Dual Decomposition with Many Overlapping Components

André F. T. Martins[*†] **Noah A. Smith**[*] **Pedro M. Q. Aguiar**[‡] **Mário A. T. Figueiredo**[†]
[*]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[‡]Instituto de Sistemas e Robótica, Instituto Superior Técnico, Lisboa, Portugal
[†]Instituto de Telecomunicações, Instituto Superior Técnico, Lisboa, Portugal
{afm,nasmith}@cs.cmu.edu, aguiar@isr.ist.utl.pt, mtf@lx.it.pt

Abstract

Dual decomposition has been recently proposed as a way of combining complementary models, with a boost in predictive power. However, in cases where lightweight decompositions are not readily available (*e.g.*, due to the presence of rich features or logical constraints), the original subgradient algorithm is inefficient. We sidestep that difficulty by adopting an augmented Lagrangian method that accelerates model consensus by regularizing towards the averaged votes. We show how first-order logical constraints can be handled efficiently, even though the corresponding subproblems are no longer combinatorial, and report experiments in dependency parsing, with state-of-the-art results.

1 Introduction

The last years have witnessed increasingly accurate models for syntax, semantics, and machine translation (Chiang, 2007; Finkel et al., 2008; Petrov and Klein, 2008; Smith and Eisner, 2008; Martins et al., 2009a; Johansson and Nugues, 2008; Koo et al., 2010). The predictive power of such models stems from their ability to break locality assumptions. The resulting combinatorial explosion typically demands some form of approximate decoding, such as sampling, heuristic search, or variational inference.

In this paper, we focus on parsers built from linear programming relaxations, the so-called "turbo parsers" (Martins et al., 2009a; Martins et al., 2010). Rush et al. (2010) applied dual decomposition as a way of combining models which alone permit efficient decoding, but whose combination is intractable. This results in a relaxation of the original problem that is elegantly solved with the subgradient algorithm. While this technique has proven quite effective in parsing (Koo et al., 2010; Auli and Lopez, 2011) as well as machine translation (Rush and Collins, 2011), we show here that its success is strongly tied to the ability of finding a "good" decomposition, *i.e.*, one involving few overlapping components (or *slaves*). With many components, the subgradient algorithm exhibits extremely slow convergence (cf. Fig. 2). Unfortunately, a lightweight decomposition is not always at hand, either because the problem does not factor in a natural way, or because one would like to incorporate features that cannot be easily absorbed in few tractable components. Examples include features generated by statements in first-order logic, features that violate Markov assumptions, or history features such as the ones employed in transition-based parsers.

To tackle the kind of problems above, we adopt DD-ADMM (Alg. 1), a recently proposed algorithm that accelerates dual decomposition (Martins et al., 2011). DD-ADMM retains the modularity of the subgradient-based method, but it speeds up consensus by regularizing each slave subproblem towards the averaged votes obtained in the previous round (cf. Eq. 14). While this yields more involved subproblems (with a quadratic term), we show that exact solutions can still be efficiently computed for all cases of interest, by using sort operations. As a result, we obtain parsers that can handle very rich features, do not require specifying a decomposition, and can be heavily parallelized. We demonstrate the success of the approach by presenting experiments in dependency parsing with state-of-the-art results.

2 Background

2.1 Structured Prediction

Let $x \in \mathcal{X}$ be an input object (*e.g.*, a sentence), from which we want to predict a structured output $y \in \mathcal{Y}$ (*e.g.*, a parse tree). The output set $\mathcal{Y}$ is assumed too large for exhaustive search to be tractable. We assume to have a model that assigns a score $f(y)$ to each candidate output, based on which we predict

$$\widehat{y} = \arg\max_{y \in \mathcal{Y}} f(y). \quad (1)$$

Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, pages 238–249,
Edinburgh, Scotland, UK, July 27–31, 2011.

Designing the model must obey certain practical considerations. If efficiency is the major concern, a simple model is usually chosen so that Eq. 1 can be solved efficiently, at the cost of limited expressive power. If we care more about accuracy, a model with richer features and more involved score functions may be designed. Decoding, however, will be more expensive, and approximations are often necessary. A typical source of intractability comes from the combinatorial explosion inherent in the composition of two or more tractable models (Bar-Hillel et al., 1964; Tromble and Eisner, 2006). Recently, Rush et al. (2010) have proposed a dual decomposition framework to address NLP problems in which the global score decomposes as $f(y) = f_1(z_1) + f_2(z_2)$, where z_1 and z_2 are two overlapping "views" of the output, so that Eq. 1 becomes:

$$\begin{array}{ll} \text{maximize} & f_1(z_1) + f_2(z_2) \\ \text{w.r.t.} & z_1 \in \mathcal{Y}_1, z_2 \in \mathcal{Y}_2 \\ \text{s.t.} & z_1 \sim z_2. \end{array} \quad (2)$$

Above, the notation $z_1 \sim z_2$ means that z_1 and z_2 "agree on their overlaps," and an isomorphism $\mathcal{Y} \simeq \{\langle z_1, z_2 \rangle \in \mathcal{Y}_1 \times \mathcal{Y}_2 \mid z_1 \sim z_2\}$ is assumed. We next formalize these notions and proceed to compositions of an *arbitrary* number of models. Of special interest is the unexplored setting where this number is very large and each component very simple.

2.2 Decomposition into Parts

A crucial step in the design of structured predictors is that of decomposing outputs into parts (Taskar et al., 2003). We assume the following setup:

Basic parts. We let $\mathcal{R}$ be a set of *basic parts*, such that each element $y \in \mathcal{Y}$ can be identified with a subset of $\mathcal{R}$. The exact meaning of a "basic part" is problem dependent. For example, in dependency parsing, $\mathcal{R}$ can be the set of all possible dependency arcs (see Fig. 1); in phrase-based parsing, it can be the set of possible spans; in sequence labeling, it can be the set of possible labels at each position. Our only assumption is that we can "read out" y from the basic parts it contains. For convenience, we represent y as a binary vector, $y = \langle y(r) \rangle_{r \in \mathcal{R}}$, where $y(r) = 1$ if part r belongs to y, and 0 otherwise.

Decomposition. We generalize the decomposition in Eq. 2 by considering sets $\mathcal{Y}_1, \ldots, \mathcal{Y}_S$ for $S \geq 2$.

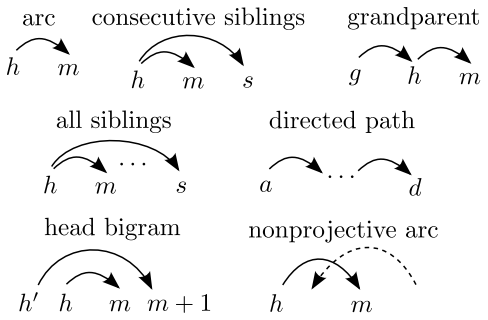

Figure 1: Parts used by our parser. *Arcs* are the basic parts: any dependency tree can be "read out" from the arcs it contains. *Consecutive siblings* and *grandparent* parts introduce horizontal and vertical Markovization (McDonald et al., 2006; Carreras, 2007). We break the horizontal Markov assumption via *all siblings* parts and the vertical one through parts which indicate a *directed path* between two words. Inspired by transition-based parsers, we also adopt *head bigram* parts, which look at the heads attached to consecutive words. Finally, we follow Martins et al. (2009a) and have parts which indicate if an arc is *non-projective* (*i.e.*, if it spans words that do not descend from its head).

