Parse, Price and Cut—Delayed Column and Row Generation for Graph Based Parsers

Sebastian Riedel **David Smith** **Andrew McCallum**
Department of Computer Science
University of Massachusetts, Amherst
{riedel,dasmith,mccallum}@cs.umass.edu

Abstract

Graph-based dependency parsers suffer from the sheer number of higher order edges they need to (a) score and (b) consider during optimization. Here we show that when working with LP relaxations, large fractions of these edges can be pruned before they are fully scored—without any loss of optimality guarantees and, hence, accuracy. This is achieved by iteratively parsing with a subset of higher-order edges, adding higher-order edges that may improve the score of the current solution, and adding higher-order edges that are implied by the current best first order edges. This amounts to delayed column and row generation in the LP relaxation and is guaranteed to provide the optimal LP solution. For second order grandparent models, our method considers, or scores, no more than 6–13% of the second order edges of the full model. This yields up to an eightfold parsing speedup, while providing the same empirical accuracy and certificates of optimality as working with the full LP relaxation. We also provide a tighter LP formulation for grandparent models that leads to a smaller integrality gap and higher speed.

1 Introduction

Many problems in NLP, and structured prediction in general, can be cast as finding high-scoring structures based on a large set of candidate parts. For example, in second order graph-based dependency parsing (Kübler et al., 2009) we have to choose a quadratic number of first order and a cubic number of second order edges such that the graph is both high-scoring and a tree. In coreference, we have to select high-scoring clusters of mentions from an exponential number of candidate clusters, such that each mention is in exactly one cluster (Culotta et al., 2007). In segmentation of citation strings, we need to consider a quadratic number of possible segments such that every token is part of exactly one segment (Poon and Domingos, 2007).

What makes such problems challenging is the large number of possible parts to consider. This number not only affects the cost of search or optimization but also slows down the process of scoring parts before they enter the optimization problem. For example, the cubic grandparent edges in second-order dependency parsing slow down dynamic programs (McDonald and Pereira, 2006), belief propagation (Smith and Eisner, 2008) and LP solvers (Martins et al., 2009), since there are more value functions to evaluate, more messages to pass, or more variables to consider. But to even calculate the score for each part we need a cubic number of operations that usually involve expensive feature extraction. This step often becomes a major bottleneck in parsing, and structured prediction in general.

Candidate parts can often be heuristically pruned. In the case of dependency parsing, previous work has used coarse-to-fine strategies where simpler first order models are used to prune unlikely first order edges, and hence all corresponding higher order edges (Koo and Collins, 2010; Martins et al., 2009; Riedel and Clarke, 2006). While such methods can be effective, they are more convoluted, often require training of addition models as well as tuning of thresholding hyper-parameters, and usually provide no guarantees of optimality.

We present an approach that can solve problems with large sets of candidate parts without considering all of these parts in either optimization or scor-

Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 732–743, Jeju Island, Korea, 12–14 July 2012.

ing. And in contrast to most pruning heuristics, our algorithm can give certificates of optimality before having optimized over, or even scored, all parts. It does so without the need of auxiliary models or tuning of threshold parameters. This is achieved by a delayed column and row generation algorithm that iteratively solves an LP relaxation over a small subset of current candidate parts, and then finds new candidates that score highly and can be inserted into the current optimal solution without removing high scoring existing structure. The latter step subtracts from the cost of a part the price of resources the part requires, and is often referred as *pricing*. Sometimes parts may score highly after pricing, but are necessary in order to make the current solution feasible. We add such parts in a step that roughly amounts to violated *cuts* to the LP.

We illustrate our approach in terms of a second-order grandparent model for dependency parsing. We solve these models by iteratively parsing, pricing, and cutting. To this end we use a variant of the LP relaxation formulated by Martins et al. (2009). Our variant of this LP is designed to be amenable to column generation. It also turns out to be a tighter outer bound that leads to fewer fractional solutions and faster runtimes. To find high scoring grandparent edges without explicitly enumerating all of them, we prune out a large fraction using factorized upper bounds on grandparent scores.

Our parse, price and cut algorithm is evaluated using a non-projective grandparent model on three languages. Compared to a brute force approach of solving the full LP, we only score about 10% of the grandparent edges, consider only 8% in optimization, and so observe an increase in parsing speed of up to 750%. This is possible without loss of optimality, and hence accuracy. We also find that our extended LP formulation leads to a 15% reduction of fractional solutions, up to 12 times higher speed, and generally higher accuracy when compared to the grandparent formulation of Martins et al. (2009).

2 Graph-Based Dependency Parsing

Dependency trees are representations of the syntactic structure of a sentence (Nivre et al., 2007). They determine, for each token of a sentence, the syntactic head the token is modifying. As a lightweight alternative to phrase-based constituency trees, dependency representations have by now seen widespread use in the community in various domains such as question answering, machine translation, and information extraction.

To simplify further exposition, we now formalize the task, and mostly follow the notation of Martins et al. (2009). Consider a sentence $\mathbf{x} = \langle t_0, t_1, \ldots, t_n \rangle$ where $t_1, \ldots, t_n$ correspond to the n tokens of the sentence, and t_0 is an artificial root token. Let $V \triangleq \{0, \ldots, n\}$ be a set of vertices corresponding to the tokens in $\mathbf{x}$, and $\mathcal{C} \subseteq V \times V$ a set of candidate directed edges. Then a directed graph $y \subseteq \mathcal{C}$ is a *legal dependency parse* if and only if it is a tree over V rooted at vertex 0. Given a sentence $\mathbf{x}$, we use $\mathcal{Y}$ to denote the set of its legal parses. Note that all of the above definitions depend on $\mathbf{x}$, but for simplicity we omit this dependency in our notation.

