Dependency Parsing with Bounded Block Degree and Well-nestedness via Lagrangian Relaxation and Branch-and-Bound

Caio Corro **Joseph Le Roux** **Mathieu Lacroix**
Antoine Rozenknop **Roberto Wolfler Calvo**
Laboratoire d'Informatique de Paris Nord,
Université Paris 13 – SPC, CNRS UMR 7030,
F-93430, Villetaneuse, France
{corro,leroux,lacroix,rozenknop,wolfler}@lipn.fr

Abstract

We present a novel dependency parsing method which enforces two structural properties on dependency trees: bounded block degree and well-nestedness. These properties are useful to better represent the set of admissible dependency structures in treebanks and connect dependency parsing to context-sensitive grammatical formalisms. We cast this problem as an Integer Linear Program that we solve with Lagrangian Relaxation from which we derive a heuristic and an exact method based on a Branch-and-Bound search. Experimentally, we see that these methods are efficient and competitive compared to a baseline unconstrained parser, while enforcing structural properties in all cases.

1 Introduction

We address the problem of enforcing two structural properties on dependency trees, namely bounded block degree and well-nestedness, without sacrificing algorithmic efficiency. Intuitively, bounded block degree constraints force each subtree to have a yield decomposable into a limited number of blocks of contiguous words, while well-nestedness asserts that every two distinct subtrees must not interleave: either the yield of one subtree is entirely inside some gap of the other or they are completely separated. These two types of constraints generalize the notion of projectivity: projective trees actually have a block degree bounded to one and are well-nested.

Our first motivation is the fact that most dependency trees in NLP treebanks are well-nested and have a low block degree which depends on the language and the linguistic representation, as shown in (Pitler et al., 2012). Unfortunately, although polynomial algorithms exist for this class of trees (Gómez-Rodríguez et al., 2009), they are not efficient enough to be of practical use in applications requiring syntactic structures. In addition, if either property is dropped, but not the other, then the underlying decision problem becomes harder. That is why practical parsing algorithms are either completely unconstrained (McDonald et al., 2005) or enforce strict projectivity (Koo and Collins, 2010). This work is, to the best of our knowledge, the first attempt to build a discriminative dependency parser that enforces well-nestedness and/or bounded block degree and to use it on treebank data.

We base our method on the following observation: a non-projective dependency parser, thus not requiring neither well-nestedness nor bounded block degree, returns dependency trees satisfying these constraints in the vast majority of sentences. This would tend to indicate that the heavy machinery involved to parse with these constraints is only needed in very few cases.

We consider arc-factored dependency parsing with well-nestedness and bounded block degree constraints. We formulate this problem as an Integer Linear Program (ILP) and apply Lagrangian Relaxation where the dualized constraints are those associated with bounded block degree and well-nestedness. The Lagrangian dual objective then reduces to a maximum spanning arborescence and can be solved very efficiently. This provides an efficient heuristic for our problem. An exact method can be derived by embedding this Lagrangian Relaxation in a Branch-and-Bound procedure to solve the problem with an optimality certificate. Despite the exponential worst-time complexity of the Branch-and-Bound procedure, it is tractable in practice. Our formulation can enforce both types of constraints or only one of them without changing the resolution method.

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 355–366,
Berlin, Germany, August 7-12, 2016.

As stated in (Bodirsky et al., 2009), well-nested dependency trees with 2-bounded block degree are structurally equivalent to derivations in Lexicalized Tree Adjoining Grammars (LTAGs) (Joshi and Schabes, 1997).[1][2] While LTAGs can be parsed in polynomial time, developing an efficient parser for these grammars remains an open problem (Eisner and Satta, 2000) and we believe that this work could be a useful step in that direction.

Related work is reviewed in Section 2. We define arc-factored dependency parsing with block degree and well-nestedness constraints in Section 3. We derive an ILP formulation of this problem in Section 4 and then present our method based on Lagrangian Relaxation in Section 5 and Branch-and-Bound in Section 6. Section 7 contains experimental results on several languages.