Each $\mathcal{Y}_s$ is associated with its own set of parts $\mathcal{R}_s$, in the same sense as above; we represent the elements of $\mathcal{Y}_s$ as binary vectors $z_s = \langle z_s(r) \rangle_{r \in \mathcal{R}_s}$. Examples are vectors indicating a tree structure, a sequence, or an assignment of variables to a factor, in which case it may happen that only some binary vectors are legal. Some parts in $\mathcal{R}_s$ are basic, while others are not. We denote by $\bar{\mathcal{R}}_s = \mathcal{R}_s \cap \mathcal{R}$ the subset of the ones that are. In addition, we assume that:

- $\mathcal{R}_1, \ldots, \mathcal{R}_S$ jointly cover $\mathcal{R}$, *i.e.*, $\mathcal{R} \subseteq \bigcup_{s=1}^{S} \mathcal{R}_s$;
- Only basic parts may overlap, *i.e.*, $\mathcal{R}_s \cap \mathcal{R}_t \subseteq \mathcal{R}, \forall s, t \in \{1, \ldots, S\}$;
- Each $z_s \in \mathcal{Y}_s$ is completely defined by its entries indexed by elements of $\bar{\mathcal{R}}_s$, from which we can guess the ones in $\mathcal{R}_s \setminus \bar{\mathcal{R}}_s$. This implies that each $y \in \mathcal{Y}$ has a unique decomposition $\langle z_1, \ldots, z_S \rangle$.

Fig. 1 shows several parts used in dependency parsing models; in phrase-based parsing, these could be spans and production rules anchored in the surface string; in sequence labeling, they can be unigram, bigram, and trigram labels.[1]

[1]There is a lot of flexibility about how to decompose the model into S components: each set $\mathcal{R}_s$ can correspond to a sin-

Global consistency. We want to be able to read out $y \in \mathcal{Y}$ by "gluing" together the components $\langle z_1, \ldots, z_S \rangle$. This is only meaningful if they are "globally consistent," a notion which we make precise. Two components $z_s \in \mathcal{Y}_s$ and $z_t \in \mathcal{Y}_t$ are said to be *consistent* (denoted $z_s \sim z_t$) if they agree on their overlaps, *i.e.*, if $z_s(r) = z_t(r), \forall r \in \mathcal{R}_s \cap \mathcal{R}_t$. A complete assignment $\langle z_1, \ldots, z_S \rangle$ is *globally consistent* if all pairs of components are consistent. This is equivalent to the existence of a witness vector $\langle u(r) \rangle_{r \in \mathcal{R}}$ such that $z_s(r) = u(r), \forall s, r \in \bar{\mathcal{R}}_s$.

With this setup, assuming that the score function decomposes as $f(z) = \sum_{s=1}^{S} f_s(z_s)$, the decoding problem (which extends Eq. 2 for $S \geq 2$) becomes:

$$\begin{array}{lll} P: & \text{maximize} & \sum_{s=1}^{S} f_s(z_s) \\ & \text{w.r.t.} & z_s \in \mathcal{Y}_s, \quad \forall s \\ & & \langle u(r) \rangle_{r \in \mathcal{R}} \in \mathbb{R}^{|\mathcal{R}|}, \\ & \text{s.t.} & z_s(r) = u(r), \quad \forall s, r \in \bar{\mathcal{R}}_s. \end{array} \tag{3}$$

We call the equality constraints expressed in the last line the "agreement constraints." It is these constraints that complicate the problem, which would otherwise be exactly separable into S subproblems. The dual decomposition method (Komodakis et al., 2007; Rush et al., 2010) builds an approximation by dualizing out these constraints, as we describe next.

2.3 Dual Decomposition

We describe dual decomposition in a slightly different manner than Rush et al. (2010): we will first build a relaxation of P (called P'), in which the entire approximation is enclosed. Then, we dualize P', yielding problem D. In the second step, the duality gap is zero, *i.e.*, P' and D are equivalent.[2]

Relaxation. For each $s \in \{1, \ldots, S\}$ we consider the convex hull of $\mathcal{Y}_s$,

$$\mathcal{Z}_s = \left\{ \sum_{z_s \in \mathcal{Y}_s} p(z_s) z_s \;\middle|\; p(z_s) \geq 0, \sum_{z_s \in \mathcal{Y}_s} p(z_s) = 1 \right\}. \tag{4}$$

We have that $\mathcal{Y}_s = \mathcal{Z}_s \cap \mathbb{Z}^{|\mathcal{R}_s|}$; hence, problem P (Eq. 3) is equivalent to one in which each $\mathcal{Y}_s$ is replaced by $\mathcal{Z}_s$ and the z-variables are constrained to be integer. By dropping the integer constraints, we obtain the following relaxed problem:

$$\begin{array}{lll} P': & \text{maximize} & \sum_{s=1}^{S} f_s(z_s) \\ & \text{w.r.t.} & z_s \in \mathcal{Z}_s, \quad \forall s \\ & & \langle u(r) \rangle_{r \in \mathcal{R}} \in \mathbb{R}^{|\mathcal{R}|}, \\ & \text{s.t.} & z_s(r) = u(r), \quad \forall s, r \in \bar{\mathcal{R}}_s. \end{array} \tag{5}$$

If the score functions f_s are convex, P' becomes a convex program (unlike P, which is discrete); being a relaxation, it provides an upper bound of P.

Lagrangian. Introducing a Lagrange multiplier $\lambda_s(r)$ for each agreement constraint in Eq. 5, one obtains the Lagrangian function

$$\begin{aligned} \mathcal{L}(z, u, \lambda) = & \sum_{s=1}^{S} \left(f_s(z_s) + \sum_{r \in \bar{\mathcal{R}}_s} \lambda_s(r) z_s(r) \right) \\ & - \sum_{r \in \mathcal{R}} \left(\sum_{s: r \in \bar{\mathcal{R}}_s} \lambda_s(r) \right) u(r), \end{aligned} \tag{6}$$

and the dual problem (the *master*)

$$\begin{array}{lll} D: & \text{minimize} & \sum_{s=1}^{S} g_s(\lambda_s) \\ & \text{w.r.t.} & \lambda = \langle \lambda_1, \ldots, \lambda_S \rangle \\ & \text{s.t.} & \sum_{s: r \in \bar{\mathcal{R}}_s} \lambda_s(r) = 0, \ \forall r \in \mathcal{R}, \end{array} \tag{7}$$

where the $g_s(\lambda_s)$ are the solution values of the following subproblems (the *slaves*):

$$\begin{array}{ll} \text{maximize} & f_s(z_s) + \sum_{r \in \bar{\mathcal{R}}_s} \lambda_s(r) z_s(r) \\ \text{w.r.t.} & z_s \in \mathcal{Z}_s. \end{array} \tag{8}$$

We assume that strong duality holds (w.r.t. Eqs. 5–7), hence we have $P \leq P' = D$.[3]

Solving the dual. Why is the dual formulation D (Eqs. 7–8) more appealing than P' (Eq. 5)? The answer is that the components $1, \ldots, S$ are now decoupled, which makes things easier provided each slave subproblem (Eq. 8) can be solved efficiently. In fact, this is always a concern in the mind of the model's designer when she chooses a decomposition (the framework that we describe in §3, in some sense, alleviates her from this concern). If the score functions are linear, *i.e.*, of the form $f_s(z_s) = \sum_{r \in \mathcal{R}_s} \theta_s(r) z_s(r)$ for some vector $\theta_s = \langle \theta_s(r) \rangle_{r \in \mathcal{R}_s}$, then Eq. 8 becomes a linear program, for which a solution exists at a vertex of $\mathcal{Z}_s$ (which

gle factor in a factor graph (Smith and Eisner, 2008), or to a entire subgraph enclosing several factors (Koo et al., 2010), or even to a formula in Markov logic (Richardson and Domingos, 2006). In these examples, the basic parts may correspond to individual variable-value pairs.