2.1 Arc-Factored Models

Graph-based models define parametrized scoring functions that are trained to discriminate between correct and incorrect parse trees. So called *arc-factored* or *first order* models are the most basic variant of such functions: they assess the quality of a tree by scoring each edge in isolation (McDonald et al., 2005b; McDonald et al., 2005a). Formally, arc-factored models are scoring functions of the form

$$s\left(y; \mathbf{x}, \mathbf{w}\right) = \sum_{\langle h,m \rangle \in y} s_{\langle h,m \rangle}\left(\mathbf{x}, \mathbf{w}\right) \qquad (1)$$

where $\mathbf{w}$ is a weight vector and $s_{\langle h,m \rangle}\left(\mathbf{x}, \mathbf{w}\right)$ scores the edge $\langle h, m \rangle$ with respect to sentence $\mathbf{x}$ and weights $\mathbf{w}$. From here on we will omit both $\mathbf{x}$ and $\mathbf{w}$ from our notation if they are clear from the context.

Given such a scoring function, parsing amounts to solving:

$$\begin{aligned} \underset{y}{\text{maximize}} \quad & \sum_{\langle h,m \rangle \in y} s_{\langle h,m \rangle} \\ \text{subject to} \quad & y \in \mathcal{Y}. \end{aligned} \qquad (2)$$

2.2 Higher Order Models

Arc-factored models cannot capture higher order dependencies between two or more edges. Higher order models remedy this by introducing scores for larger configurations of edges appearing in the

tree (McDonald and Pereira, 2006). For example, in *grandparent* models, the score of a tree also includes a score $s^{\text{gp}}_{\langle g,p,c\rangle}$ for each grandparent-parent-child triple $\langle g, p, c\rangle$:

$$s(y) = \sum_{\langle h,m\rangle \in y} s_{\langle h,m\rangle} + \sum_{\langle g,p\rangle \in y, \langle p,c\rangle \in y} s^{\text{gp}}_{\langle g,p,c\rangle} \quad (3)$$

There are other variants of higher order models that include, in addition to grandparent triples, pairs of siblings (adjacent or not) or third order edges. However, to illustrate our approach we will focus on grandparent models and note that most of what we present can be generalized to other higher order models.

2.3 Feature Templates

For our later exposition the factored and parametrized nature of the scoring functions will be crucial. In the following we therefore illustrate this property in more detail.

The scoring functions for arcs or higher order edges usually decompose into a sum of feature *template* scores. For example, the grandparent edge score $s^{\text{gp}}_{\langle g,p,c\rangle}$ is defined as

$$s^{\text{gp}}_{\langle g,p,c\rangle} \triangleq \sum_{t \in \mathcal{T}^{\text{gp}}} s^{\text{gp},t}_{\langle g,p,c\rangle} \quad (4)$$

where $\mathcal{T}^{\text{gp}}$ is the set of grandparent templates, and each template $t \in \mathcal{T}^{\text{gp}}$ defines a scoring function $s^{\text{gp},t}_{\langle g,p,c\rangle}$ to assess a specific property of the grandparent-parent-child edge $\langle g, p, c\rangle$.

The template scores again decompose. Considering grandparent scores, we get

$$s^{t}_{\langle g,p,c\rangle} \triangleq \mathbf{w}_t^\top \mathbf{f}^t\left(h^t_g, h^t_p, h^t_c, d^t_{g,p,c}\right) \quad (5)$$

where h^t_i is an attribute of token t_i, say $h^{101}_i = \text{Part-of-Speech}(t_i)$. The term $d^t_{g,p,c}$ corresponds to a representation of the relation between tokens corresponding to g, p and g. For example, for template 101 it could return their relative positions to each other:

$$d^{101}_{g,p,c} \triangleq \langle \mathbb{I}[g > p], \mathbb{I}[g > c], \mathbb{I}[p > c]\rangle . \quad (6)$$

The feature function $\mathbf{f}^t$ maps the representations of g, p and c into a vector space. For the purposes of our work this mapping is not important, and hence we omit details.

2.4 Learning

The scoring functions we consider are parametrized by a family of per-template weight vectors $\mathbf{w} = \langle \mathbf{w}_t \rangle_{t \in \mathcal{T}}$. During learning we need to estimate $\mathbf{w}$ such that our scoring functions learns to differentiate between correct and incorrect parse trees. This can be achieved in many ways: large margin training, maximizing conditional likelihood, or variants in between. In this work we follow Smith and Eisner (2008) and train the models with stochastic gradient descent on the conditional log-likelihood of the training data, using belief propagation in order to calculate approximate gradients.

3 LP and ILP Formulations

Riedel and Clarke (2006) showed that dependency parsing can be framed as Integer Linear Program (ILP), and efficiently solved using an off-the-shelf optimizer if a cutting plane approach is used.[1] Compared to tailor made dynamic programs, such generic solvers give the practitioner more modeling flexibility (Martins et al., 2009), albeit at the cost of efficiency. Likewise, compared to approximate solvers, ILP and Linear Program (LP) formulations can give strong guarantees of optimality. The study of Linear LP relaxations of dependency parsing has also lead to effective alternative methods for parsing, such as dual decomposition (Koo et al., 2010; Rush et al., 2010). As we see later, the capability of LP solvers to calculate dual solutions is also crucial for efficient and exact pruning. Note, however, that dynamic programs provide dual solutions as well (see section 4.5 for more details).

3.1 Arc-Factored Models

To represent a parse $y \in \mathcal{Y}$ we first introduce an vector of variables $\mathbf{z} \triangleq \langle z_a \rangle_a$ where z_a is 1 if $a \in y$ and 0 otherwise. With this representation parsing amounts to finding a vector $\mathbf{z}$ that corresponds to a legal parse tree and that maximizes $\sum_a z_a s_a$. One way to achieve this is to search through the *convex hull* of all legal incidence vectors, knowing that any linear objectives would take on its maximum on one of the hull's vertices. We will use $\mathcal{Z}$ to denote this convex hull of incidence vectors of legal parse trees,

[1] Such as the highly efficient and free-for-academic-use Gurobi solver.

and call $\mathcal{Z}$ the *arborescence polytope* (Martins et al., 2009). The Minkowski-Weyl theorem tells us that $\mathcal{Z}$ can be represented as an intersection of halfspaces, or constraints, $\mathcal{Z} = \{\mathbf{z}|\mathbf{A}\mathbf{z} \leq \mathbf{b}\}$. Hence optimal dependency parsing, in theory, can be addressed using LPs.

