On Dual Decomposition and Linear Programming Relaxations for Natural Language Processing

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

Abstract

This paper introduces *dual decomposition* as a framework for deriving inference algorithms for NLP problems. The approach relies on standard dynamic-programming algorithms as oracle solvers for sub-problems, together with a simple method for forcing agreement between the different oracles. The approach provably solves a linear programming (LP) relaxation of the global inference problem. It leads to algorithms that are *simple*, in that they use existing decoding algorithms; *efficient*, in that they avoid exact algorithms for the full model; and often *exact*, in that empirically they often recover the correct solution in spite of using an LP relaxation. We give experimental results on two problems: 1) the combination of two lexicalized parsing models; and 2) the combination of a lexicalized parsing model and a trigram part-of-speech tagger.

1 Introduction

Dynamic programming algorithms have been remarkably useful for inference in many NLP problems. Unfortunately, as models become more complex, for example through the addition of new features or components, dynamic programming algorithms can quickly explode in terms of computational or implementational complexity.[1] As a result, efficiency of inference is a critical bottleneck for many problems in statistical NLP.

This paper introduces *dual decomposition* (Dantzig and Wolfe, 1960; Komodakis et al., 2007) as a framework for deriving inference algorithms in NLP. Dual decomposition leverages the observation that complex inference problems can often be decomposed into efficiently solvable sub-problems. The approach leads to inference algorithms with the following properties:

- The resulting algorithms are simple and efficient, building on standard dynamic-programming algorithms as oracle solvers for sub-problems,[2] together with a method for forcing agreement between the oracles.
- The algorithms provably solve a linear programming (LP) relaxation of the original inference problem.
- Empirically, the LP relaxation often leads to an exact solution to the original problem.

The approach is very general, and should be applicable to a wide range of problems in NLP. The connection to linear programming ensures that the algorithms provide a certificate of optimality when they recover the exact solution, and also opens up the possibility of methods that incrementally tighten the LP relaxation until it is exact (Sherali and Adams, 1994; Sontag et al., 2008).

The structure of this paper is as follows. We first give two examples as an illustration of the approach: 1) integrated parsing and trigram part-of-speech (POS) tagging; and 2) combined phrase-structure and dependency parsing. In both settings, it is possible to solve the integrated problem through an "intersected" dynamic program (e.g., for integration of parsing and tagging, the construction from Bar-Hillel et al. (1964) can be used). However, these methods, although polynomial time, are substantially less efficient than our algorithms, and are considerably more complex to implement.

Next, we describe exact polyhedral formulations for the two problems, building on connections between dynamic programming algorithms and marginal polytopes, as described in Martin et al. (1990). These allow us to precisely characterize the relationship between the exact formulations and the

[1] The same is true for NLP inference algorithms based on other exact combinatorial methods, for example methods based on minimum-weight spanning trees (McDonald et al., 2005), or graph cuts (Pang and Lee, 2004).

[2] More generally, other exact inference methods can be used as oracles, for example spanning tree algorithms for non-projective dependency structures.

Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 1–11,
MIT, Massachusetts, USA, 9-11 October 2010.

LP relaxations that we solve. We then give guarantees of convergence for our algorithms by showing that they are instantiations of Lagrangian relaxation, a general method for solving linear programs of a particular form.

Finally, we describe experiments that demonstrate the effectiveness of our approach. First, we consider the integration of the generative model for phrase-structure parsing of Collins (2003), with the second-order discriminative dependency parser of Koo et al. (2008). This is an interesting problem in its own right: the goal is to inject the high performance of discriminative dependency models into phrase-structure parsing. The method uses off-the-shelf decoders for the two models. We find three main results: 1) in spite of solving an LP relaxation, empirically the method finds an exact solution on over 99% of the examples; 2) the method converges quickly, typically requiring fewer than 10 iterations of decoding; 3) the method gives gains over a baseline method that forces the phrase-structure parser to produce the same dependencies as the first-best output from the dependency parser (the Collins (2003) model has an F1 score of 88.1%; the baseline method has an F1 score of 89.7%; and the dual decomposition method has an F1 score of 90.7%).

In a second set of experiments, we use dual decomposition to integrate the trigram POS tagger of Toutanova and Manning (2000) with the parser of Collins (2003). We again find that the method finds an exact solution in almost all cases, with convergence in just a few iterations of decoding.

Although the focus of this paper is on dynamic programming algorithms—both in the experiments, and also in the formal results concerning marginal polytopes—it is straightforward to use other combinatorial algorithms within the approach. For example, Koo et al. (2010) describe a dual decomposition approach for non-projective dependency parsing, which makes use of both dynamic programming and spanning tree inference algorithms.

2 Related Work

Dual decomposition is a classical method for solving optimization problems that can be decomposed into efficiently solvable sub-problems. Our work is inspired by dual decomposition methods for inference in Markov random fields (MRFs) (Wainwright et al., 2005a; Komodakis et al., 2007; Globerson and Jaakkola, 2007). In this approach, the MRF is decomposed into sub-problems corresponding to tree-structured subgraphs that together cover all edges of the original graph. The resulting inference algorithms provably solve an LP relaxation of the MRF inference problem, often significantly faster than commercial LP solvers (Yanover et al., 2006).

Our work is also related to methods that incorporate combinatorial solvers within loopy belief propagation (LBP), either for MAP inference (Duchi et al., 2007) or for computing marginals (Smith and Eisner, 2008). Our approach similarly makes use of combinatorial algorithms to efficiently solve sub-problems of the global inference problem. However, unlike LBP, our algorithms have strong theoretical guarantees, such as guaranteed convergence and the possibility of a certificate of optimality. These guarantees are possible because our algorithms directly solve an LP relaxation.

Other work has considered LP or integer linear programming (ILP) formulations of inference in NLP (Martins et al., 2009; Riedel and Clarke, 2006; Roth and Yih, 2005). These approaches typically use general-purpose LP or ILP solvers. Our method has the advantage that it leverages underlying structure arising in LP formulations of NLP problems. We will see that dynamic programming algorithms such as CKY can be considered to be very efficient solvers for particular LPs. In dual decomposition, these LPs—and their efficient solvers—can be embedded within larger LPs corresponding to more complex inference problems.

3 Background: Structured Models for NLP

We now describe the type of models used throughout the paper. We take some care to set up notation that will allow us to make a clear connection between inference problems and linear programming.

Our first example is weighted CFG parsing. We assume a context-free grammar, in Chomsky normal form, with a set of non-terminals N. The grammar contains all rules of the form $A \rightarrow B\ C$ and $A \rightarrow w$ where $A, B, C \in N$ and $w \in V$ (it is simple to relax this assumption to give a more constrained grammar). For rules of the form $A \rightarrow w$ we refer to A as the part-of-speech tag for w. We allow any non-terminal to be at the root of the tree.