[2] Instead of following the path $P \to P' \to D$, Rush et al. (2010) go straight from P to D via a Lagrangian relaxation. The two formulations are equivalent for linear score functions.

[3] This is guaranteed if the score functions f_s are linear.

in turn is an element of $\mathcal{Y}_s$). Depending on the structure of the problem, Eq. 8 may be solved by brute force, dynamic programming, or specialized combinatorial algorithms (Rush et al., 2010; Koo et al., 2010; Rush and Collins, 2011).

Applying the projected subgradient method (Komodakis et al., 2007; Rush et al., 2010) to the master problem (Eq. 7) yields a remarkably simple algorithm, which at each round t solves the subproblems in Eq. 8 for $s = 1, \ldots, S$, and then gathers these solutions (call them z_s^{t+1}) to compute an "averaged" vote for each basic part,

$$u^{t+1}(r) = \frac{1}{\delta(r)} \sum_{s:r \in \bar{\mathcal{R}}_s} z_s^{t+1}(r), \quad (9)$$

where $\delta(r) = |\{s : r \in \mathcal{R}_s\}|$ is the number of components which contain part r. An update of the Lagrange variables follows,

$$\lambda_s^{t+1}(r) = \lambda_s^t(r) - \eta_t(z_s^{t+1}(r) - u^{t+1}(r)), \quad (10)$$

where η_t is a stepsize. Intuitively, the algorithm pushes for a consensus among the slaves (Eq. 9), via an adjustment of the Lagrange multipliers which takes into consideration deviations from the average (Eq. 10). The subgradient method is guaranteed to converge to the solution of D (Eq. 7), for suitably chosen stepsizes (Shor, 1985; Bertsekas et al., 1999); it also provides a certificate of optimality in case the relaxation is tight (*i.e.*, $P = D$) and the *exact* solution has been found. However, convergence is slow when S is large (as we will show in the experimental section), and no certificates are available when there is a relaxation gap ($P < P'$). In the next section, we describe the DD-ADMM algorithm (Martins et al., 2011), which does not have these drawbacks and shares a similar simplicity.

3 Alternating Directions Method

There are two reasons why subgradient-based dual decomposition is not completely satisfying:

- it may take a long time to reach a consensus;
- it puts all its resources in solving the dual problem D, and does not attempt to make progress in the primal P', which is closer to our main concern.[4]

[4]Our main concern is P; however solving P' is often a useful step towards that goal, either because a good rounding scheme exists, or because one may build tighter relaxations to approach P (Sontag et al., 2008; Rush and Collins, 2011).

Taking a look back at the relaxed primal problem P' (Eq. 5), we see that any primal feasible solution must satisfy the agreement constraints. This suggests that penalizing violations of these constraints could speed up consensus.

Augmented Lagrangian. By adding a penalty term to Eq. 6, we obtain the *augmented Lagrangian function* (Hestenes, 1969; Powell, 1969):

$$\mathcal{A}_\rho(z, u, \lambda) = \mathcal{L}(z, u, \lambda) - \frac{\rho}{2} \sum_{s=1}^{S} \sum_{r \in \bar{\mathcal{R}}_s} (z_s(r) - u(r))^2, \quad (11)$$

where the parameter $\rho \geq 0$ controls the intensity of the penalty. Augmented Lagrangian methods are well-known in the optimization community (see, *e.g.*, Bertsekas et al. (1999), §4.2). They alternate updates to the λ-variables, while seeking to maximize $\mathcal{A}_\rho$ with respect to z and u. In our case, however, this joint maximization poses difficulties, since the penalty term couples the two variables. The *alternating directions method of multipliers* (ADMM), coined by Gabay and Mercier (1976) and Glowinski and Marroco (1975), sidesteps this issue by performing alternate maximizations,

$$z^{t+1} = \arg\max_z \mathcal{A}_\rho(z, u^t, \lambda^t), \quad (12)$$

$$u^{t+1} = \arg\max_u \mathcal{A}_\rho(z^{t+1}, u, \lambda^t), \quad (13)$$

followed by an update of the Lagrange multipliers as in Eq. 10. Recently, ADMM has attracted interest, being applied in a variety of problems; see the recent book by Boyd et al. (2011) for an overview. As derived in the App. A, the u-updates in Eq. 13 have a closed form, which is precisely the averaging operation performed by the subgradient method (Eq. 9). We are left with the problem of computing the z-updates. Like in the subgradient approach, the maximization in Eq. 12 can be separated into S independent slave subproblems, which now take the form:

$$\begin{array}{ll} \text{maximize} & f_s(z_s) + \sum_{r \in \bar{\mathcal{R}}_s} \lambda_s(r) z_s(r) \\ & -\frac{\rho}{2} \sum_{r \in \bar{\mathcal{R}}_s} (z_s(r) - u^t(r))^2 \\ \text{w.r.t.} & z_s \in \mathcal{Z}_s(x). \end{array} \quad (14)$$

Comparing Eq. 8 and Eq. 14, we observe that the only difference is the presence in the latter of a

quadratic term which regularizes towards the previous averaged votes $u^t(r)$. Because of this term, the solution of Eq. 14 for linear score functions may not be at a vertex (in contrast to the subgradient method). We devote §4 to describing exact and efficient ways of solving the problem in Eq. 14 for important, widely used slaves. Before going into details, we mention another advantage of ADMM over the subgradient algorithm: it knows when to stop.

Primal and dual residuals. Recall that the subgradient method provides optimality certificates when the relaxation is tight ($P = P'$) and an exact solution of P has been found. While this is good enough when tight relaxations are frequent, as in the settings explored by Rush et al. (2010), Koo et al. (2010), and Rush and Collins (2011), it is hard to know when to stop when a relaxation gap exists. We would like to have similar guarantees concerning the relaxed primal P'.[5] A general weakness of subgradient algorithms is that they do not have this capacity, and so are usually stopped by specifying a maximum number of iterations. In contrast, ADMM allows to keep track of primal and dual residuals (Boyd et al., 2011). This allows providing certificates not only for the exact solution of P (when the relaxation is tight), but also to terminate when a near optimal solution of the relaxed problem P' has been found. The *primal residual* r_P^t measures the amount by which the agreement constraints are violated:

$$r_P^t = \frac{\sum_{s=1}^{S} \sum_{r \in \bar{\mathcal{R}}_s} (z_s^t(r) - u^t(r))^2}{\sum_{r \in \mathcal{R}} \delta(r)}; \qquad (15)$$

the *dual residual* r_D^t is the amount by which a dual optimality condition is violated (see Boyd et al. (2011), p.18, for details). It is computed via:

$$r_D^t = \frac{\sum_{r \in \mathcal{R}} \delta(r)(u^t(r) - u^{t-1}(r))^2}{\sum_{r \in \mathcal{R}} \delta(r)}, \qquad (16)$$