However, it is difficult to describe $\mathcal{Z}$ with a *compact* number of constraints and variables that lend themselves to efficient optimization. In general we therefore work with *relaxations*, or outer bounds, on $\mathcal{Z}$. Such outer bounds are designed to cut off all illegal *integer* solutions of the problem, but still allow for *fractional* solutions. In case the optimum is achieved at an integer vertex of the outer bound, it is clear that we have found the optimal solution to the original problem. In case we find a fractional point, we need to map it onto $\mathcal{Z}$ (e.g., by projection or rounding). Alternatively, we can use the outer bound together with 0/1 constraints on $\mathbf{z}$, and then employ an ILP solver (say, branch-and-bound) to find the true optimum. Given the NP-hardness of ILP, this will generally be slow.

In the following we will present the outer bound $\bar{\mathcal{Z}} \supseteq \mathcal{Z}$ proposed by Martins et al. (2009). Compared to the representation Riedel and Clarke (2006), this bound has the benefit a small polynomial number of constraints. Note, however, that often exponentially many constraints can be efficiently handled if polynomial separation algorithms exists, and that such representations can lead to tighter outer bounds.

The constraints we employ are:

No Head For Root In a dependency tree the root node never has a head. While this could be captured through linear constraints, it is easier to simply restrict the candidate set $\mathcal{C}$ to never contain edges of the form $\langle \cdot, 0 \rangle$.

Exactly One Head for Non-Roots Any non-root token has to have exactly one head token. We can enforce this property through the set of constraints:

$$m > 0 : \sum_h z_{\langle h,m \rangle} = 1. \qquad \text{(OneHead)}$$

No Cycles A parse tree cannot have cycles. This is equivalent, together with the head constraints above, to enforcing that the tree be fully connected. Martins et al. (2009) capture this connectivity constraint using a *single commodity flow formulation*. This requires the introduction of flow variables $\boldsymbol{\phi} \triangleq \langle \phi_a \rangle_{a \in \mathcal{C}}$. By enforcing that token 0 has n outgoing flow,

$$\sum_{m>0} \phi_{\langle 0,m \rangle} = n, \qquad \text{(Source)}$$

that any other token *consumes one unit of flow*,

$$t > 0 : \sum_h \phi_{\langle h,t \rangle} - \sum_{m>0} \phi_{\langle t,m \rangle} = 1 \qquad \text{(Consume)}$$

and that *flow is zero on disabled arcs*

$$\phi_{\langle h,m \rangle} \leq n z_{\langle h,m \rangle}, \qquad \text{(NoFlow)}$$

connectivity can be ensured.

Assuming we have such a representation, parsing with an LP relaxation amounts to solving

$$\begin{aligned} \underset{\mathbf{z} \geq \mathbf{0}}{\text{maximize}} \quad & \sum_{a \in A} z_a s_a \\ \text{subject to} \quad & \mathbf{A} \begin{bmatrix} \mathbf{z} \\ \boldsymbol{\phi} \end{bmatrix} \leq \mathbf{b}. \end{aligned} \qquad (7)$$

3.2 Higher Order Models

The 1st-Order LP can be easily extended to capture second (or higher) order models. For for the case of grandparent models, this amounts to introducing another class of variables, $z^{\text{gp}}_{g,p,c}$, that indicate if the parse contains both the edge $\langle g, p \rangle$ and the edge $\langle p, c \rangle$. With the help of the indicators $\mathbf{z}^{\text{gp}}$ we can represent the second order objective as a linear function. We now need an outer bound on the convex hull of vectors $\langle \mathbf{z}, \mathbf{z}^{\text{gp}} \rangle$ where $\mathbf{z}$ is a legal parse tree and $\mathbf{z}^{\text{gp}}$ is a consistent set of grandparent indicators. We will refer to this convex hull as the *grandparent polytope* $\mathcal{Z}^{\text{gp}}$.

We can re-use the constraints $\mathbf{A}$ of section 3.1 to ensure that $\mathbf{z}$ is in $\mathcal{Z}$. To make sure $\mathbf{z}^{\text{gp}}$ is consistent with $\mathbf{z}$, Martins et al. (2009) linearize the equivalence $z^{gp}_{g,p,c} \Leftrightarrow z_{g,p} \wedge z_{p,c}$ we know to hold for legal incidence vectors, yielding

$$g, p, c : z_{\langle g,p \rangle} + z_{\langle p,c \rangle} - z^{\text{gp}}_{\langle g,p,c \rangle} \leq 1 \qquad \text{(ArcGP)}$$

and

$$g, p, c : z_{\langle g,p \rangle} \geq z^{\text{gp}}_{\langle g,p,c \rangle}, z_{\langle p,c \rangle} \geq z^{\text{gp}}_{\langle g,p,c \rangle} \qquad \text{(GPArc)}$$

There are additional constraints we know to hold in $\mathcal{Z}^{\text{gp}}$. First, we know that for any active edge $\langle p, c \rangle \in$

y with $p > 0$ there is exactly one grandparent edge $\langle g, p, c\rangle$. Likewise, for an inactive edge $\langle p, c\rangle \notin y$ there must be no grandparent edge $\langle g, p, c\rangle$. This can be captured through the constraint:

$$p > 0, c : \sum_g z^{\text{gp}}_{\langle g,p,c\rangle} = z_{\langle p,c\rangle}. \qquad \text{(OneGP)}$$

We also know that if an edge $\langle g, p\rangle$ in inactive, there must not be any grandparent edge $\langle g, p, c\rangle$ that goes through $\langle g, p\rangle$:

$$g, p : \sum_c z^{\text{gp}}_{\langle g,p,c\rangle} \leq n z_{\langle g,p\rangle}. \qquad \text{(NoGP)}$$

It can be easily shown that for *integer* solutions the constraints ArcGP and GPArc of Martins et al. (2009) are sufficient conditions for consistency between $\mathbf{z}$ and $\mathbf{z}^{\text{gp}}$. It can equally be shown that the same holds for the constraints OneGP and NoGP. However, when working with LP relaxations, the two polytopes have different fractional vertices. Hence, by combining both constraint sets, we can get a tighter outer bound on the grandparent polytope $\mathcal{Z}^{\text{gp}}$. In section 6 we show empirically that this combined polytope in fact leads to fewer fractional solutions. Note that when using the union of all four types of constraints, the NoGP constraint is implied by the constraint GPArc (left) by summing over c on both sides, and can hence be omitted.