2 Related Work

A dynamic programming algorithm has been proposed for parsing well-nested and k-bounded block degree dependency trees in (Gómez-Rodríguez et al., 2009; Gómez-Rodríguez et al., 2011). Unfortunately, it has a prohibitive $O(n^{3+2k})$ time complexity, equivalent to Lexicalized TAG parsing when $k = 2$. Variants of this algorithm have also been proposed for further restricted classes of dependency trees: 1-inherit ($O(n^6)$) (Pitler et al., 2012), head-split ($O(n^6)$) (Satta and Kuhlmann, 2014) and both 1-inherit and head-split ($O(n^5)$) (Satta and Kuhlmann, 2014). Although those restricted classes have good empirical coverage, they do not cover the exact search space of Lexicalized TAG derivation and their time complexity is still prohibitive. Spinal TAGs, described as a dependency parsing task in (Carreras et al., 2008), weaken even more the empirical coverage in practice, restricted to projective trees, but still remain hardly tractable with a complexity of $O(n^4)$. On the contrary, the present work does not restrict the search space.

Parsing mildly context-sensitive languages with dependencies has been explored in (Fernández-González and Martins, 2015) but the resulting parser cannot guarantee compliance with strict structural properties. On the other hand, the present method enforces the well-nestedness and bounded block degree of solutions.

The methods mentioned above all use the graph-based approach and rely on dynamic programming to achieve tractability. There is also a line of of work in transition-based parsing for various dependency classes. Systems have been proposed for projective dependency trees (Nivre, 2003), non-projective, or even unknown classes (Attardi, 2006). Pitler and McDonald (2015) propose a transition system for crossing interval trees, a more general class than well-nested trees with bounded block degree. In the case of spinal TAGs, we can mention the work of Ballesteros and Carreras (2015) and Shen and Joshi (2007). Transition-based algorithms offer low space and time complexity, typically linear in the length of sentences usually by relying on local predictors and beam strategies and thus do not provide any optimality guarantee on the produced structures. The present work follows the graph-based approach, but replaces dynamic programming with a greedy algorithm and Lagrangian Relaxation.

The use of Lagrangian Relaxation to elaborate sophisticated parsing models based on plain maximum spanning arborescence solutions originated in (Koo et al., 2010) where this method was used to parse with higher-order features. This technique has been explored to parse CCG dependencies in (Du et al., 2015) without a precise definition of the class of trees. We can also draw connections between our problem reduction procedure and the use of Lagrangian Relaxation to speed up dynamic programming and beam search with exact pruning in (Rush et al., 2013).

In this work we rely on Non-Delayed Relax-and-Cut for lazy constraint generation (Lucena, 2006). This can be linked to (Riedel, 2009) which uses a cutting plane algorithm to solve MAP inference in Markov Logic and (Riedel et al., 2012) which uses column and row generation for higher-order dependency parsing.

In NLP, the Branch-and-Bound framework (Land and Doig, 1960) has previously been used for dependency parsing with high order features in (Qian and Liu, 2013), and Das et al. (2012) combined Branch-and-Bound to Lagrangian Relaxation in order to retrieve integral solutions for shallow semantic parsing.

[1] It is possible to express a wider class of dependencies with LTAG if we allow dependencies direction to be different from the derivation tree (Kallmeyer and Kuhlmann, 2012).

[2] In order to be fully compatible with LTAGs, we must ensure that the root has only one child. For algorithmic issues see (Fischetti and Toth, 1992) or (Gabow and Tarjan, 1984).

3 Dependency Parsing

We model the dependency parsing problem using a graph-based approach. Given a sentence $s = \langle s_0, \dots, s_n \rangle$ where s_0 is a dummy root symbol, we consider the directed graph $D = (V, A)$ with $V = \{0, \dots n\}$ and $A \subseteq V \times V$. Vertex $i \in V$ corresponds to word s_i and arc $(i, j) \in A$ models a dependency from word s_i to word s_j. In the rest of the paper, we denote $V \setminus \{0\}$ by V^+.

An *arborescence* is a set of arcs T inducing a connected graph with no circuit such that every vertex has at most one entering arc. The set of vertices incident with any arc of T is denoted by $V[T]$. If $V[T] = V$, then T is a *spanning arborescence*. Among the vertices of $V[T]$, the one with no entering arc is called the *root* of T. A vertex t is *reachable* from a vertex s with respect to T if there exists a path from s to t using only arcs of T. The *yield* of a vertex $v \in V$ corresponds to the set of vertices reachable from v with respect to T.

It is well-known that there is a bijection between dependency trees for s and spanning arborescences with root 0 (McDonald et al., 2005). In what follows, the term dependency tree will refer to both the dependency tree of s and its associated spanning arborescence of D with root 0.

In the dependency parsing problem, one has to find a dependency tree with maximal score. Several scores can be associated with each dependency tree and different conditions can restrict the set of valid dependency trees.