Our stopping criterion is thus that these two residuals are below a threshold, *e.g.*, 1×10^{-3}. The complete algorithm is depicted as Alg. 1. As stated in Martins et al. (2011), convergence to the solution of P' is guaranteed with a fixed stepsize $\eta_t = \tau\rho$, with $\tau \in [1, 1.618]$ (Glowinski and Le Tallec, 1989, Thm. 4.2). In our experiments, we set $\tau = 1.5$, and adapt ρ as described in (Boyd et al., 2011, p.20).[6]

Algorithm 1 ADMM-based Dual Decomposition

input: score functions $\langle f_s(.)\rangle_{s=1}^S$, parameters ρ, η, thresholds ϵ_P and ϵ_D.
initialize $t \leftarrow 1$
initialize $u^1(r) \leftarrow 0.5$ and $\lambda_s^1(r) \leftarrow 0, \quad \forall s, \forall r \in \bar{\mathcal{R}}_s$
repeat
  for each $s = 1, \ldots, S$ **do**
    make a z_s-update, yielding z_s^{t+1} (Eq. 14)
  end for
  make a u-update, yielding u^{t+1} (Eq. 9)
  make a λ-update, yielding λ^{t+1} (Eq. 10)
  $t \leftarrow t + 1$
until $r_P^{t+1} < \epsilon_P$ and $r_D^{t+1} < \epsilon_D$ (Eqs. 15–16)
output: relaxed primal and dual solutions u, z, λ

4 Solving the Subproblems

In this section, we address the slave subproblems of DD-ADMM (Eq. 14). We show how these subproblems can be solved efficiently for several important cases that arise in NLP applications. Throughout, we assume that the score functions f_s are linear, *i.e.*, they can be written as $f_s(z_s) = \sum_{r \in \mathcal{R}_s} \theta_s(r) z_s(r)$. This is the case whenever a linear model is used, in which case $\theta_s(r) = \frac{1}{\delta(r)} \boldsymbol{w} \cdot \boldsymbol{\phi}(x, r)$, where $\boldsymbol{w}$ is a weight vector and $\boldsymbol{\phi}(x, r)$ is a feature vector. It is also the scenario studied in previous work in dual decomposition (Rush et al., 2010). Under this assumption, and discarding constant terms, the slave subproblem in Eq. 14 becomes:

$$\max_{z_s \in \mathcal{Z}_s} \sum_{r \in \mathcal{R}_s \setminus \bar{\mathcal{R}}_s} \theta_s(r) z_s(r) - \frac{\rho}{2} \sum_{r \in \bar{\mathcal{R}}_s} (z_s(r) - a_s(r))^2. \qquad (17)$$

where $a_s(r) = u^t(r) + \rho^{-1}(\theta_s(r) + \lambda_s^t(r))$. Since $\mathcal{Z}_s$ is a polytope, Eq. 17 is a quadratic program, which can be solved with a general purpose solver. However, that does not exploit the structure of $\mathcal{Z}_s$ and is inefficient when $|\mathcal{R}_s|$ is large. We next show that for many cases, a closed-form solution is available and

[5] This problem is more important than it may look. Problems with many slaves tend to be less exact, hence relaxation gaps are frequent. Also, when decoding is embedded in training, it is useful to obtain the *fractional* solution of the relaxed primal P (rather than an approximate integer solution). See Kulesza and Pereira (2007) and Martins et al. (2009b) for details.

[6] Briefly, we initialize $\rho = 0.03$ and then increase/decrease ρ by a factor of 2 whenever the primal residual becomes > 10 times larger/smaller than the dual residual.

can be computed in $O(|\mathcal{R}_s|)$ time, up to log factors.[7]

Pairwise Factors. This is the case where $\mathcal{R}_{\text{PAIR}} = \{r_1, r_2, r_{12}\}$, where r_1 and r_2 are basic parts and r_{12} is their conjunction, *i.e.*, we have $\mathcal{Y}_{\text{PAIR}} = \{\langle z_1, z_2, z_{12}\rangle \mid z_{12} = z_1 \wedge z_2\}$. This factor is useful to make conjunctions of variables participate in the score function (see *e.g.* the grandparent, sibling, and head bigram parts in Fig. 1). The convex hull of $\mathcal{Y}_{\text{PAIR}}$ is the polytope $\mathcal{Z}_{\text{PAIR}} = \{\langle z_1, z_2, z_{12}\rangle \in [0,1]^3 \mid z_{12} \leq z_1,\ z_{12} \leq z_2, z_{12} \geq z_1 + z_2 - 1\}$, as shown by Martins et al. (2010). In this case, problem (17) can be written as

$$\begin{array}{ll} \max & \theta_{12} z_{12} - \frac{\rho}{2}[(z_1 - a_1)^2 + (z_2 - a_2)^2] \\ \text{w.r.t.} & \langle z_1, z_2, z_{12}\rangle \in [0,1]^3 \\ \text{s.t.} & z_{12} \leq z_1,\ z_{12} \leq z_2,\ z_{12} \geq z_1 + z_2 - 1 \end{array} \tag{18}$$

and has a closed form solution (see App. B).

Uniqueness Quantification and XOR. Many problems involve constraining variables to take a single value: for example, in dependency parsing, a modifier can only take one head. This can be expressed as the statement $\exists! y : Q(y)$ in first-order logic,[8] or as a one-hot XOR factor in a factor graph (Smith and Eisner, 2008; Martins et al., 2010). In this case, $\mathcal{R}_{\text{XOR}} = \{r_1, \ldots, r_n\}$, and $\mathcal{Y}_{\text{XOR}} = \{\langle z_1, \ldots, z_n\rangle \in \{0,1\}^n \mid \sum_{i=1}^n z_i = 1\}$. The convex hull of $\mathcal{Y}_{\text{XOR}}$ is $\mathcal{Z}_{\text{XOR}} = \{\langle z_1, \ldots, z_n\rangle \in [0,1]^n \mid \sum_{i=1}^n z_i = 1\}$. Assume for the sake of simplicity that all parts in $\mathcal{R}_{\text{XOR}}$ are basic.[9] Up to a constant, the slave subproblem becomes:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\sum_{i=1}^n (z_i - a_i)^2 \\ \text{w.r.t.} & \langle z_1, \ldots, z_n\rangle \in [0,1]^n \\ \text{s.t.} & \sum_i^n z_i = 1. \end{array} \tag{19}$$

This is the problem of projecting onto the probability simplex, which can be done in $O(n \log n)$ time via a sort operation (see App. C).[10]

[7] This matches the asymptotic time that would be necessary to solve the corresponding problems in the subgradient method, for which algorithms are straightforward to derive. The point is that with ADMM fewer instances of these subproblems need to be solved, due to faster convergence of the master problem.

[8] The symbol $\exists!$ means "there is one and only one."

[9] A similar derivation can be made otherwise.

[10] Also common is the need for constraining existence of "at most one" element. This can be reduced to uniqueness quantification by adding a dummy NULL label.

