*.
- V. Punyakanok, D. Roth, W. Yih, and D. Zimak. 2004. Semantic role labeling via integer linear programming inference. In *Proc. of COLING*.
- M. Richardson and P. Domingos. 2006. Markov logic networks. *Machine Learning*, 62(1):107–136.
- S. Riedel and J. Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proc. of EMNLP*.
- R.T. Rockafellar. 1970. *Convex Analysis*. Princeton University Press.
- D. Roth and W. T. Yih. 2005. Integer linear programming inference for conditional random fields. In *ICML*.
- A. Schrijver. 2003. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer.
- D. A. Smith and J. Eisner. 2008. Dependency parsing by belief propagation. In *Proc. of EMNLP*.
- 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 CoNLL*.

- R.E. Tarjan. 1977. Finding optimum branchings. *Networks*, 7(1):25–36.
- M. Wang, N. A. Smith, and T. Mitamura. 2007. What is the Jeopardy model? A quasi-synchronous grammar for QA. In *Proceedings of EMNLP-CoNLL*.