4 Parse, Price and Cut

We now introduce our parsing algorithm. To this end, we first give a general description of column and row generation for LPs; then, we illustrate how these techniques can be applied to dependency parsing.

4.1 Column and Row Generation

LPs often have too many variables and constraints to be efficiently solved. In such cases delayed column and row generation can substantially reduce runtime by lazily adding variables only when needed (Gilmore and Gomory, 1961; Lübbecke and Desrosiers, 2004).

To illustrate column and row generation let us consider the following general primal LP and its corresponding dual problem:

Primal		*Dual*	
$\underset{\mathbf{z}\geq\mathbf{0}}{\text{maximize}}$	$\mathbf{s}^\intercal\mathbf{z}$	$\underset{\boldsymbol{\lambda}\geq\mathbf{0}}{\text{minimize}}$	$\boldsymbol{\lambda}^\intercal\mathbf{b}$
subject to	$\mathbf{A}\mathbf{z} \leq \mathbf{b}$	subject to	$\mathbf{A}^\intercal\boldsymbol{\lambda} \geq \mathbf{s}$.

Say you are given a primal feasible $\mathbf{z}'$ and a dual feasible $\boldsymbol{\lambda}'$ for which *complementary slackness* holds: for all variables i we have $z'_i > 0 \Rightarrow s_i = \sum_j \lambda'_j a_{i,j}$ and for all constraints j we have $\lambda'_j > 0 \Rightarrow b_j = \sum_i z'_i a_{i,j}$. In this case it is easy to show that $\mathbf{z}'$ is an optimal primal solution, $\boldsymbol{\lambda}'$ and optimal dual solution, and that both objectives meet at these values (Bertsekas, 1999).

The idea behind delayed column and row generation is to only consider a small subset of variables (or *columns*) I and subset of constraints (or *rows*) J. Optimizing over this restricted problem, either with an off-the-shelf solver or a more specialized method, yields the pair $(\mathbf{z}'_I, \boldsymbol{\lambda}'_J)$ of partial primal and dual solutions. This pair is feasible and complementary with respect to variables I and constraints J. We can extend it to a solution $(\mathbf{z}', \mathbf{y}')$ over all variables and constraints by heuristically setting the remaining primal and dual variables. If it so happens that $(\mathbf{z}', \mathbf{y}')$ is feasible and complementary for all variables and constraints, we have found the optimal solution. If not, we add the constraints and variables for which feasibility and slackness are violated, and resolve the new partial problem.

In practice, the uninstantiated primal and dual variables are often set to 0. In this case complementary slackness holds trivially, and we only need to find violated primal and dual constraints. For primal constraints, $\sum_i z_i a_{i,j} \leq b_i$, searching for violating constraints j is the well-known *separation* step in cutting plane algorithms. For the dual constraints, $\sum_j \lambda_j a_{i,j} \geq s_i$, the same problem is referred to as *pricing*. Pricing is often framed as searching for all, or some, variables i with positive *reduced cost* $r_i \triangleq s_i - \sum_j \lambda_j a_{i,j}$. Note that while these problems are, naturally, dual to each other, they can have very different flavors. When we assess dual constraints we need to calculate a cost s_i for variable i, and usually this cost would be different for different i. For primal constraints the corresponding right-hand-sides are usually much more homogenous.

Algorithm 1 Parse, Price and Cut.

Require: Initial candidate edges and hyperedges P.
Ensure: The optimal $\mathbf{z}$.

```
repeat
    z, λ ← parse(P)
    N    ← price(λ)
    M    ← cut(z)
    P    ← P ∪ N ∪ M
until N = ∅ ∧ M = ∅
return z
```

The reduced cost $r_i = s_i - \sum_j \lambda_j a_{i,j}$ has several interesting interpretations. First, intuitively it measures the score we could gain by setting $z_i = 1$, and subtracts an estimate of what we would loose because $z_i = 1$ may compete with other variables for shared resources (constraints). Second, it corresponds to the coefficient of z_i in the Lagrangian $L(\boldsymbol{\lambda}, \mathbf{z}) \triangleq \mathbf{s}^\intercal \mathbf{z} + \boldsymbol{\lambda}[\mathbf{b} - \mathbf{A}\mathbf{z}]$. For any $\boldsymbol{\lambda}$, $U_{z_i=k} = \max_{\mathbf{z} \geq 0, z_i = k} L(\boldsymbol{\lambda}, \mathbf{z})$ is an upper bound on the best possible primal objective with $z_i = k$. This means that $r_i = U_{z_i=1} - U_{z_i=0}$ is the difference between an upper bound that considers $z_i = 1$, and one that considers $z_i = 0$. The tighter the bound $U_{z_i=0}$ is, the closer r_i is to an upper bound on the maximal increase we can get for setting z_i to 1. At convergence of column generation, complementary slackness guarantees that $U_{z_i=0}$ is tight for all $z'_i = 0$, and hence r_i is a true an upper bound.

4.2 Application to Dependency Parsing

The grandparent formulation in section 3.2 has a cubic number of variables $z_{\langle g,p,c \rangle}$ as well as a cubic number of constraints. For longer sentences this number can slow us down in two ways. First, the optimizer works with a large search space, and will naturally become slower. Second, for every grandparent edge we need to calculate the score $s_{\langle g,p,c \rangle}$, and this calculation can often be a major bottleneck, in particular when using complex feature functions. To overcome this bottleneck, our parse, price and cut algorithm, as shown in algorithm 1, uses column and row generation. In particular, it lazily instantiates the grandparent edge variables $z^{\text{gp}}_{\langle g,p,c \rangle}$, and the corresponding cubic number of constraints. All uninstantiated variables are implicitly set to 0.