Existential Quantification and OR. Sometimes, only existence is required, not necessarily uniqueness. This can be expressed with disjunctions, existential quantifiers in first-order logic ($\exists y : Q(y)$), or as a OR factor. In this case, $\mathcal{R}_{\text{OR}} = \{r_1, \ldots, r_n\}$, $\mathcal{Y}_{\text{OR}} = \{\langle z_1, \ldots, z_n\rangle \in \{0,1\}^n \mid \bigvee_{i=1}^n z_i = 1\}$, and the convex hull is $\mathcal{Z}_{\text{OR}} = \{\langle z_1, \ldots, z_n\rangle \in [0,1]^n \mid \sum_{i=1}^n z_i \geq 1\}$ (see Tab. 1 in Martins et al. (2010)). The slave subproblem becomes:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\sum_{i=1}^n (z_i - a_i)^2 \\ \text{w.r.t.} & \langle z_1, \ldots, z_n\rangle \in [0,1]^n \\ \text{s.t.} & \sum_i^n z_i \geq 1. \end{array} \tag{20}$$

We derive a procedure in App. D to compute this projection in $O(n \log n)$ runtime, also with a sort.

Negations. The two cases above can be extended to allow some of their inputs to be *negated*. By a change of variables in Eqs. 19–20 it is possible to reuse the same black box that solves those problems. The procedure is as follows:

1. For $i = 1, \ldots, n$, set $a'_i = 1 - a_i$ if the ith variable is negated, and $a'_i = a_i$ otherwise.
2. Obtain $\langle z'_1, \ldots, z'_n\rangle$ as the solution of Eqs. 19 or 20 providing $\langle a'_1, \ldots, a'_n\rangle$ as input.
3. For $i = 1, \ldots, n$, set $z_i = 1 - z'_i$ if the ith variable is negated, and $z_i = z'_i$ otherwise.

The ability to handle negated variables adds a great degree of flexibility. From De Morgan's laws, we can now handle conjunctions and implications (since $\bigwedge_{i=1}^n Q_i(x) \Rightarrow R(x)$ is equivalent to $\bigvee_{i=1}^n \neg Q_i(x) \vee R(x)$).

Logical Variable Assignments. All previous examples involve taking a group of existing variables and defining a constraint. Alternatively, we may want to define a new variable which is the result of an operation involving other variables. For example, $R(x) := \exists! y : Q(x, y)$. This corresponds to the XOR-WITH-OUTPUT factor in Martins et al. (2010). Interestingly, this can be expressed as a XOR where $R(x)$ is negated (*i.e.*, either $\neg R(x)$ holds or exactly one y satisfies $Q(x, y)$, but not both).

A more difficult problem is that of the OR-WITH-OUTPUT factor, expressed by the formula $R(x) := \exists y : Q(x, y)$. We have $\mathcal{R}_{\text{OR-OUT}} = \{r_0, \ldots, r_n\}$, and $\mathcal{Y}_{\text{OR-OUT}} = \{\langle z_0, \ldots, z_n\rangle \in \{0,1\}^n \mid z_0 =$

		# Slaves	Runtime	Description
Tree	$\exists! h : \texttt{arc}(h, m), \quad m \neq 0$	$O(n)$	$O(n \log n)$	Each non-root word has a head
	$\texttt{flow}(h, m, k) \Rightarrow \texttt{arc}(h, m)$	$O(n^3)$	$O(1)$	Only active arcs may carry flow
	$\texttt{path}(m, d) := \exists! h : \texttt{flow}(h, m, d), \quad m \neq 0$	$O(n^2)$	$O(n \log n)$	
	$\texttt{path}(h, d) := \exists! m : \texttt{flow}(h, m, d)$	$O(n^2)$	$O(n \log n)$	Paths and flows are consistent
	$\texttt{path}(0, m) := \text{TRUE}, \quad \texttt{flow}(h, m, m) := \text{TRUE}$			(see Martins et al. (2010))
All siblings	$\texttt{sibl}(h, m, s) := \texttt{arc}(h, m) \wedge \texttt{arc}(h, s)$	$O(n^3)$	$O(1)$	By definition
Grandp.	$\texttt{grand}(g, h, m) := \texttt{arc}(g, h) \wedge \texttt{arc}(h, m)$	$O(n^3)$	$O(1)$	By definition
Head Bigram	$\texttt{bigram}(b, h, m) := \texttt{arc}(b, m-1) \wedge \texttt{arc}(h, m), \quad m \neq 0$	$O(n^3)$	$O(1)$	By definition
Consec. Sibl.	$\texttt{lastsibl}(h, m, m) := \texttt{arc}(h, m)$			
	$\exists! m \in [h, k] : \texttt{lastsibl}(h, m, k)$	$O(n^2)$	$O(n \log n)$	Head automaton model
	$\texttt{lastsibl}(h, m, k) := \texttt{lastsibl}(h, m, k+1)$			(see supplementary material)
	$\quad \oplus \texttt{nextsibl}(h, m, k+1)$	$O(n^3)$	$O(1)$	
	$\texttt{arc}(h, m) := \exists! s \in [h, m] : \texttt{nextsibl}(h, s, m)$	$O(n^2)$	$O(n \log n)$	
Nonproj. Arc	$\texttt{nonproj}(h, m) := \texttt{arc}(h, m) \wedge \exists k \in [h, m] : \neg \texttt{path}(h, k)$	$O(n^2)$	$O(n \log n)$	By definition

Table 1: First-order logic formulae underlying our dependency parser. The basic parts are the predicate variables $\texttt{arc}(h, m)$ (indicating an arc linking head h to modifier m), $\texttt{path}(a, d)$ (indicating a directed path from ancestor a to descendant d), $\texttt{nextsibl}(h, m, s)$ (indicating that $\langle h, m \rangle$ and $\langle h, s \rangle$ are consecutive siblings), $\texttt{nonproj}(h, m)$ (indicating that $\langle h, m \rangle$ is a non-projective arc), as well as the auxiliary variables $\texttt{flow}(h, m, d)$ (indicating that arc $\langle h, m \rangle$ carries flow to d), and $\texttt{lastsibl}(h, m, k)$ (indicating that, up to position k, the last seen modifier of h occurred at position m). The non-basic parts are the pairwise factors $\texttt{sibl}(h, m, s)$, $\texttt{grand}(g, h, m)$, and $\texttt{bigram}(b, h, m)$; as well as each logical formula. Columns 3–4 indicate the number of parts of each kind, and the time complexity for solving each subproblem. For a sentence of length n, there are $O(n^3)$ parts and the total complexity is $O(n^3 \log n)$.

$\bigvee_{i=1}^n z_i\}$. The convex hull of $\mathcal{Y}_{\text{OR-OUT}}$ is the following set: $\mathcal{Z}_{\text{OR-OUT}} = \{\langle z_0, \ldots, z_n \rangle \in [0, 1]^n \mid z_0 \geq \sum_{i=1}^n z_i, z_0 \leq z_i, \ \forall i = 1, \ldots, n\}$ (Martins et al., 2010, Tab.1). The slave subproblem is:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} \sum_{i=0}^n (z_i - a_i)^2 \\ \text{w.r.t.} & \langle z_0, \ldots, z_n \rangle \in [0, 1]^n \\ \text{s.t.} & z_0 \geq \sum_{i=1}^n z_i; \ z_0 \leq z_i, \ \forall i = 1, \ldots, n. \end{array} \tag{21}$$

The problem in Eq. 21 is more involved than the ones in Eqs. 19–20. Yet, there is still an efficient procedure with runtime $O(n \log n)$ (see App. E). By using the result above for negated variables, we are now endowed with a procedure for many other cases, such that AND-WITH-OUTPUT and formulas with universal quantifiers (*e.g.*, $R(x) := \forall y : Q(x, y)$). Up to a log-factor, the runtimes will be linear in the number of predicates.