Given a sentence with n words, $w_1, w_2, \ldots w_n$, a parse tree is a set of rule productions of the form $\langle A \rightarrow B\ C, i, k, j \rangle$ where $A, B, C \in N$, and $1 \leq i \leq k < j \leq n$. Each rule production represents the use of CFG rule $A \rightarrow B\ C$ where non-terminal A spans words $w_i \ldots w_j$, non-terminal B spans words $w_i \ldots w_k$, and non-terminal C spans words $w_{k+1} \ldots w_j$. There are $O(|N|^3 n^3)$ such rule productions. Each parse tree corresponds to a subset of these rule productions, of size $n-1$, that forms a well-formed parse tree.[3]

We now define the *index set* for CFG parsing as

$$\mathcal{I} = \{\langle A \rightarrow B\ C, i, k, j \rangle :\ A, B, C \in N,\ 1 \leq i \leq k < j \leq n\}$$

Each parse tree is a vector $y = \{y_r : r \in \mathcal{I}\}$, with $y_r = 1$ if rule r is in the parse tree, and $y_r = 0$ otherwise. Hence each parse tree is represented as a vector in $\{0,1\}^m$, where $m = |\mathcal{I}|$. We use $\mathcal{Y}$ to denote the set of all valid parse-tree vectors; the set $\mathcal{Y}$ is a subset of $\{0,1\}^m$ (not all binary vectors correspond to valid parse trees).

In addition, we assume a vector $\theta = \{\theta_r : r \in \mathcal{I}\}$ that specifies a weight for each rule production.[4] Each θ_r can take any value in the reals. The optimal parse tree is $y^* = \arg\max_{y \in \mathcal{Y}}\ y \cdot \theta$ where $y \cdot \theta = \sum_r y_r \theta_r$ is the inner product between y and θ.

We use y_r and $y(r)$ interchangeably (similarly for θ_r and $\theta(r)$) to refer to the r'th component of the vector y. For example $\theta(A \rightarrow B\ C, i, k, j)$ is a weight for the rule $\langle A \rightarrow B\ C, i, k, j \rangle$.

We will use similar notation for other problems. As a second example, in POS tagging the task is to map a sentence of n words $w_1 \ldots w_n$ to a tag sequence $t_1 \ldots t_n$, where each t_i is chosen from a set T of possible tags. We assume a trigram tagger, where a tag sequence is represented through decisions $\langle (A, B) \rightarrow C, i \rangle$ where $A, B, C \in T$, and $i \in \{3 \ldots n\}$. Each production represents a transition where C is the tag of word w_i, and (A, B) are the previous two tags. The index set for tagging is

$$\mathcal{I}_{\text{tag}} = \{\langle (A, B) \rightarrow C, i \rangle\ :\ A, B, C \in T, 3 \leq i \leq n\}$$

Note that we do not need transitions for $i = 1$ or $i = 2$, because the transition $\langle (A, B) \rightarrow C, 3 \rangle$ specifies the first three tags in the sentence.[5]

Each tag sequence is represented as a vector $z = \{z_r : r \in \mathcal{I}_{\text{tag}}\}$, and we denote the set of valid tag sequences, a subset of $\{0,1\}^{|\mathcal{I}_{\text{tag}}|}$, as $\mathcal{Z}$. Given a parameter vector $\theta = \{\theta_r : r \in \mathcal{I}_{\text{tag}}\}$, the optimal tag sequence is $\arg\max_{z \in \mathcal{Z}} z \cdot \theta$.

As a modification to the above approach, we will find it convenient to introduce extended index sets for both the CFG and POS tagging examples. For the CFG case we define the extended index set to be $\mathcal{I}' = \mathcal{I} \cup \mathcal{I}_{\text{uni}}$ where

$$\mathcal{I}_{\text{uni}} = \{(i, t) : i \in \{1 \ldots n\}, t \in T\}$$

Here each pair (i, t) represents word w_i being assigned the tag t. Thus each parse-tree vector y will have additional (binary) components $y(i, t)$ specifying whether or not word i is assigned tag t. (Throughout this paper we will assume that the tag-set used by the tagger, T, is a subset of the set of non-terminals considered by the parser, N.) Note that this representation is over-complete, since a parse tree determines a unique tagging for a sentence: more explicitly, for any $i \in \{1 \ldots n\}$, $Y \in T$, the following linear constraint holds:

$$y(i, Y) = \sum_{k=i+1}^{n} \sum_{X, Z \in N} y(X \rightarrow Y\ Z, i, i, k) + \sum_{k=1}^{i-1} \sum_{X, Z \in N} y(X \rightarrow Z\ Y, k, i-1, i)$$

We apply the same extension to the tagging index set, effectively mapping trigrams down to unigram assignments, again giving an over-complete representation. The extended index set for tagging is referred to as $\mathcal{I}'_{\text{tag}}$.

From here on we will make exclusive use of extended index sets for CFG parsing and trigram tagging. We use the set $\mathcal{Y}$ to refer to the set of valid parse structures under the extended representation;

[3] We do not require rules of the form $A \rightarrow w_i$ in this representation, as they are redundant: specifically, a rule production $\langle A \rightarrow B\ C, i, k, j \rangle$ implies a rule $B \rightarrow w_i$ iff $i = k$, and $C \rightarrow w_j$ iff $j = k + 1$.

[4] We do not require parameters for rules of the form $A \rightarrow w$, as they can be folded into rule production parameters. E.g., under a PCFG we define $\theta(A \rightarrow B\ C, i, k, j) = \log P(A \rightarrow B\ C \mid A) + \delta_{i,k} \log P(B \rightarrow w_i | B) + \delta_{k+1,j} \log P(C \rightarrow w_j | C)$ where $\delta_{x,y} = 1$ if $x = y$, 0 otherwise.

[5] As one example, in an HMM, the parameter $\theta((A, B) \rightarrow C, 3)$ would be $\log P(A|**) + \log P(B|*A) + \log P(C|AB) + \log P(w_1|A) + \log P(w_2|B) + \log P(w_3|C)$, where $*$ is the start symbol.

each $y \in \mathcal{Y}$ is a binary vector of length $|\mathcal{I}'|$. We similarly use $\mathcal{Z}$ to refer to the set of valid tag structures under the extended representation. We assume parameter vectors for the two problems, $\theta^{\text{cfg}} \in \mathbb{R}^{|\mathcal{I}'|}$ and $\theta^{\text{tag}} \in \mathbb{R}^{|\mathcal{I}'_{\text{tag}}|}$.

4 Two Examples

This section describes the dual decomposition approach for two inference problems in NLP.