In this paper, we consider an *arc-factored* model: each arc $(i, j) \in A$ is assigned a score w_{ij}; the score of a dependency tree is defined as the sum of the scores of the arcs it contains. This model can be computed in $O(n^2)$ with Chu–Liu-Edmonds' algorithm for Maximum Spanning Arborescence (MSA) (McDonald et al., 2005). Unfortunately, this algorithm forbids any modification of the score function, for example adding score contribution for combinations of arcs (i.e. grand-parent or sibling models). Moreover, adding score contribution for combinations of couple of arcs makes the problem NP-hard (McDonald and Pereira, 2006), although several methods have been developed to tackle this problem, for instance dual decomposition (Koo et al., 2010).

Likewise, restrictions on the tree structure such as the well-nestedness and bounded block degree conditions are not permitted in the MSA algorithm. We first give a precise definition of these structural properties, equivalent to (Bodirsky et al., 2009), before we present a method to take them into account. From now on, we suppose that instances are equipped with a positive integer k and we call *valid dependency trees* those satisfying the k-bounded block degree and well-nestedness conditions. A graph-theoretic definition of these two conditions can be given as follows.

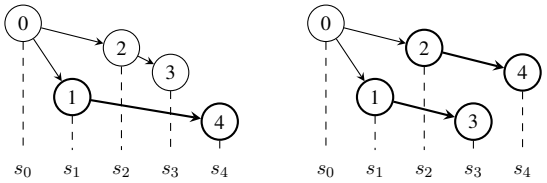

Figure 1: (Left) A 2-BBD arborescence: the block degree of vertex 1 is 2 (its yield is $\{1, 4\}$) whereas the block degree of all other vertices is 1. (Right) A not well-nested arborescence: the yields of vertices 1 and 2 interleave.

Block degree The *block degree* of a vertex set $W \subseteq V$ is the number of vertices of W without a predecessor[3] inside W. Given an arborescence T, the *block degree* of a vertex is the block degree of its yield and the *block degree* of T is the maximum block degree of its incident vertices. An arborescence satisfies the k-bounded block degree condition if its block degree is less than or equal to k. We then say it is *k-BBD* for short. Figure 1 (left) gives an example of a 2-BBD arborescence.

Well-nestedness Two disjoint subsets $I_1, I_2 \subset V^+$ *interleave* if there exist $i, j \in I_1$ and $k, l \in I_2$ such that $i < k < j < l$. An arborescence is *well-nested* if it is not incident to two vertices whose yields interleave. Figure 1 (right) shows an arborescence which is not well-nested.

4 ILP Formulation

In this section we formulate the dependency parsing problem described in Section 3 as an ILP. We start with some notation and two theorems characterizing k-BBD and well-nested dependency trees.

Given a subset $W \subseteq V$, the set of arcs entering W is denoted by $\delta^{\text{in}}(W)$ and the set of arcs leaving W is denoted by $\delta^{\text{out}}(W)$. The set $\delta(W) = \delta^{\text{in}}(W) \cup \delta^{\text{out}}(W)$ is called the *cut* of W. Given a positive integer l, let $\mathcal{W}^{\geq l}$ be the family of vertex subsets of V^+ with block degree greater than or equal to l. For instance, given any sentence with more than 6 words, $\{1, 3, 5, 6\} \in \mathcal{W}^{\geq 3}$,

[3]The predecessor of a vertex $v \in V$ is $v - 1$.

while $\{1,2,5,6\} \notin \mathcal{W}^{\geq 3}$. We also denote by $\mathcal{I}$ the family of couples of disjoint interleaving vertex subsets of V^+. For instance, $(\{1,4\},\{2,3,5\})$ belongs to $\mathcal{I}$. Finally, given a vector $x \in \mathbb{R}^A$ and a subset $B \subseteq A$, $x(B)$ corresponds to $\sum_{a\in B} x_a$.