Larger Slaves. The only disadvantage of DD-ADMM in comparison with the subgradient algorithm is that there is not an obvious way of solving the subproblem in Eq. 14 exactly for large combinatorial factors, such as the TREE constraint in dependency parsing, or a sequence model. Hence, our method seems to be more suitable for decompositions which involve "simple slaves," even if their number is large. However, this does not rule out the possibility of using this method otherwise. Eckstein and Bertsekas (1992) show that the ADMM algorithm may still converge when the z-updates are inexact. Hence the method may still work if the slaves are solved numerically up to some accuracy. We defer this to future investigation.

5 Experiments: Dependency Parsing

We used 14 datasets with non-projective dependencies from the CoNLL-2006 and CoNLL-2008 shared tasks (Buchholz and Marsi, 2006; Surdeanu et al., 2008). We also used a projective English dataset derived from the Penn Treebank by applying the standard head rules of Yamada and Matsumoto (2003).[11] We did not force the parser to output projective trees or unique roots for any of the datasets; everything is learned from the data. We trained by running 10 iterations of the cost-augmented MIRA algorithm (Crammer et al., 2006) with LP-relaxed decoding, as in Martins et al. (2009b). Following common practice (Charniak and Johnson, 2005; Carreras et al., 2008), we employed a coarse-to-fine procedure to prune away unlikely candidate arcs, as described by Koo and Collins (2010). To ensure valid parse trees at test time, we rounded fractional

[11] As usual, we train on sections §02–21, use §22 as validation data, and test on §23. We ran SVMTool (Giménez and Marquez, 2004) to obtain automatic part-of-speech tags for §22–23.

solutions as described in Martins et al. (2009a) (yet, solutions were integral most of the time).

The parts used in our full model are the ones depicted in Fig. 1. Note that a subgradient-based method could handle some of those parts efficiently (*arcs*, *consecutive siblings*, *grandparents*, and *head bigrams*) by composing arc-factored models, head automata, and a sequence labeler. However, no lightweight decomposition seems possible for incorporating parts for *all siblings*, *directed paths*, and *non-projective arcs*. Tab. 1 shows the first-order logical formulae that encode the constraints in our model. Each formula gives rise to a subproblem which is efficiently solvable (see §4). By ablating some of rows of Tab. 1 we recover known methods:

- Resorting to the *tree* and *consecutive sibling* formulae gives one of the models in Koo et al. (2010), with the *same* linear relaxation (a proof of this fact is included in App. F);
- Resorting to *tree*, *all siblings*, *grandparent*, and *non-projective arcs*, recovers a multi-commodity flow configuration proposed by Martins et al. (2009a); the relaxation is also the same.[12]

The experimental results are shown in Tab. 2. For comparison, we include the best published results for each dataset (at the best of our knowledge), among transition-based parsers (Nivre et al., 2006; Huang and Sagae, 2010), graph-based parsers (McDonald et al., 2006; Koo and Collins, 2010), hybrid methods (Nivre and McDonald, 2008; Martins et al., 2008), and turbo parsers (Martins et al., 2010; Koo et al., 2010). Our full model achieved the best reported scores for 7 datasets. The last two columns show a consistent improvement (with the exceptions of Chinese and Arabic) when using the full set of features over a second order model with grandparent and consecutive siblings, which is our reproduction of the model of Koo et al. (2010).[13]

[12] Although Martins et al. (2009a) also incorporated consecutive siblings in one of their configurations, our constraints are tighter than theirs. See App. F.

[13] Note however that the actual results of Koo et al. (2010) are higher than our reproduction, as can be seen in the second column. The differences are due to the features that were used and on the way the models were trained. The cause is not search error: exact decoding with an ILP solver (CPLEX) revealed no significant difference with respect to our G+CS column. We leave further analysis for future work.

	Best known UAS		G+CS	Full
Arabic	80.18	[Ma08]	**81.12**	81.10 (-0.02)
Bulgar.	92.88	[Ma10]	93.04	**93.50** (+0.46)
Chinese	**91.89**	[Ma10]	91.05	90.62 (-0.43)
Czech	88.78	[Ma10]	88.80	**89.46** (+0.66)
English	92.57	[Ko10]	92.45	**92.68** (+0.23)
Danish	91.78	[Ko10]	91.70	**91.86** (+0.16)
Dutch	**85.81**	[Ko10]	84.77	85.53 (+0.76)
German	91.49	[Ma10]	91.29	**91.89** (+0.60)
Japane.	93.42	[Ma10]	93.62	**93.72** (+0.10)
Portug.	**93.03**	[Ko10]	92.05	92.29 (+0.24)
Slovene	86.21	[Ko10]	86.09	**86.95** (+0.86)
Spanish	**87.04**	[Ma10]	85.99	86.74 (+0.75)
Swedish	**91.36**	[Ko10]	89.94	90.16 (+0.22)
Turkish	**77.55**	[Ko10]	76.24	76.64 (+0.40)
PTB §23	**93.04**	[KC10]	92.19	92.53 (+0.34)

Table 2: Unlabeled attachment scores, excluding punctuation. In the second column, [Ma08] denotes Martins et al. (2008), [KC10] is Koo and Collins (2010), [Ma10] is Martins et al. (2010), and [Ko10] is Koo et al. (2010). In columns 3–4, "Full" is our full model, and "G+CS" is our reproduction of the model of Koo et al. (2010), *i.e.*, the same as "Full" but with all features ablated excepted for grandparents and consecutive siblings.

	AF	+G+CS	+AS	+NP	Full
PTB §22	91.02	92.13	92.32	92.36	92.41
PTB §23	91.36	92.19	92.41	92.50	92.53

Table 3: Feature ablation experiments. AF is an arc-factored model; +G+CS adds grandparent and consecutive siblings; +AS adds all-siblings; +NP adds non-projective arcs; Full adds the bigram and directed paths.

Feature ablation and error analysis. We conducted a simple ablation study by training several models on the English PTB with different sets of features. Tab. 3 shows the results. As expected, performance keeps increasing as we use models with greater expressive power. We show some concrete examples in App. G of sentences that the full model parsed correctly, unlike less expressive models.