4.1 Integrated Parsing and Trigram Tagging

We now describe the dual decomposition approach for integrated parsing and trigram tagging. First, define the set $\mathcal{Q}$ as follows:

$$\mathcal{Q} = \{(y, z) : y \in \mathcal{Y}, z \in \mathcal{Z}, \\ y(i,t) = z(i,t) \text{ for all } (i,t) \in \mathcal{I}_{uni}\} \quad (1)$$

Hence $\mathcal{Q}$ is the set of all (y, z) pairs that agree on their part-of-speech assignments. The integrated parsing and trigram tagging problem is then to solve

$$\max_{(y,z)\in\mathcal{Q}} \left(y \cdot \theta^{\text{cfg}} + z \cdot \theta^{\text{tag}} \right) \quad (2)$$

This problem is equivalent to

$$\max_{y\in\mathcal{Y}} \left(y \cdot \theta^{\text{cfg}} + g(y) \cdot \theta^{\text{tag}} \right)$$

where $g : \mathcal{Y} \rightarrow \mathcal{Z}$ is a function that maps a parse tree y to its set of trigrams $z = g(y)$. The benefit of the formulation in Eq. 2 is that it makes explicit the idea of maximizing over all pairs (y, z) under a set of agreement constraints $y(i,t) = z(i,t)$—this concept will be central to the algorithms in this paper.

With this in mind, we note that we have efficient methods for the inference problems of tagging and parsing alone, and that our combined objective almost separates into these two independent problems. In fact, if we drop the $y(i,t) = z(i,t)$ constraints from the optimization problem, the problem splits into two parts, each of which can be efficiently solved using dynamic programming:

$$(y^*, z^*) = (\arg\max_{y\in\mathcal{Y}} y \cdot \theta^{\text{cfg}}, \arg\max_{z\in\mathcal{Z}} z \cdot \theta^{\text{tag}})$$

Dual decomposition exploits this idea; it results in the algorithm given in figure 1. The algorithm optimizes the combined objective by repeatedly solving the two sub-problems separately—that is, it directly solves the harder optimization problem using an existing CFG parser and trigram tagger. After each iteration the algorithm adjusts the weights $u(i,t)$; these updates modify the objective functions for the two models, encouraging them to agree on the same POS sequence. In section 6.1 we will show that the variables $u(i,t)$ are Lagrange multipliers enforcing agreement constraints, and that the algorithm corresponds to a (sub)gradient method for optimization of a dual function. The algorithm is easy to implement: all that is required is a decoding algorithm for each of the two models, and simple additive updates to the Lagrange multipliers enforcing agreement between the two models.

Set $u^{(1)}(i,t) \leftarrow 0$ for all $(i,t) \in \mathcal{I}_{\text{uni}}$
for $k = 1$ to K **do**

$$y^{(k)} \leftarrow \arg\max_{y\in\mathcal{Y}} \left(y \cdot \theta^{\text{cfg}} - \sum_{(i,t)\in\mathcal{I}_{\text{uni}}} u^{(k)}(i,t) y(i,t) \right)$$

$$z^{(k)} \leftarrow \arg\max_{z\in\mathcal{Z}} \left(z \cdot \theta^{\text{tag}} + \sum_{(i,t)\in\mathcal{I}_{\text{uni}}} u^{(k)}(i,t) z(i,t) \right)$$

  if $y^{(k)}(i,t) = z^{(k)}(i,t)$ for all $(i,t) \in \mathcal{I}_{\text{uni}}$ **then**
    return $(y^{(k)}, z^{(k)})$
  for all $(i,t) \in \mathcal{I}_{\text{uni}}$,
    $u^{(k+1)}(i,t) \leftarrow u^{(k)}(i,t) + \alpha_k (y^{(k)}(i,t) - z^{(k)}(i,t))$
return $(y^{(K)}, z^{(K)})$

Figure 1: The algorithm for integrated parsing and tagging. The parameters $\alpha_k > 0$ for $k = 1 \ldots K$ specify step sizes for each iteration, and are discussed further in the Appendix. The two $\arg\max$ problems can be solved using dynamic programming.

4.2 Integrating Two Lexicalized Parsers

Our second example problem is the integration of a phrase-structure parser with a higher-order dependency parser. The goal is to add higher-order features to phrase-structure parsing without greatly increasing the complexity of inference.

First, we define an index set for second-order unlabeled projective dependency parsing. The second-order parser considers first-order dependencies, as well as grandparent and sibling second-order dependencies (e.g., see Carreras (2007)). We assume that $\mathcal{I}_{\text{dep}}$ is an index set containing all such dependencies (for brevity we omit the details of this index set). For convenience we define an extended index set that makes explicit use of first-order dependen-

cies, $\mathcal{I}'_{\text{dep}} = \mathcal{I}_{\text{dep}} \cup \mathcal{I}_{\text{first}}$, where

$$\mathcal{I}_{\text{first}} = \{(i,j) \ : \ i \in \{0 \ldots n\}, j \in \{1 \ldots n\}, i \neq j\}$$

Here (i,j) represents a dependency with head w_i and modifier w_j ($i = 0$ corresponds to the root symbol in the parse). We use $\mathcal{D} \subseteq \{0,1\}^{|\mathcal{I}'_{\text{dep}}|}$ to denote the set of valid projective dependency parses.

The second model we use is a lexicalized CFG. Each symbol in the grammar takes the form $A(h)$ where $A \in N$ is a non-terminal, and $h \in \{1 \ldots n\}$ is an index specifying that w_h is the head of the constituent. Rule productions take the form $\langle A(a) \rightarrow B(b) \ C(c), i, k, j \rangle$ where $b \in \{i \ldots k\}$, $c \in \{(k+1) \ldots j\}$, and a is equal to b or c, depending on whether A receives its head-word from its left or right child. Each such rule implies a dependency (a,b) if $a = c$, or (a,c) if $a = b$. We take $\mathcal{I}_{\text{head}}$ to be the index set of all such rules, and $\mathcal{I}'_{\text{head}} = \mathcal{I}_{\text{head}} \cup \mathcal{I}_{\text{first}}$ to be the extended index set. We define $\mathcal{H} \subseteq \{0,1\}^{|\mathcal{I}'_{\text{head}}|}$ to be the set of valid parse trees.

The integrated parsing problem is then to find

$$(y^*, d^*) = \arg \max_{(y,d) \in \mathcal{R}} \left(y \cdot \theta^{\text{head}} + d \cdot \theta^{\text{dep}} \right) \quad (3)$$

$$\text{where} \quad \mathcal{R} = \{(y,d) : y \in \mathcal{H}, d \in \mathcal{D}, \\ y(i,j) = d(i,j) \text{ for all } (i,j) \in \mathcal{I}_{\text{first}}\}$$

This problem has a very similar structure to the problem of integrated parsing and tagging, and we can derive a similar dual decomposition algorithm. The Lagrange multipliers u are a vector in $\mathbb{R}^{|\mathcal{I}_{\text{first}}|}$ enforcing agreement between dependency assignments. The algorithm (omitted for brevity) is identical to the algorithm in figure 1, but with $\mathcal{I}_{\text{uni}}$, $\mathcal{Y}$, $\mathcal{Z}$, θ^{cfg}, and θ^{tag} replaced with $\mathcal{I}_{\text{first}}$, $\mathcal{H}$, $\mathcal{D}$, θ^{head}, and θ^{dep} respectively. The algorithm only requires decoding algorithms for the two models, together with simple updates to the Lagrange multipliers.