Theorem 1. *A dependency tree T is not k-BBD iff there exists a vertex subset $W \in \mathcal{W}^{\geq k+1}$ whose cut $\delta(W)$ contains a unique arc of T.*

Proof. By definition of block degree, a dependency tree is not k-BBD iff it is incident with a vertex whose yield W belongs to $\mathcal{W}^{\geq k+1}$. It is equivalent to say that T contains a subarborescence T' such that $V[T']$ equals W. This holds iff W has one entering arc (since $0 \notin W$) and no leaving arc belonging to T. □

Theorem 2. *A dependency tree T is not well-nested iff there exists $(I_1, I_2) \in \mathcal{I}$ such that $\delta(I_1) \cap T$ and $\delta(I_2) \cap T$ are singletons.*

Proof. $\delta(I_1)$ and $\delta(I_2)$ both intersect T only once iff T contains two arborescences T_1 and T_2 such that $V[T_1] = I_1$ and $V[T_2] = I_2$. This means that T is incident with two vertices whose yields are I_1 and I_2, respectively. Result follows from the definition of $\mathcal{I}$ and well-nested arborescences. □

The dependency parsing problem can be formulated as follows. A dependency tree will be represented by its incidence vector. Hence, we use variables $z \in \mathbb{R}^A$ such that $z_a = 1$ if arc a belongs to the dependency tree and 0 otherwise.

$$\max_z \sum_{a\in A} w_a z_a \tag{1}$$

$$z(\delta^{\text{in}}(v)) = 1 \qquad \forall v \in V^+ \tag{2}$$

$$z(\delta^{\text{in}}(W)) \geq 1 \qquad \forall W \subseteq V^+ \tag{3}$$

$$z(\delta(W)) \geq 2 \qquad \forall\, W \in \mathcal{W}^{\geq k+1} \tag{4}$$

$$z(\delta(I_1)) + z(\delta(I_2)) \geq 3 \quad \forall (I_1, I_2) \in \mathcal{I} \tag{5}$$

$$z \in \{0,1\}^A \tag{6}$$

The objective function (1) maximizes the score of the dependency tree. Inequalities (2) ensure that all vertices but the root have one entering arc. Inequalities (3) force the set of arcs associated with z to induce a connected graph. Inequalities (2) and (3), together with $z \geq 0$, give a linear description of the convex hull of the incidence vectors of the spanning arborescences with root 0 — see *e.g.,* (Schrijver, 2003). Inequalities (4) ensure that the dependency tree is k-BBD and inequalities (5) impose well-nestedness. The validity of (4) and (5) follows from Theorems 1 and 2, respectively.

Remark that (3) could be replaced by a polynomial number of additional flow variables and constraints, see (Martins et al., 2009).[4]

5 Lagrangian Relaxation

Solving this ILP using an off-the-shelf solver is ineffective due to the huge number of constraints. We tackle this problem with Lagrangian Relaxation, which has become popular in the NLP community, see for instance (Rush and Collins, 2012). Note that contrary to most previous work on Lagrangian Relaxation for NLP, we do not use it to derive a decomposition method.

We note that optimizing objective (1) subject to constraints (2), (3) and (6) amounts to finding a MSA and can be solved combinatorially (McDonald et al., 2005). Thus, since formulation (1)–(6) is based only on arc variables, by relaxing constraints (4) and (5), one obtains a Lagrangian dual objective which is nothing but a MSA problem with reparameterized arc scores. Our Lagrangian approach relies on a subgradient descent where a MSA problem is solved at each iteration. We give more details in the rest of the section.

5.1 Dual Problem

Let Z be the set of the incidence vectors of dependency trees. Keeping tree shape constraints (2), (3) and (6) while dualizing k-bounded block degree constraints (4) and well-nestedness constraints (5), we build the following Lagrangian (Lemaréchal, 2001):

$$\begin{aligned} L(z,u) = &\sum_{a\in A} w_a z_a \\ &+ \sum_{W\in\mathcal{W}^{\geq k}} u_1^W \times (z(\delta(W)) - 2) \\ &+ \sum_{(I_1,I_2)\in\mathcal{I}} u_2^{I_1,I_2} \times (z(\delta(I_1)) + z(\delta(I_2)) - 3) \end{aligned} \tag{7}$$

[4] Based on this remark, we also developed a formulation of this problem with a polynomial number of variables and constraints. However it requires adding many more variables than (Martins et al., 2009). This leads to a formulation which is not tractable, see Section 7.2. Moreover, it cannot be tackled by our Lagrangian Relaxation approach.

with $z \in Z$ and $u = (u_1, u_2) \geq 0$ is a vector of Lagrangian multipliers. We refactor to:

$$L(z, u) = \sum_{a \in A} \theta_a z_a + c \tag{8}$$

where θ are modified scores and c a constant term.