Convergence speed and optimality. Fig. 2 compares the performance of DD-ADMM and the subgradient algorithms in the validation section of the PTB.[14] For the second order model, the subgradient

[14] The learning rate in the subgradient method was set as $\eta_t = \eta_0/(1+N_{\text{incr}}(t))$, as in Koo et al. (2010), where $N_{\text{incr}}(t)$ is the number of dual increases up to the tth iteration, and η_0 is chosen to maximize dual decrease after 20 iterations (in a per sentence basis). Those preliminary iterations are not plotted in Fig. 2.

method has more slaves than in Koo et al. (2010): it has a slave imposing the TREE constraint (whose subproblems consists on finding a minimum spanning tree) and several for the all-sibling parts, yielding an average number of 310.5 and a maximum of 4310 slaves. These numbers are still manageable, and we observe that a "good" UAS is achieved relatively quickly. The ADMM method has many more slaves due to the multicommodity flow constraints (average 1870.8, maximum 65446), yet it attains optimality sooner, as can be observed in the right plot. For the full model, the subgradient-based method becomes extremely slow, and the UAS score severely degrades (after 1000 iterations it is 2% less than the one obtained with the ADMM-based method, with very few instances having been solved to optimality). The reason is the number of slaves: in this configuration and dataset the average number of slaves per instance is 3327.4, and the largest number is 113207. On the contrary, the ADMM method keeps a robust performance, with a large fraction of optimality certificates in early iterations.

Runtime and caching strategies. Despite its suitability to problems with many overlapping components, our parser is still 1.6 times slower than Koo et al. (2010) (0.34 against 0.21 sec./sent. in PTB §23), and is far beyond the speed of transition-based parsers (*e.g.*, Huang and Sagae (2010) take 0.04 sec./sent. on the same data, although accuracy is lower, 92.1%). Our implementation, however, is not fully optimized. We next describe how considerable speed-ups are achieved by caching the subproblems, following a strategy similar to Koo et al. (2010).

Fig. 3 illustrates the point. After a few iterations, many variables $u(r)$ see a consensus being achieved (*i.e.*, $u^t(r) = z_s^{t+1}(r), \forall s$) and enter an idle state: they are left unchanged by the u-update in Eq. 9, and so do the Lagrange variables $\lambda_s^{t+1}(r)$ (Eq. 10). If by iteration t all variables in a subproblem s are idle, then $z_s^{t+1}(r) = z_s^t(r)$, hence the subproblem does not need to be resolved.[15] Fig. 3 shows that many variables and subproblems are left untouched after the first few rounds.

[15]Even if not all variables are idle in s, caching may still be useful: note that the z-updates in Eq. 14 tend to be sparse for the subproblems described in §4 (these are Euclidean projections onto polytopes with 0/1 vertices, which tend to hit corners). Another trick that may accelerate the algorithm is warm-starting: since many subproblems involve a sort operation, storing the sorted indexes may speedup the next round.

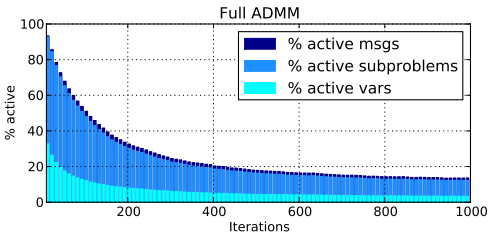

Figure 3: Fraction of active variables, subproblems and messages along DD-ADMM iterations (full model). The number of active messages denotes the total number of variables (active or not) that participate in an active factor.

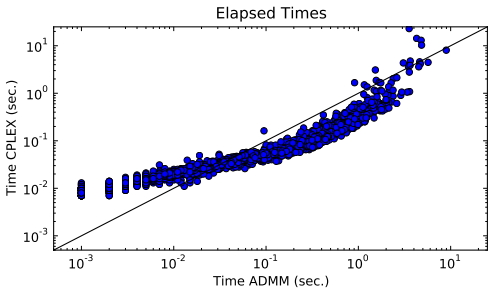

Figure 4: Runtimes of DD-ADMM and CPLEX on PTB §22 (each point is a sentence). Average runtimes are 0.362 (DD-ADMM) and 0.565 sec./sent. (CPLEX).

Finally, Fig. 4 compares the runtimes of our implementation of DD-ADMM with those achieved by a state-of-the-art LP solver, CPLEX, in its best performing configuration: the simplex algorithm applied to the dual LP. We observe that DD-ADMM is faster in some regimes but slower in others. For short sentences (< 15 words), DD-ADMM tends to be faster. For longer sentences, CPLEX is quite effective as it uses good heuristics for the pivot steps in the simplex algorithm; however, we observed that it sometimes gets trapped on large problems. Note also that DD-ADMM is not fully optimized, and that it is much more amenable to parallelization than the simplex algorithm, since it is composed of many independent slaves. This suggests potentially significant speed-ups in multi-core environments.

6 Related Work

Riedel and Clarke (2006) first formulated dependency parsing as an integer program, along with logical constraints. The multicommodity flow for-

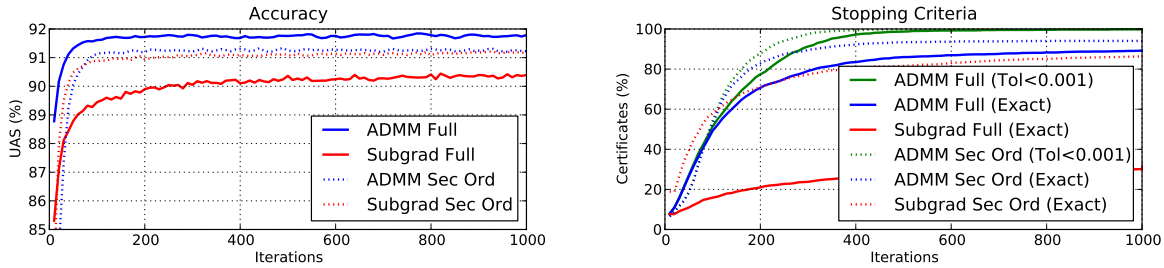

Figure 2: UAS including punctuation (left) and fraction of optimality certificates (right) accross iterations of the subgradient and DD-ADMM algorithms, in PTB §22. "Full" is our full model; "Sec Ord" is a second-order model with grandparents and all siblings, for which the subgradient method uses a coarser decomposition with a TREE factor. Since subgradient and DD-ADMM are solving the same problems, the solid lines (as the dashed ones) would meet in the limit, however subgradient converges very slowly for the full model. The right plot shows optimality certificates for both methods, indicating that an exact solution of P has been found; for DD-ADMM we also plot the fraction of instances that converged to an accurate solution of P' (primal and dual residuals $< 10^{-3}$) and hence can be stopped.

mulation was introduced by Martins et al. (2009a), along with some of the parts considered here. Koo et al. (2010) proposed a subgradient-based dual decomposition method that elegantly combines head automata with maximum spanning tree algorithms; these parsers, as well as the loopy belief propagation method of Smith and Eisner (2008), are all instances of turbo parsers (Martins et al., 2010).

DD-ADMM has been proposed and theoretically analyzed by Martins et al. (2011) for problems representable as factor graphs. The general ADMM method has a long-standing history in optimization (Hestenes, 1969; Powell, 1969; Glowinski and Marroco, 1975; Gabay and Mercier, 1976; Boyd et al., 2011). Other methods have been recently proposed to accelerate dual decomposition, such as Jojic et al. (2010) and Meshi and Globerson (2011) (the latter applying ADMM in the dual rather than the primal).

While our paper shows limitations of the subgradient method when there are many overlapping components, this method may still be advantageous over ADMM in problems that are nicely decomposable, since it often allows reusing existing combinatorial machinery. Yet, the scenario we consider here is realistic in NLP, where we often have to deal with not-lightly-decomposable constrained problems (e.g., exploiting linguistic knowledge).