5 Marginal Polytopes and LP Relaxations

We now give formal guarantees for the algorithms in the previous section, showing that they solve LP relaxations of the problems in Eqs. 2 and 3.

To make the connection to linear programming, we first introduce the idea of *marginal polytopes* in section 5.1. In section 5.2, we give a precise statement of the LP relaxations that are being solved by the example algorithms, making direct use of marginal polytopes. In section 6 we will prove that the example algorithms solve these LP relaxations.

5.1 Marginal Polytopes

For a finite set $\mathcal{Y}$, define the set of all distributions over elements in $\mathcal{Y}$ as $\Delta = \{\alpha \in \mathbb{R}^{|\mathcal{Y}|} : \alpha_y \geq 0, \sum_{y \in \mathcal{Y}} \alpha_y = 1\}$. Each $\alpha \in \Delta$ gives a vector of marginals, $\mu = \sum_{y \in \mathcal{Y}} \alpha_y y$, where μ_r can be interpreted as the probability that $y_r = 1$ for a y selected at random from the distribution α.

The set of all possible marginal vectors, known as the *marginal polytope*, is defined as follows:

$$\mathcal{M} = \{\mu \in \mathbb{R}^m : \exists \alpha \in \Delta \text{ such that } \mu = \sum_{y \in \mathcal{Y}} \alpha_y y\}$$

$\mathcal{M}$ is also frequently referred to as the *convex hull* of $\mathcal{Y}$, written as $\text{conv}(\mathcal{Y})$. We use the notation $\text{conv}(\mathcal{Y})$ in the remainder of this paper, instead of $\mathcal{M}$.

For an arbitrary set $\mathcal{Y}$, the marginal polytope $\text{conv}(\mathcal{Y})$ can be complex to describe.[6] However, Martin et al. (1990) show that for a very general class of dynamic programming problems, the corresponding marginal polytope can be expressed as

$$\text{conv}(\mathcal{Y}) = \{\mu \in \mathbb{R}^m : A\mu = b, \mu \geq 0\} \quad (4)$$

where A is a $p \times m$ matrix, b is vector in $\mathbb{R}^p$, and the value p is linear in the size of a hypergraph representation of the dynamic program. Note that A and b specify a set of p linear constraints.

We now give an explicit description of the resulting constraints for CFG parsing:[7] similar constraints arise for other dynamic programming algorithms for parsing, for example the algorithms of Eisner (2000). The exact form of the constraints, and the fact that they are polynomial in number, is not essential for the formal results in this paper. However, a description of the constraints gives valuable intuition for the structure of the marginal polytope.

The constraints are given in figure 2. To develop some intuition, consider the case where the variables μ_r are restricted to be binary: hence each binary vector μ specifies a parse tree. The second constraint in Eq. 5 specifies that exactly one rule must be used at the top of the tree. The set of constraints in Eq. 6 specify that for each production of the form

[6] For any finite set $\mathcal{Y}$, $\text{conv}(\mathcal{Y})$ can be expressed as $\{\mu \in \mathbb{R}^m : A\mu \leq b\}$ where A is a matrix of dimension $p \times m$, and $b \in \mathbb{R}^p$ (see, e.g., Korte and Vygen (2008), pg. 65). The value for p depends on the set $\mathcal{Y}$, and can be exponential in size.

[7] Taskar et al. (2004) describe the same set of constraints, but without proof of correctness or reference to Martin et al. (1990).

$$\forall r \in \mathcal{I}',\ \mu_r \geq 0\ ; \quad \sum_{\substack{X,Y,Z \in N \\ k=1\ldots(n-1)}} \mu(X \rightarrow Y\ Z, 1, k, n) = 1 \quad (5)$$

$\forall X \in N, \forall (i,j)$ such that $1 \leq i < j \leq n$ and $(i,j) \neq (1,n)$:

$$\sum_{\substack{Y,Z \in N \\ k=i\ldots(j-1)}} \mu(X \rightarrow Y\ Z, i, k, j) = \sum_{\substack{Y,Z \in N \\ k=1\ldots(i-1)}} \mu(Y \rightarrow Z\ X, k, i-1, j) + \sum_{\substack{Y,Z \in N \\ k=(j+1)\ldots n}} \mu(Y \rightarrow X\ Z, i, j, k) \quad (6)$$

$$\forall Y \in T,\ \forall i \in \{1 \ldots n\} : \quad \mu(i, Y) = \sum_{\substack{X,Z \in N \\ k=(i+1)\ldots n}} \mu(X \rightarrow Y\ Z, i, i, k) + \sum_{\substack{X,Z \in N \\ k=1\ldots(i-1)}} \mu(X \rightarrow Z\ Y, k, i-1, i) \quad (7)$$

Figure 2: The linear constraints defining the marginal polytope for CFG parsing.

$\langle X \rightarrow Y\ Z, i, k, j \rangle$ in a parse tree, there must be exactly one production higher in the tree that generates (X, i, j) as one of its children. The constraints in Eq. 7 enforce consistency between the $\mu(i, Y)$ variables and rule variables higher in the tree. Note that the constraints in Eqs.(5–7) can be written in the form $A\mu = b$, $\mu \geq 0$, as in Eq. 4.

Under these definitions, we have the following:

Theorem 5.1 *Define $\mathcal{Y}$ to be the set of all CFG parses, as defined in section 4. Then*

$$\text{conv}(\mathcal{Y}) = \{\mu \in \mathbb{R}^m : \mu \textit{ satisifies Eqs.(5–7)}\}$$

Proof: This theorem is a special case of Martin et al. (1990), theorem 2.

The marginal polytope for tagging, $\text{conv}(\mathcal{Z})$, can also be expressed using linear constraints as in Eq. 4; see figure 3. These constraints follow from results for graphical models (Wainwright and Jordan, 2008), or from the Martin et al. (1990) construction.