The algorithm requires some initial set of variables to start with. In our case this set P contains all first-order edges $\langle h, m \rangle$ in the candidate set $\mathcal{C}$, and for each of these one grandparent edge $\langle 0, h, m \rangle$. The primary purpose of these grandparent edges is to ensure feasibility of the OneGP constraints.

In step 2, the algorithm parses with the current set of candidates P by solving the corresponding LP relaxation. The LP contains all columns and constraints that involve the edges and grandparent edges of P. The solver returns both the best primal solution $\mathbf{z}$ (for both edges and grandparents), and a complementary dual solution $\boldsymbol{\lambda}$.

In step 3 the dual variables $\boldsymbol{\lambda}$ are used to find uninstantiated grandparent edges $\langle g, p, c \rangle$ with positive reduced cost. The price routine returns such edges in N. In step 4 the primal solution is inspected for violations of constraint ArcGP. The cut routine performs this operation, and returns M, the set of edges $\langle g, p, c \rangle$ that violate ArcGP.

In step 5 the algorithm converges if no more constraint violations, or promising new columns, can be found. If there have been violations ($M \neq \emptyset$) or promising columns ($N \neq \emptyset$), steps 2 to 4 are repeated, with the newly found parts added to the problem. Note that LP solvers can be efficiently *warm-started* after columns and rows have been added, and hence the cost of calls to the solver in step 2 is substantially reduced after the first iteration.

4.3 Pricing

In the pricing step we need to efficiently find a set of grandparent edge variables $z^{\text{gp}}_{\langle g,p,c \rangle}$ with positive reduced cost, or the empty set if no such variables exist. Let $\lambda^{\text{OneGP}}_{\langle p,c \rangle}$ be the dual variables for the OneGP constraints and $\lambda^{\text{NoGP}}_{\langle g,p \rangle}$ the duals for constraints NoGP. Then for the reduced cost of $z^{\text{gp}}_{\langle g,p,c \rangle}$ we know that:

$$r_{\langle g,p,c \rangle} = s_{\langle g,p,c \rangle} - \lambda^{\text{OneGP}}_{\langle p,c \rangle} - \lambda^{\text{NoGP}}_{\langle g,p \rangle}. \qquad (8)$$

Notice that the duals for the remaining two constraints ArcGP and GPArc do not appear in this equation. This is valid because we can safely set their duals to zero without violating dual feasibility or complementary slackness of the solution returned by the solver.

4.3.1 Upper Bounds for Efficient Pricing

A naive pricing implementation would exhaustively iterate over all $\langle g, p, c\rangle$ and evaluate $r_{\langle g,p,c\rangle}$ for each. In this case we can still substantially reduce the number of grandparent variables that enter the LP, provided many of these variables have non-positive reduced cost. However, we still need to calculate the score $s_{\langle g,p,c\rangle}$ for each $\langle g, p, c\rangle$, an expensive operation we hope to avoid. In the following we present an upper bound on the reduced cost, $\bar{r}^{\text{gp}}_{\langle g,p,c\rangle} \geq r^{\text{gp}}_{\langle g,p,c\rangle}$, which decomposes in a way that allows for more efficient search. Using this bound, we find all new grandparent edges $\bar{N}$ for which this upper bound is positive:

$$\bar{N} \leftarrow \left\{ \langle g, p, c\rangle \,|\bar{r}^{\text{gp}}_{\langle g,p,c\rangle} > 0 \right\}. \tag{9}$$

Next we prune away all but the grandparent edges for which the exact reduced cost is positive:

$$N \leftarrow \bar{N} \setminus \{e : r^{\text{gp}}_{e} > 0\}. \tag{10}$$

Our bound $\bar{r}^{\text{gp}}_{\langle g,p,c\rangle}$ on the reduced cost of $\langle g, p, c\rangle$ is based on an upper bound $\bar{s}^{\text{gp}}_{\langle g,p,\cdot\rangle} \geq \max_c s^{\text{gp}}_{\langle g,p,c\rangle}$ on the grandparent score involving $\langle g, p\rangle$ as grandparent and parent, and the bound $\bar{s}^{\text{gp}}_{\langle \cdot,p,c\rangle} \geq \max_g s^{\text{gp}}_{\langle g,p,c\rangle}$ on the grandparent score involving $\langle p, c\rangle$ as parent and child. Concretely, we have

$$\bar{r}^{\text{gp}}_{\langle g,p,c\rangle} \triangleq \min\left(\bar{s}^{\text{gp}}_{\langle g,p,\cdot\rangle}, \bar{s}^{\text{gp}}_{\langle \cdot,p,c\rangle}\right) - \lambda^{\text{OneGP}}_{\langle p,c\rangle} - \lambda^{\text{NoGP}}_{\langle g,p\rangle}. \tag{11}$$

To find edges $\langle g, p, c\rangle$ for which this bound is positive, we can filter out all edges $\langle p, c\rangle$ such that $s^{\text{gp}}_{\langle \cdot,p,c\rangle} - \lambda^{\text{OneGP}}_{\langle p,c\rangle}$ is non-positive. This is possible because NoGP is a $\leq$ constraint and therefore $\lambda^{\text{NoGP}}_{\langle g,p\rangle} \geq 0$.[2] Hence $\bar{r}^{\text{gp}}_{\langle g,p,c\rangle}$ is at most $\bar{s}^{\text{gp}}_{\langle \cdot,p,c\rangle} - \lambda^{\text{OneGP}}_{\langle p,c\rangle}$. This filtering step cuts off a substantial number of edges, and is the main reason why can avoid scoring all edges.