7 Conclusion

We have introduced new feature-rich turbo parsers. Since exact decoding is intractable, we solve an LP relaxation through a recently proposed consensus algorithm, DD-ADMM, which is suitable for problems with many overlapping components. We study the empirical runtime and convergence properties of DD-ADMM, complementing the theoretical treatment in Martins et al. (2011). DD-ADMM compares favourably against the subgradient method in several aspects: it is faster to reach a consensus, it has better stopping conditions, and it works better in non-lightweight decompositions. While its slave subproblems are more involved, we derived closed-form solutions for many cases of interest, such as first-order logic formulas and combinatorial factors.

DD-ADMM may be useful in other frameworks involving logical constraints, such as the models for compositional semantics presented by Liang et al. (2011). Non-logical constraints may also yield efficient subproblems, *e.g.*, the length constraints in summarization and compression (Clarke and Lapata, 2008; Martins and Smith, 2009; Berg-Kirkpatrick et al., 2011). Finally, DD-ADMM can be adapted to tighten its relaxations towards exact decoding, as in Sontag et al. (2008) and Rush and Collins (2011). We defer this for future work.

Acknowledgments

We thank all reviewers for their comments, Eric Xing for helpful discussions, and Terry Koo and Sasha Rush for answering questions about their parser and for providing code. A. M. was supported by a FCT/ICTI grant through the CMU-Portugal Program, and by Priberam. This work was partially supported by the FET programme (EU FP7), under the SIMBAD project (contract 213250). N. S. was supported by NSF CAREER IIS-1054319.

References

M. Auli and A. Lopez. 2011. A Comparison of Loopy Belief Propagation and Dual Decomposition for Integrated CCG Supertagging and Parsing. In *Proc. of ACL*.

Y. Bar-Hillel, M. Perles, and E. Shamir. 1964. On formal properties of simple phrase structure grammars. *Language and Information: Selected Essays on their Theory and Application*, pages 116–150.

Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. 2011. Jointly learning to extract and compress. In *Proc. of ACL*.

D. Bertsekas, W. Hager, and O. Mangasarian. 1999. *Nonlinear programming*. Athena Scientific.

D.P. Bertsekas, A. Nedic, and A.E. Ozdaglar. 2003. *Convex analysis and optimization*. Athena Scientific.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. 2011. *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Now Publishers (to appear).

J.P. Boyle and R.L. Dykstra. 1986. A method for finding projections onto the intersections of convex sets in Hilbert spaces. In *Advances in order restricted statistical inference*, pages 28–47. Springer Verlag.

S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *CoNLL*.

X. Carreras, M. Collins, and T. Koo. 2008. TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-rich Parsing. In *CONLL*.

X. Carreras. 2007. Experiments with a higher-order projective dependency parser. In *CoNLL*.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proc. ACL*, pages 173–180. Association for Computational Linguistics Morristown, NJ, USA.

D. Chiang. 2007. Hierarchical phrase-based translation. *computational linguistics*, 33(2):201–228.

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.

J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. 2008. Efficient projections onto the L1-ball for learning in high dimensions. In *ICML*.

J. Eckstein and D. Bertsekas. 1992. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1):293–318.

J. R. Finkel, A. Kleeman, and C. D. Manning. 2008. Efficient, feature-based, conditional random field parsing. *Proceedings of ACL-08: HLT*, pages 959–967.

D. Gabay and B. Mercier. 1976. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers and Mathematics with Applications*, 2(1):17–40.

J. Giménez and L. Marquez. 2004. Svmtool: A general pos tagger generator based on support vector machines. In *Proc. of LREC*.

R. Glowinski and P. Le Tallec. 1989. *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*. Society for Industrial Mathematics.

R. Glowinski and A. Marroco. 1975. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par penalisation-dualité, d'une classe de problèmes de Dirichlet non linéaires. *Rev. Franc. Automat. Inform. Rech. Operat.*, 9:41–76.

M. Hestenes. 1969. Multiplier and gradient methods. *Jour. Optim. Theory and Applic.*, 4:302–320.

L. Huang and K. Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proc. of ACL*, pages 1077–1086.

R. Johansson and P. Nugues. 2008. Dependency-based Semantic Role Labeling of PropBank. In *EMNLP*.

V. Jojic, S. Gould, and D. Koller. 2010. Accelerated dual decomposition for MAP inference. In *ICML*.

N. Komodakis, N. Paragios, and G. Tziritas. 2007. MRF optimization via dual decomposition: Message-passing revisited. In *ICCV*.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In *Proc. of ACL*, pages 1–11.

T. Koo, A. M. Rush, M. Collins, T. Jaakkola, and D. Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *EMNLP*.

A. Kulesza and F. Pereira. 2007. Structured Learning with Approximate Inference. *NIPS*.

P. Liang, M.I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *Proc. Association for Computational Linguistics (ACL)*.

A. F. T. Martins and N. A. Smith. 2009. Summarization with a joint model for sentence extraction and compression. In *NAACL-HLT Workshop on Integer Linear Programming for NLP*.

A. F. T. Martins, D. Das, N. A. Smith, and E. P. Xing. 2008. Stacking dependency parsers. In *EMNLP*.

A. F. T. Martins, N. A. Smith, and E. P. Xing. 2009a. Concise integer linear programming formulations for dependency parsing. In *ACL-IJCNLP*.

A. F. T. Martins, N. A. Smith, and E. P. Xing. 2009b. Polyhedral outer approximations with application to natural language parsing. In *ICML*.

A. F. T. Martins, N. A. Smith, E. P. Xing, M. A. T. Figueiredo, and P. M. Q. Aguiar. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *EMNLP*.

A. F. T. Martins, M. A. T. Figueiredo, P. M. Q. Aguiar, N. A. Smith, and E. P. Xing. 2011. An Augmented Lagrangian Approach to Constrained MAP Inference. In *ICML*.

R. McDonald, K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *CoNLL*.

O. Meshi and A. Globerson. 2011. An Alternating Direction Method for Dual MAP LP Relaxation. In *ECML PKDD*.

J. Nivre and R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *ACL-HLT*.

J. Nivre, J. Hall, J. Nilsson, G. Eryiğit, and S. Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Procs. of CoNLL*.

S. Petrov and D. Klein. 2008. Sparse multi-scale grammars for discriminative latent variable parsing. In *Proc. of EMNLP*.

M. Powell. 1969. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization*, pages 283–298. Academic Press.

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 *EMNLP*.

A. M. Rush and M. Collins. 2011. Exact decoding of syntactic translation models through lagrangian relaxation. In *ACL*.

A. Rush, D. Sontag, M. Collins, and T. Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *EMNLP*.

N. Shor. 1985. *Minimization methods for non-differentiable functions*. Springer.

D. Smith and J. Eisner. 2008. Dependency parsing by belief propagation. In *EMNLP*.

D. Sontag, T. Meltzer, A. Globerson, Y. Weiss, and T Jaakkola. 2008. Tightening LP relaxations for MAP using message-passing. In *UAI*.

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. *CoNLL*.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin Markov networks. In *NIPS*.

R.W. Tromble and J. Eisner. 2006. A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *Proc. of NAACL*, pages 423–430.

M. Wainwright and M. Jordan. 2008. *Graphical Models, Exponential Families, and Variational Inference.* Now Publishers.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *IWPT*.