As a final point, the following theorem gives an important property of marginal polytopes, which we will use at several points in this paper:

Theorem 5.2 *(Korte and Vygen (2008), page 66.) For any set $\mathcal{Y} \subseteq \{0,1\}^k$, and for any vector $\theta \in \mathbb{R}^k$,*

$$\max_{y \in \mathcal{Y}} y \cdot \theta = \max_{\mu \in \text{conv}(\mathcal{Y})} \mu \cdot \theta \quad (8)$$

The theorem states that for a linear objective function, maximization over a discrete set $\mathcal{Y}$ can be replaced by maximization over the convex hull

$$\forall r \in \mathcal{I}'_{\text{tag}},\ \nu_r \geq 0\ ; \quad \sum_{X,Y,Z \in T} \nu((X,Y) \rightarrow Z, 3) = 1$$

$\forall X \in T, \forall i \in \{3 \ldots n-1\}$:

$$\sum_{Y,Z \in T} \nu((Y,Z) \rightarrow X, i) = \sum_{Y,Z \in T} \nu((Y,X) \rightarrow Z, i+1)$$

$\forall X \in T, \forall i \in \{3 \ldots n-2\}$:

$$\sum_{Y,Z \in T} \nu((Y,Z) \rightarrow X, i) = \sum_{Y,Z \in T} \nu((X,Y) \rightarrow Z, i+2)$$

$$\forall X \in T, \forall i \in \{3 \ldots n\} : \ \nu(i, X) = \sum_{Y,Z \in T} \nu((Y,Z) \rightarrow X, i)$$

$$\forall X \in T : \quad \nu(1, X) = \sum_{Y,Z \in T} \nu((X,Y) \rightarrow Z, 3)$$

$$\forall X \in T : \quad \nu(2, X) = \sum_{Y,Z \in T} \nu((Y,X) \rightarrow Z, 3)$$

Figure 3: The linear constraints defining the marginal polytope for trigram POS tagging.

$\text{conv}(\mathcal{Y})$. The problem $\max_{\mu \in \text{conv}(\mathcal{Y})} \mu \cdot \theta$ is a linear programming problem.

For parsing, this theorem implies that:

1. Weighted CFG parsing can be framed as a linear programming problem, of the form $\max_{\mu \in \text{conv}(\mathcal{Y})} \mu \cdot \theta$, where $\text{conv}(\mathcal{Y})$ is specified by a polynomial number of linear constraints.

2. Conversely, dynamic programming algorithms such as the CKY algorithm can be considered to be oracles that efficiently solve LPs of the form $\max_{\mu \in \text{conv}(\mathcal{Y})} \mu \cdot \theta$.

Similar results apply for the POS tagging case.

5.2 Linear Programming Relaxations

We now describe the LP relaxations that are solved by the example algorithms in section 4. We begin with the algorithm in Figure 1.

The original optimization problem was to find $\max_{(y,z) \in \mathcal{Q}} \left(y \cdot \theta^{\text{cfg}} + z \cdot \theta^{\text{tag}} \right)$ (see Eq. 2). By theorem 5.2, this is equivalent to solving

$$\max_{(\mu,\nu) \in \text{conv}(\mathcal{Q})} \left(\mu \cdot \theta^{\text{cfg}} + \nu \cdot \theta^{\text{tag}} \right) \quad (9)$$

To formulate our approximation, we first define:

$$\mathcal{Q}' = \{(\mu, \nu) : \mu \in \text{conv}(\mathcal{Y}), \nu \in \text{conv}(\mathcal{Z}), \mu(i,t) = \nu(i,t) \text{ for all } (i,t) \in \mathcal{I}_{\text{uni}}\}$$

The definition of $\mathcal{Q}'$ is very similar to the definition of $\mathcal{Q}$ (see Eq. 1), the only difference being that $\mathcal{Y}$ and $\mathcal{Z}$ are replaced by $\text{conv}(\mathcal{Y})$ and $\text{conv}(\mathcal{Z})$ respectively. Hence any point in $\mathcal{Q}$ is also in $\mathcal{Q}'$. It follows that any point in $\text{conv}(\mathcal{Q})$ is also in $\mathcal{Q}'$, because $\mathcal{Q}'$ is a convex set defined by linear constraints.

The LP relaxation then corresponds to the following optimization problem:

$$\max_{(\mu,\nu)\in\mathcal{Q}'} \left(\mu \cdot \theta^{\text{cfg}} + \nu \cdot \theta^{\text{tag}}\right) \quad (10)$$

$\mathcal{Q}'$ is defined by linear constraints, making this a linear program. Since $\mathcal{Q}'$ is an *outer bound* on $\text{conv}(\mathcal{Q})$, i.e. $\text{conv}(\mathcal{Q}) \subseteq \mathcal{Q}'$, we obtain the guarantee that the value of Eq. 10 always upper bounds the value of Eq. 9.

In Appendix A we give an example showing that in general $\mathcal{Q}'$ includes points that are not in $\text{conv}(\mathcal{Q})$. These points exist because the agreement between the two parts is now enforced in expectation ($\mu(i,t) = \nu(i,t)$ for $(i,t) \in \mathcal{I}_{\text{uni}}$) rather than based on actual assignments. This agreement constraint is weaker since different distributions over assignments can still result in the same first order expectations. Thus, the solution to Eq. 10 may be in $\mathcal{Q}'$ but not in $\text{conv}(\mathcal{Q})$. It can be shown that all such solutions will be *fractional*, making them easy to distinguish from $\mathcal{Q}$. In many applications of LP relaxations—including the examples discussed in this paper—the relaxation in Eq. 10 turns out to be *tight*, in that the solution is often integral (i.e., it is in $\mathcal{Q}$). In these cases, solving the LP relaxation *exactly* solves the original problem of interest.

In the next section we prove that the algorithm in Figure 1 solves the problem in Eq 10. A similar result holds for the algorithm in section 4.2: it solves a relaxation of Eq. 3, where $\mathcal{R}$ is replaced by

$$\mathcal{R}' = \{(\mu,\nu) : \mu \in \text{conv}(\mathcal{H}), \nu \in \text{conv}(\mathcal{D}), \\ \mu(i,j) = \nu(i,j) \text{ for all } (i,j) \in \mathcal{I}_{\text{first}}\}$$

6 Convergence Guarantees

6.1 Lagrangian Relaxation

We now show that the example algorithms solve their respective LP relaxations given in the previous section. We do this by first introducing a general class of linear programs, together with an optimization method, *Lagrangian relaxation*, for solving these LPs. We then show that the algorithms in section 4 are special cases of the general algorithm.

The linear programs we consider take the form

$$\max_{x_1\in X_1, x_2\in X_2} (\theta_1 \cdot x_1 + \theta_2 \cdot x_2) \quad \text{such that } Ex_1 = Fx_2$$

The matrices $E \in \mathbb{R}^{q\times m}$ and $F \in \mathbb{R}^{q\times l}$ specify q linear "agreement" constraints between $x_1 \in \mathbb{R}^m$ and $x_2 \in \mathbb{R}^l$. The sets X_1, X_2 are also specified by linear constraints, $X_1 = \{x_1 \in \mathbb{R}^m : Ax_1 = b, x_1 \geq 0\}$ and $X_2 = \{x_2 \in \mathbb{R}^l : Cx_2 = d, x_2 \geq 0\}$, hence the problem is an LP.