Next we filter, for each remaining $\langle p, c\rangle$, all possible grandparents g according to the definition of $\bar{r}^{\text{gp}}_{\langle g,p,c\rangle}$. This again allows us to avoid calling the grandparent scoring function on $\langle g, p, c\rangle$, and yields the candidate set $\bar{N}$. Only if $\bar{r}^{\text{gp}}_{\langle g,p,c\rangle}$ is positive do we have to evaluate the exact reduced cost and score.

[2]Notice that in section 4.1 we discussed the LP dual in case were all constraints are inequalities. When equality constraints are used, the corresponding dual variables have no sign constraints. Hence we could not make the same argument for $\lambda^{\text{OneGP}}_{\langle p,c\rangle}$.

4.3.2 Upper Bounds on Scores

What remains to be done is the calculation of upper bounds $\bar{s}^{\text{gp}}_{\langle g,p,\cdot\rangle}$ and $\bar{s}^{\text{gp}}_{\langle \cdot,p,c\rangle}$. Our bounds factor into per-template bounds according to the definitions in section 2.3. In particular, we have

$$\bar{s}^{\text{gp}}_{\langle \cdot,p,c\rangle} \triangleq \sum_{t \in \mathcal{T}^{\text{gp}}} \bar{s}^{\text{gp},t}_{\langle \cdot,p,c\rangle} \tag{12}$$

where $\bar{s}^{t}_{\langle \cdot,p,c\rangle}$ is a per-template upper bound defined as

$$\bar{s}^{\text{gp},t}_{\langle \cdot,p,c\rangle} \triangleq \max_{\substack{v \in \text{range}(h^t) \\ e \in \text{range}(d^t)}} \mathbf{w}_t^\top \mathbf{f}^t\left(v, h^t_p, h^t_c, e\right). \tag{13}$$

That is, we maximize over all possible attribute values v any token g could have, and any possible relation e a token g can have to p and c.

Notice that these bounds can be calculated offline, and hence amortize after deployment of the parser.

4.3.3 Tightening Duals

To price variables, we use the duals returned by the solver. This is a valid default strategy, but may lead to $\boldsymbol{\lambda}$ with overcautious reduced costs. Note, however, that we can arbitrary alter $\boldsymbol{\lambda}$ to minimize reduced costs of uninstantiated variables, as long as we ensure that feasibility and complementary slackness are maintained for the instantiated problem.

We use this flexibility for increasing $\lambda^{\text{OneGP}}_{\langle p,c\rangle}$, and hence lowering reduced costs $z^{\text{gp}}_{\langle g,p,c\rangle}$ for all tokens c. Assume that $z_{\langle p,c\rangle} = 0$ and let $r_{\langle p,c\rangle} = \lambda^{\text{OneGP}}_{\langle p,c\rangle} + K$ be the current reduced cost for $z_{\langle p,c\rangle}$ in the instantiated problem. Here K is a value depending on $s_{\langle p,c\rangle}$ and the remaining constraints $z_{\langle p,c\rangle}$ is involved in.

We know that $r_{\langle p,c\rangle} \leq 0$ due to dual feasibility and hence $r_{\langle p,c\rangle}$ may be 0, but note that $r_{\langle p,c\rangle} < 0$ in many cases. In such cases we can increase $\lambda^{\text{OneGP}}_{\langle p,c\rangle}$ to $-K$ and get $r_{\langle p,c\rangle} = 0$. With respect to $z_{\langle p,c\rangle}$ this maintains dual feasibility (because $r_{\langle p,c\rangle} \leq 0$) and complementary slackness (because $z_{\langle p,c\rangle} = 0$). Furthermore, with respect to the $z^{\text{gp}}_{\langle g,p,c\rangle}$ for all tokens c this also maintains feasibility (because the increased $\lambda^{\text{OneGP}}_{\langle p,c\rangle}$ appears with negative sign in 8) and complementary slackness (because $z^{\text{gp}}_{\langle g,p,c\rangle} = 0$ due to $z_{\langle p,c\rangle} = 0$).

4.4 Separation

What happens if both $z_{\langle g,p \rangle}$ and $z_{\langle p,c \rangle}$ are active while $z^{\text{gp}}_{\langle g,p,c \rangle}$ is still implicitly set to 0? In this case we violate constraint ArcGP. We could remedy this by adding the cut $z_{\langle g,p \rangle} + z_{\langle p,c \rangle} \leq 1$, resolve the LP, and then use the dual variable corresponding to this constraint to get an updated reduced cost $r_{\langle g,p,c \rangle}$. However, in practice we found this does not happen as often, and when it does, it is cheaper for us to add the corresponding column $r_{\langle g,p,c \rangle}$ right away instead of waiting to the next iteration to price it.

To find all pairs of variables for $z_{\langle g,p \rangle} + z_{\langle p,c \rangle} \leq 1$ is violated, we first filter out all edges $\langle h, m \rangle$ for which $z_{\langle h,m \rangle} = 0$ as these automatically satisfy any ArcGP constraint they appear in. Now for each $z_{\langle g,p \rangle} > 0$ all $z_{\langle p,c \rangle} > 0$ are found, and if their sum is larger than 1, the corresponding grandparent edge $\langle g, p, c \rangle$ is returned in the result set.

4.5 Column Generation in Dynamic Programs

Column and Row Generation can substantially reduce the runtime of an off-the-shelf LP solver, as we will find in section 6. Perhaps somewhat surprisingly, it can also be applied in the context of dynamic programs. It is well known that for each dynamic program there is an equivalent polynomial LP formulation (Martin et al., 1990). Roughly speaking, in this formulation primal variables correspond to state transitions, and dual variables to value functions (e.g., the forward scores in the Viterbi algorithm).

In pilot studies we have already used DCG to speed up (exact) Viterbi on linear chains (Belanger et al., 2012). We believe it could be equally applied to dynamic programs for higher order dependency parsing.

5 Related Work

Our work is most similar in spirit to the relaxation method presented by Riedel and Smith (2010) that incrementally adds second order edges to a graphical model based on a gain measure—the analog of our reduced cost. However, they always score every higher order edge, and also provide no certificates of optimality.