Note that if we set $X_1 = \text{conv}(\mathcal{Y})$, $X_2 = \text{conv}(\mathcal{Z})$, and define E and F to specify the agreement constraints $\mu(i,t) = \nu(i,t)$, then we have the LP relaxation in Eq. 10.

It is natural to apply Lagrangian relaxation in cases where the sub-problems $\max_{x_1\in X_1} \theta_1 \cdot x_1$ and $\max_{x_2\in X_2} \theta_2 \cdot x_2$ can be efficiently solved by combinatorial algorithms for any values of θ_1, θ_2, but where the constraints $Ex_1 = Fx_2$ "complicate" the problem. We introduce Lagrange multipliers $u \in \mathbb{R}^q$ that enforce the latter set of constraints, giving the Lagrangian:

$$L(u, x_1, x_2) = \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + u \cdot (Ex_1 - Fx_2)$$

The dual objective function is

$$L(u) = \max_{x_1\in X_1, x_2\in X_2} L(u, x_1, x_2)$$

and the dual problem is to find $\min_{u\in\mathbb{R}^q} L(u)$.

Because X_1 and X_2 are defined by linear constraints, by strong duality we have

$$\min_{u\in\mathbb{R}^q} L(u) = \max_{x_1\in X_1, x_2\in X_2 : Ex_1 = Fx_2} (\theta_1 \cdot x_1 + \theta_2 \cdot x_2)$$

Hence minimizing $L(u)$ will recover the maximum value of the original problem. This leaves open the question of how to recover the LP solution (i.e., the pair (x_1^*, x_2^*) that achieves this maximum); we discuss this point in section 6.2.

The dual $L(u)$ is convex. However, $L(u)$ is not differentiable, so we cannot use gradient-based methods to optimize it. Instead, a standard approach is to use a subgradient method. Subgradients are tangent lines that lower bound a function even at points of non-differentiability: formally, a subgradient of a convex function $L : \mathcal{R}^n \to \mathcal{R}$ at a point u is a vector g_u such that for all v, $L(v) \geq L(u) + g_u \cdot (v - u)$.

$u^{(1)} \leftarrow 0$
for $k = 1$ to K **do**
  $x_1^{(k)} \leftarrow \arg\max_{x_1 \in X_1} (\theta_1 + (u^{(k)})^T E) \cdot x_1$
  $x_2^{(k)} \leftarrow \arg\max_{x_2 \in X_2} (\theta_2 - (u^{(k)})^T F) \cdot x_2$
  if $Ex_1^{(k)} = Fx_2^{(k)}$ **return** $u^{(k)}$
  $u^{(k+1)} \leftarrow u^{(k)} - \alpha_k (Ex_1^{(k)} - Fx_2^{(k)})$
return $u^{(K)}$

Figure 4: The Lagrangian relaxation algorithm.

By standard results, the subgradient for L at a point u takes a simple form, $g_u = Ex_1^* - Fx_2^*$, where

$$x_1^* = \arg \max_{x_1 \in X_1} (\theta_1 + (u^{(k)})^T E) \cdot x_1$$

$$x_2^* = \arg \max_{x_2 \in X_2} (\theta_2 - (u^{(k)})^T F) \cdot x_2$$

The beauty of this result is that the values of x_1^* and x_2^*, and by implication the value of the subgradient, can be computed using oracles for the two $\arg\max$ sub-problems.

Subgradient algorithms perform updates that are similar to gradient descent:

$$u^{(k+1)} \leftarrow u^{(k)} - \alpha_k g^{(k)}$$

where $g^{(k)}$ is the subgradient of L at $u^{(k)}$ and $\alpha_k > 0$ is the step size of the update. The complete subgradient algorithm is given in figure 4. The following convergence theorem is well-known (e.g., see page 120 of Korte and Vygen (2008)):

Theorem 6.1 *If* $\lim_{k \to \infty} \alpha_k = 0$ *and* $\sum_{k=1}^{\infty} \alpha_k = \infty$*, then* $\lim_{k \to \infty} L(u^{(k)}) = \min_u L(u)$.

The following proposition is easily verified:

Proposition 6.1 *The algorithm in figure 1 is an instantiation of the algorithm in figure 4,*[8] *with* $X_1 = \text{conv}(\mathcal{Y})$*,* $X_2 = \text{conv}(\mathcal{Z})$*, and the matrices* E *and* F *defined to be binary matrices specifying the constraints* $\mu(i,t) = \nu(i,t)$ *for all* $(i,t) \in \mathcal{I}_{uni}$.

Under an appropriate definition of the step sizes α_k, it follows that the algorithm in figure 1 defines a sequence of Lagrange multipliers $u^{(k)}$ minimizing a dual of the LP relaxation in Eq. 10. A similar result holds for the algorithm in section 4.2.

[8] with the caveat that it returns $(x_1^{(k)}, x_2^{(k)})$ rather than $u^{(k)}$.

6.2 Recovering the LP Solution

The previous section described how the method in figure 4 can be used to minimize the dual $L(u)$ of the original linear program. We now turn to the problem of recovering a primal solution (x_1^*, x_2^*) of the LP. The method we propose considers two cases:

(Case 1) If $Ex_1^{(k)} = Fx_2^{(k)}$ at any stage during the algorithm, then simply take $(x_1^{(k)}, x_2^{(k)})$ to be the primal solution. In this case the pair $(x_1^{(k)}, x_2^{(k)})$ *exactly* solves the original LP.[9] If this case arises in the algorithm in figure 1, then the resulting solution is binary (i.e., it is a member of $\mathcal{Q}$), and the solution exactly solves the original inference problem.

(Case 2) If case 1 does not arise, then a couple of strategies are possible. (This situation could arise in cases where the LP is not tight—i.e., it has a fractional solution—or where K is not large enough for convergence.) The first is to define the primal solution to be the average of the solutions encountered during the algorithm: $\hat{x}_1 = \sum_k x_1^{(k)}/K$, $\hat{x}_2 = \sum_k x_2^{(k)}/K$. Results from Nedić and Ozdaglar (2009) show that as $K \to \infty$, these averaged solutions converge to the optimal primal solution.[10] A second strategy (as given in figure 1) is to simply take $(x_1^{(K)}, x_2^{(K)})$ as an approximation to the primal solution. This method is a heuristic, but previous work (e.g., Komodakis et al. (2007)) has shown that it is effective in practice; we use it in this paper.

In our experiments we found that in the vast majority of cases, case 1 applies, after a small number of iterations; see the next section for more details.

7 Experiments

7.1 Integrated Phrase-Structure and Dependency Parsing

Our first set of experiments considers the integration of Model 1 of Collins (2003) (a lexicalized phrase-structure parser, from here on referred to as Model

[9] We have that $\theta_1 \cdot x_1^{(k)} + \theta_2 \cdot x_2^{(k)} = L(u^{(k)}, x_1^{(k)}, x_2^{(k)}) = L(u^{(k)})$, where the last equality is because $x_1^{(k)}$ and $x_2^{(k)}$ are defined by the respective $\arg\max$'s. Thus, $(x_1^{(k)}, x_2^{(k)})$ and $u^{(k)}$ are primal and dual optimal.

[10] The resulting fractional solution can be projected back to the set $\mathcal{Q}$, see (Smith and Eisner, 2008; Martins et al., 2009).

Itn.	1	2	3	4	5-10	11-20	20-50	**
Dep	43.5	20.1	10.2	4.9	14.0	5.7	1.4	0.4
POS	58.7	15.4	6.3	3.6	10.3	3.8	0.8	1.1

Table 1: Convergence results for Section 23 of the WSJ Treebank for the dependency parsing and POS experiments. Each column gives the percentage of sentences whose *exact* solutions were found in a given range of subgradient iterations. ** is the percentage of sentences that did not converge by the iteration limit (K=50).

1),[11] and the 2nd order discriminative dependency parser of Koo et al. (2008). The inference problem for a sentence x is to find

$$y^* = \arg\max_{y \in \mathcal{Y}} (f_1(y) + \gamma f_2(y)) \qquad (11)$$

where $\mathcal{Y}$ is the set of all lexicalized phrase-structure trees for the sentence x; $f_1(y)$ is the score (log probability) under Model 1; $f_2(y)$ is the score under Koo et al. (2008) for the dependency structure implied by y; and $\gamma > 0$ is a parameter dictating the relative weight of the two models.[12] This problem is similar to the second example in section 4; a very similar dual decomposition algorithm to that described in section 4.2 can be derived.

We used the Penn Wall Street Treebank (Marcus et al., 1994) for the experiments, with sections 2-21 for training, section 22 for development, and section 23 for testing. The parameter γ was chosen to optimize performance on the development set.

We ran the dual decomposition algorithm with a limit of $K = 50$ iterations. The dual decomposition algorithm returns an exact solution if case 1 occurs as defined in section 6.2; we found that of 2416 sentences in section 23, case 1 occurred for 2407 (99.6%) sentences. Table 1 gives statistics showing the number of iterations required for convergence. Over 80% of the examples converge in 5 iterations or fewer; over 90% converge in 10 iterations or fewer.

We compare the accuracy of the dual decomposition approach to two baselines: first, Model 1; and second, a naive integration method that enforces the hard constraint that Model 1 must only consider dependencies seen in the first-best output from the dependency parser. Table 2 shows all three results. The dual decomposition method gives a significant gain in precision and recall over the naive combination method, and boosts the performance of Model 1 to a level that is close to some of the best single-pass parsers on the Penn treebank test set. Dependency accuracy is also improved over the Koo et al. (2008) model, in spite of the relatively low dependency accuracy of Model 1 alone.

[11] We use a reimplementation that is a slight modification of Collins Model 1, with very similar performance, and which uses the TAG formalism of Carreras et al. (2008).

[12] Note that the models f_1 and f_2 were trained separately, using the methods described by Collins (2003) and Koo et al. (2008) respectively.

	Precision	Recall	F_1	Dep
Model 1	88.4	87.8	88.1	91.4
Koo08 Baseline	89.9	89.6	89.7	93.3
DD Combination	91.0	90.4	90.7	93.8

Table 2: Performance results for Section 23 of the WSJ Treebank. Model 1: a reimplementation of the generative parser of (Collins, 2002). Koo08 Baseline: Model 1 with a hard restriction to dependencies predicted by the discriminative dependency parser of (Koo et al., 2008). DD Combination: a model that maximizes the joint score of the two parsers. Dep shows the unlabeled dependency accuracy of each system.

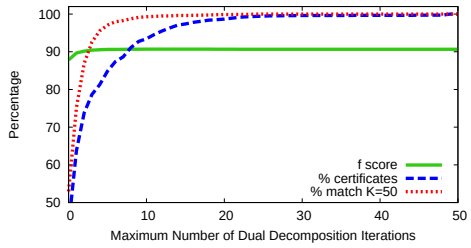

Figure 5: Performance on the parsing task assuming a fixed number of iterations K. f-score: accuracy of the method. % certificates: percentage of examples for which a certificate of optimality is provided. % match: percentage of cases where the output from the method is identical to the output when using $K = 50$.

Figure 5 shows performance of the approach as a function of K, the maximum number of iterations of dual decomposition. For this experiment, for cases where the method has not converged for $k \leq K$, the output from the algorithm is chosen to be the $y^{(k)}$ for $k \leq K$ that maximizes the objective function in Eq. 11. The graphs show that values of K less than 50 produce almost identical performance to $K = 50$, but with fewer cases giving certificates of optimality (with $K = 10$, the f-score of the method is 90.69%; with $K = 5$ it is 90.63%).

	Precision	Recall	F_1	POS Acc
Fixed Tags	88.1	87.6	87.9	96.7
DD Combination	88.7	88.0	88.3	97.1

Table 3: Performance results for Section 23 of the WSJ. Model 1 (Fixed Tags): a baseline parser initialized to the best tag sequence of from the tagger of Toutanova and Manning (2000). DD Combination: a model that maximizes the joint score of parse and tag selection.

7.2 Integrated Phrase-Structure Parsing and Trigram POS tagging

In a second experiment, we used dual decomposition to integrate the Model 1 parser with the Stanford max-ent trigram POS tagger (Toutanova and Manning, 2000), using a very similar algorithm to that described in section 4.1. We use the same training/dev/test split as in section 7.1. The two models were again trained separately.

We ran the algorithm with a limit of $K = 50$ iterations. Out of 2416 test examples, the algorithm found an exact solution in 98.9% of the cases. Table 1 gives statistics showing the speed of convergence for different examples: over 94% of the examples converge to an exact solution in 10 iterations or fewer. In terms of accuracy, we compare to a baseline approach of using the first-best tag sequence as input to the parser. The dual decomposition approach gives 88.3 F1 measure in recovering parse-tree constituents, compared to 87.9 for the baseline.

8 Conclusions

We have introduced dual-decomposition algorithms for inference in NLP, given formal properties of the algorithms in terms of LP relaxations, and demonstrated their effectiveness on problems that would traditionally be solved using intersections of dynamic programs (Bar-Hillel et al., 1964). Given the widespread use of dynamic programming in NLP, there should be many applications for the approach.