Several works in parsing, and in MAP inference in general, perform some variant of row generation (Riedel and Clarke, 2006; Tromble and Eisner, 2006; Sontag and Jaakkola, 2007; Sontag et al., 2008). However, none of the corresponding methods lazily add columns, too. The cutting plane method of Riedel (2008) can omit columns, but only if their coefficient is negative. By using the notion of reduced costs we can also omit columns with positive coefficient. Niepert (2010) applies column generation, but his method is limited to the case of k-Bounded MAP Inference.

Several ILP and LP formulations of dependency parsing have been proposed. Our formulation is inspired by Martins et al. (2009), and hence uses fewer constraints than Riedel and Clarke (2006). For the case of grandparent edges, our formulation also improves upon the outer bound of Martins et al. (2009) in terms of speed, tightness, and utility for column generation. Other recent LP relaxations are based on dual decomposition (Rush et al., 2010; Koo et al., 2010; Martins et al., 2011). These relaxations allow the practitioner to utilize tailor-made dynamic programs for tractable substructure, but still every edge needs to be scored. Given that column generation can also be applied in dynamic programs (see section 4.5), our algorithm could in fact accelerate dual decomposition parsing as well.

Pruning methods are a major part of many structured prediction algorithms in general, and of parsing algorithms in particular (Charniak and Johnson, 2005; Martins et al., 2009; Koo and Collins, 2010; Rush and Petrov, 2012). Generally these methods follow a coarse-to-fine scheme in which simpler models filter out large fractions of edges. Such methods are effective, but require tuning of threshold parameters, training of additional models, and generally lead to more complex pipelines that are harder to analyze and have fewer theoretical guarantees.

A* search (Ahuja et al., 1993) has been used to search for optimal parse trees, for example by Klein and Manning (2003) or, for dependency parsing, by Dienes et al. (2003). There is a direct relation between both A* and Column Generation based on an LP formulation of the shortest path problem. Roughly speaking, in this formulation any feasible dual assignments correspond to a consistent (and thus admissible) heuristic, and the corresponding reduced costs can be used as edge weights. Run-

ning Dijkstra's algorithm with these weights then amounts to A*. Column generation for the shortest path problem can then be understood as a method to lazily construct a consistent heuristic. In every step this method finds edges for which consistency is violated, and updates the heuristic such that all these edges are consistent.

6 Experiments

We claim that LP relaxations for higher order parsing can be solved without considering, and scoring, all candidate higher order edges. In practice, how many grandparent edges do we need to score, and how many do we need to add to the optimization problem? And what kind of reduction in runtime does this reduction in edges lead to?

We have also pointed out that our outer bound on the grandparent polytope of legal edge and grandparent vectors is tighter than the one presented by Martins et al. (2009). What effect does this bound have on the number of fractional solutions and the overall accuracy?

To answer these questions we will focus on a set of non-projective grandparent models, but point out that our method and formulation can be easily extended to projective parsing as well as other types of higher order edges. We use the Danish test data of Buchholz and Marsi (2006) and the Italian and Hungarian test datasets of Nivre et al. (2007).

6.1 Impact of Price and Cut

Table 1 compares brute force optimization (BF) with the full model, in spirit of Martins et al. (2009), to running parse, price and cut (PPC) on the same model. This model contains all constraints presented in 3.2. The table shows the average number of parsed sentences per second, the average objective, number of grandparent edges scored and added, all relative to the brute force approach. We also present the average unlabeled accuracy, and the percentage of sentences with integer solutions. This number shows us how often we not only found the optimal solution to the LP relaxation, but also the optimal solution to the full ILP.

We first note that both systems achieve the same objective, and therefore, also the same accuracy. This is expected, given that column and row generation are known to yield optimal solutions. Next we see that the number of grandparent edges scored and added to the problem is reduced to 5–13% of the full model. This leads to up to 760% improvement in speed. This improvement comes for free, without any sacrifice in optimality or guarantees. We also notice that in all cases at least 97% of the sentences have no fractional solutions, and are therefore optimal even with respect to the ILP. Table 1 also shows that our bounds on reduced costs are relatively tight. For example, in the case of Italian we score only one percent more grandparent edges than we actually need to add.

Our fastest PCC parser processes about one sentence per second. This speed falls below the reported numbers of Martins et al. (2009) of about 0.6 seconds per sentence. Crucially, however, in contrast to their work, our speed is achieved without any first-order pruning. In addition, we expect further improvements in runtime by optimizing the implementation of our pricing algorithm.

6.2 Tighter Grandparent Polytope

To investigate how the additional grandparent constraints in section 3.2 help, we compare three models, this time without PPC. The first model follows Martins et al. (2009) and uses constraints ArcGP and GPArc only. The second model uses only constraints OneGP and NoGP. The final model incorporates all four constraints.

Table 2 shows speed relative to the baseline model with constraints ArcGP and GPArc, as well as the percentage of integer solutions and the average unlabeled accuracy—all for the Italian and Hungarian datasets. We notice that the full model has less fractional solutions than the partial models, and either substantially (Italian) or slightly (Hungarian) faster runtimes than ArcGP+GPArc. Interestingly, both sets of constraints in isolation perform worse, in particular the OneGP and NoGP model.

7 Conclusion

We have presented a novel method for parsing in second order grandparent models, and a general blueprint for more efficient and optimal structured prediction. Our method lazily instantiates candidate parts based on their reduced cost, and on constraint

	Italian		Hungarian		Danish	
	BF	PPC	BF	PPC	BF	PPC
Sent./sec. relative to BF	100%	**760%**	100%	**380%**	100%	**390%**
GPs Scored relative to BF	100%	**6%**	100%	**12%**	100%	**13%**
GPs Added relative to BF	100%	**5%**	100%	**7%**	100%	**7%**
Objective rel. to BF	100%	100%	100%	100%	100%	100%
% of Integer Solutions	98%	98%	97%	97%	97%	97%
Unlabeled Acc.	88%	88%	81%	81%	88%	88%

Table 1: Parse, Price and Cut (PPC) vs Brute Force (BF). Speed is the number of sentences per second, relative to the speed of BF. Objective, GPs scored and added are also relative to BF.

Constraints	GPArc+ ArcGP	OneGP+ NoGP	All
Sent./sec.	100%	1000%	1200%
% Integer	77%	9%	98%
Unlabeled Acc.	87%	85%	88%

(a) Italian

Constraints	GPArc+ ArcGP	OneGP+ NoGP	All
Sent./sec.	100%	162%	105%
% Integer	71%	3%	97%
Unlabeled Acc.	80%	77%	81%

(b) Hungarian

Table 2: Different outer bounds on the grandparent polytope, for nonprojective parsing of Italian and Danish.

violations. This allows us to discard a large fraction of parts during both scoring and optimization, leading to nearly 800% speed-ups without loss of accuracy and certificates. We also present a tighter bound on the grandparent polytope that is useful in its own right.

Delayed column and row generation is very useful when solving large LPs with off-the-shelf solvers. Given the multitude of work in NLP that uses LPs and ILPs in this way (Roth and Yih, 2004; Clarke and Lapata, 2007), we hope that our approach will prove itself useful for other applications. We stress that this approach can also be used when working with dynamic programs, as pointed out in section 4.5, and therefore also in the context of dual decomposition. This suggests even wider applicability, and usefulness in various structured prediction problems.

The underlying paradigm could also be useful for more approximate methods. In this paradigm, algorithms maintain an estimate of the cost of certain resources (duals), and use these estimates to guide search and the propose new structures. For example, a local-search based dependency parser could estimate how contested certain tokens, or edges, are, and then use these estimates to choose better next proposals. The notion of reduced cost can give guidance on what such estimates should look like.

Acknowledgements

This work was supported in part by the Center for Intelligent Information Retrieval and the University of Massachusetts and in part by UPenn NSF medium IIS-0803847. We gratefully acknowledge 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 authors and do not necessarily reflect the view of DARPA, AFRL, or the US government.

References

Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1 edition, February.

David Belanger, Alexandre Passos, Sebastian Riedel, and Andrew McCallum. 2012. A column generation approach to connecting regularization and map inference. In *Inferning: Interactions between Inference and Learning, ICML 2012 Workshop*.

Dimitri P. Bertsekas. 1999. *Nonlinear Programming*. Athena Scientific, 2nd edition, September.

Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL' 06)*, CoNLL-X '06, pages 149–164, Stroudsburg, PA, USA. Association for Computational Linguistics.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL '05)*, pages 173–180.

James Clarke and Mirella Lapata. 2007. Modelling compression with discourse constraints. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '07)*, pages 1–11.

A. Culotta, M. Wick, R. Hall, and A. McCallum. 2007. First-order probabilistic models for coreference resolution. In *Joint Human Language Technology Conference/Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL '07)*, pages 81–88.

Peter Dienes, Alexander Koller, and Marco Kuhlmann. 2003. Statistical a-star dependency parsing. In *Proceedings of the workshop on Prospects and Advances of the Syntax/Semantics Interface, Nancy, 2003, pp.85-89*.

P.C. Gilmore and R.E. Gomory. 1961. A linear programming approach to the cutting-stock problem. *Operations research*, pages 849–859.

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact viterbi parse selection. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL '03)*, pages 119–126.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL '11)*.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with nonprojective head automata. In *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '10)*.

Sandra Kübler, Ryan T. McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Marco Lübbecke and Jacques Desrosiers. 2004. Selected topics in column generation. *Operations Research*, 53:1007–1023.

R. Kipp Martin, Ronald L. Rardin, and Brian A. Campbell. 1990. Polyhedral characterization of discrete dynamic programming. *Oper. Res.*, 38(1):127–138, February.

André F. T. Martins, Noah A. Smith, and Eric P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL '09)*, pages 342–350, Morristown, NJ, USA. Association for Computational Linguistics.

André F. T. Martins, Noah A. Smith, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. 2011. Dual decomposition with many overlapping components. In *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '11)*, EMNLP '11, pages 238–249, Stroudsburg, PA, USA. Association for Computational Linguistics.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the ACL (EACL '06)*, pages 81–88.

R. McDonald, K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL '05)*, pages 91–98.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *HLT-EMNLP, 2005*.

Mathias Niepert. 2010. A delayed column generation strategy for exact k-bounded map inference in markov logic networks. In *Proceedings of the 26th Annual Conference on Uncertainty in AI (UAI '10)*, pages 384–391, Corvallis, Oregon. AUAI Press.

J. Nivre, J. Hall, S. Kubler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The conll 2007 shared task on dependency parsing. In *Conference on Empirical Methods in Natural Language Processing and Natural Language Learning*, pages 915—932.

Hoifung Poon and Pedro Domingos. 2007. Joint inference in information extraction. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI '07)*, pages 913–918.

Sebastian Riedel and James Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '06)*, pages 129–137.

Sebastian Riedel and David A. Smith. 2010. Relaxed marginal inference and its application to dependency

parsing. In *Joint Human Language Technology Conference/Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL '10)*, pages 760–768, Los Angeles, California, June. Association for Computational Linguistics.

Sebastian Riedel. 2008. Improving the accuracy and efficiency of MAP inference for markov logic. In *Proceedings of the 24th Annual Conference on Uncertainty in AI (UAI '08)*, pages 468–475.

D. Roth and W. Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL' 04)*, pages 1–8.

Alexander Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Joint Human Language Technology Conference/Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL '12)*.

Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '10)*.

David A. Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 145–156, Honolulu, October.

D. Sontag and T. Jaakkola. 2007. New outer bounds on the marginal polytope. In *Advances in Neural Information Processing Systems (NIPS '07)*, pages 1393–1400.

David Sontag, T. Meltzer, A. Globerson, T. Jaakkola, and Y. Weiss. 2008. Tightening LP relaxations for MAP using message passing. In *Proceedings of the 24th Annual Conference on Uncertainty in AI (UAI '08)*.

Roy W. Tromble and Jason Eisner. 2006. A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *Joint Human Language Technology Conference/Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL '06)*, pages 423–430.