There are several possible extensions of the method we have described. We have focused on cases where two models are being combined; the extension to more than two models is straightforward (e.g., see Komodakis et al. (2007)). This paper has considered approaches for MAP inference; for closely related methods that compute approximate marginals, see Wainwright et al. (2005b).

A Fractional Solutions

We now give an example of a point $(\mu, \nu) \in \mathcal{Q}' \backslash \text{conv}(\mathcal{Q})$ that demonstrates that the relaxation $\mathcal{Q}'$ is strictly larger than $\text{conv}(\mathcal{Q})$. Fractional points such as this one can arise as solutions of the LP relaxation for worst case instances, preventing us from finding an exact solution.

Recall that the constraints for $\mathcal{Q}'$ specify that $\mu \in \text{conv}(\mathcal{Y})$, $\nu \in \text{conv}(\mathcal{Z})$, and $\mu(i,t) = \nu(i,t)$ for all $(i,t) \in \mathcal{I}_{\text{uni}}$. Since $\mu \in \text{conv}(\mathcal{Y})$, μ must be a convex combination of 1 or more members of $\mathcal{Y}$; a similar property holds for ν. The example is as follows. There are two possible parts of speech, A and B, and an additional non-terminal symbol X. The sentence is of length 3, $w_1\ w_2\ w_3$. Let ν be the convex combination of the following two tag sequences, each with probability 0.5: $w_1/A\ w_2/A\ w_3/A$ and $w_1/A\ w_2/B\ w_3/B$. Let μ be the convex combination of the following two parses, each with probability 0.5: $(X(A\ w_1)(X(A\ w_2)(B\ w_3)))$ and $(X(A\ w_1)(X(B\ w_2)(A\ w_3)))$. It can be verified that $\mu(i,t) = \nu(i,t)$ for all (i,t), i.e., the marginals for single tags for μ and ν agree. Thus, $(\mu, \nu) \in \mathcal{Q}'$.

To demonstrate that this fractional point is *not* in $\text{conv}(\mathcal{Q})$, we give parameter values such that this fractional point is optimal and all integral points (i.e., actual parses) are suboptimal. For the tagging model, set $\theta(AA \rightarrow A, 3) = \theta(AB \rightarrow B, 3) = 0$, with all other parameters having a negative value. For the parsing model, set $\theta(X \rightarrow A\ X, 1, 1, 3) = \theta(X \rightarrow A\ B, 2, 2, 3) = \theta(X \rightarrow B\ A, 2, 2, 3) = 0$, with all other rule parameters being negative. For this objective, the fractional solution has value 0, while all integral points (i.e., all points in $\mathcal{Q}$) have a negative value. By Theorem 5.2, the maximum of any linear objective over $\text{conv}(\mathcal{Q})$ is equal to the maximum over $\mathcal{Q}$. Thus, $(\mu, \nu) \notin \text{conv}(\mathcal{Q})$.

B Step Size

We used the following step size in our experiments. First, we initialized α_0 to equal 0.5, a relatively large value. Then we defined $\alpha_k = \alpha_0 * 2^{-\eta_k}$, where η_k is the number of times that $L(u^{(k')}) > L(u^{(k'-1)})$ for $k' \leq k$. This learning rate drops at a rate of $1/2^t$, where t is the number of times that the dual increases from one iteration to the next. See Koo et al. (2010) for a similar, but less aggressive step size used to solve a different task.

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

References

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

X. Carreras, M. Collins, and T. Koo. 2008. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proc CONLL*, pages 9–16.

X. Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proc. CoNLL*, pages 957–961.

M. Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. EMNLP*, page 8.

M. Collins. 2003. Head-driven statistical models for natural language parsing. In *Computational linguistics*, volume 29, pages 589–637.

G.B. Dantzig and P. Wolfe. 1960. Decomposition principle for linear programs. In *Operations research*, volume 8, pages 101–111.

J. Duchi, D. Tarlow, G. Elidan, and D. Koller. 2007. Using combinatorial optimization within max-product belief propagation. In *NIPS*, volume 19.

J. Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62.

A. Globerson and T. Jaakkola. 2007. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *NIPS*, volume 21.

N. Komodakis, N. Paragios, and G. Tziritas. 2007. MRF optimization via dual decomposition: Message-passing revisited. In *International Conference on Computer Vision*.

T. Koo, X. Carreras, and M. Collins. 2008. Simple semi-supervised dependency parsing. In *Proc. ACL/HLT*.

T. Koo, A.M. Rush, M. Collins, T. Jaakkola, and D. Sontag. 2010. Dual Decomposition for Parsing with Non-Projective Head Automata. In *Proc. EMNLP*, pages 63–70.

B.H. Korte and J. Vygen. 2008. *Combinatorial optimization: theory and algorithms*. Springer Verlag.

M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. In *Computational linguistics*, volume 19, pages 313–330.

R.K. Martin, R.L. Rardin, and B.A. Campbell. 1990. Polyhedral characterization of discrete dynamic programming. *Operations research*, 38(1):127–138.

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

R. McDonald, F. Pereira, K. Ribarov, and J. Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proc. HLT/EMNLP*, pages 523–530.

Angelia Nedić and Asuman Ozdaglar. 2009. Approximate primal solutions and rate analysis for dual subgradient methods. *SIAM Journal on Optimization*, 19(4):1757–1780.

B. Pang and L. Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proc. ACL*.

S. Riedel and J. Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proc. EMNLP*, pages 129–137.

D. Roth and W. Yih. 2005. Integer linear programming inference for conditional random fields. In *Proc. ICML*, pages 737–744.

Hanif D. Sherali and Warren P. Adams. 1994. A hierarchy of relaxations and convex hull characterizations for mixed-integer zero–one programming problems. *Discrete Applied Mathematics*, 52(1):83 – 106.

D.A. Smith and J. Eisner. 2008. Dependency parsing by belief propagation. In *Proc. EMNLP*, pages 145–156.

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

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-margin parsing. In *Proc. EMNLP*, pages 1–8.

K. Toutanova and C.D. Manning. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proc. EMNLP*, pages 63–70.

M. Wainwright and M. I. Jordan. 2008. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers Inc., Hanover, MA, USA.

M. Wainwright, T. Jaakkola, and A. Willsky. 2005a. MAP estimation via agreement on trees: message-passing and linear programming. In *IEEE Transactions on Information Theory*, volume 51, pages 3697–3717.

M. Wainwright, T. Jaakkola, and A. Willsky. 2005b. A new class of upper bounds on the log partition function. In *IEEE Transactions on Information Theory*, volume 51, pages 2313–2335.

C. Yanover, T. Meltzer, and Y. Weiss. 2006. Linear Programming Relaxations and Belief Propagation–An Empirical Study. In *The Journal of Machine Learning Research*, volume 7, page 1907. MIT Press.