The dual objective is $L^*(u) = \max_z L(z, u)$ with $z \in Z$. Note that computing $L^*(u)$ amounts to solving the MSA problem with modified scores θ and can be efficiently computed. The dual problem is $\min_{u \geq 0} L^*(u)$. L^* is a non-differentiable convex piece-wise linear function and one can find its minimum via subgradient descent. For any vector u, we use the following subgradient. Denote $Mz \leq b$ the set of constraints given by (4) and (5) and $z^* = \arg\max_z L(z, u)$. Let $g = b - Mz^*$ be a subgradient at u, see (Lemaréchal, 2001) for more details. From this subgradient, we compute the descent direction following (Camerini et al., 1975), which aggregates information during the iteration of the subgradient descent algorithm. Unfortunately, optimizing the dual is expensive with so many relaxed constraints. We handle this problem in the next subsection.

5.2 Efficient Optimization with Many Constraints

The *Non Delayed Relax-and-Cut* (NDRC) method (Lucena, 2005) tackles the problem of optimizing a Lagrangian dual problem with exponentially many relaxed constraints. In standard subgradient descent, at each iteration p of the descent, the Lagrangian update can be formulated as:

$$u^{p+1} = (u^p - s^p \times g^p)_+ \tag{9}$$

where $s^p > 0$ is the stepsize[5] and $()_+$ denotes the projection onto $\mathbb{R}^+$, which replaces each negative component by 0. If all Lagrangian multipliers are initialized to 0, the compononent corresponding to a constraint will not be changed until this constraint is violated for the first time. Indeed, by definition of g, we have $[g^p]_i \geq 0$ if constraint i is satisfied at iteration p: the projection on $\mathbb{R}^+$ ensure that $[u^{p+1}]_i$ stays at 0.[6] Thus we do not need to know constraints that have not been violated yet in order to correctly update the Lagrangian multipliers: this is the main intuition behind the NDRC method. However, s^p may depend on the full subgradient information. A common step size (Fisher, 1981) is:

$$s^p = \alpha^p \times \frac{L^*(u^p) - LB^p}{\|g^p\|^2} \tag{10}$$

with α^p a scalar and LB^p the best known lower bound. This is also the case with more recent approaches like AdaGrad (Duchi et al., 2011) and AdaDelta (Zeiler, 2012). As reported in (Beasley, 1993; Lucena, 2006), when dealing with many relaxed constraints, the $\|g^p\|^2$ term can result in each Lagrangian update being almost equal to 0. Therefore, a good practice is to modify the subgradient such that if $[g^p]_i > 0$ and $[u^p]_i = 0$, then we set $[g^p]_i = 0$: this has the same effect on the multipliers as the projection on $\mathbb{R}^+$ in (9), but it prevents the stepsize from becoming too small. Hence, instead of generating a full subgradient at each iteration, which is an expensive operation because we would need to consider all multipliers associated with constraints, we process only a subpart, namely the one associated with constraints that have been violated.

Following (Lucena, 2005), at each iteration p of the subgradient descent we define two sets: *Currently Violated Active Constraints* (CA^p) and *Previously Violated Active Constraints* (PA^p). CA^p and PA^p are not necessarily disjoint. The subgradient is computed only for constraints in $CA^p \cup PA^p$. At each iteration p, we update $PA^p = PA^{p-1} \cup CA^{p-1}$ and a *violation detection step*, similar to the separation step in a cutting plane algorithm, generates CA^p. Two strategies are possible for the detection: (1) adding to CA^p all the constraints violated by the current dual solution; (2) adding only a subset of them. The latter is justified by the fact that many constraints may overlap thus leading to exageration of modified scores on some arcs. We found that strategy (2) gives better convergence results.

Detection for violated block degree constraints (4) can be done with the algorithm described in (Möhl, 2006) in $O(n^2)$. If no violated block degree constraint is found, we search for violated well-nestedness constraints (5) using the $O(n^2)$ algorithm described in (Havelka, 2007).

5.3 Lagrangian Heuristic

We derive a heuristic from the Lagragian Relaxation. First, a dependency tree is computed with

[5] As stated above, instead of the subgradient we follow an improved descent direction which aggregates information of previous iterations. However, this does not change the proposal of this subsection.

[6] $[x]_i$ denotes the ith component of vector x.

the MSA algorithm. If it is valid, it then corresponds to the optimal solution. Otherwise, we proceed as follows. The computation of the step size in (10) in the subgradient descent needs a lower bound which can be given by the score of any valid dependency tree. In our experiments, we compute the best projective spanning arborescence (Eisner, 2000). Each iteration of the subgradient descent computes a spanning arborescence. Since violating (4) and (5) is penalized in the objective function, it tends to produce valid dependency trees. The heuristic returns the highest scoring one.

6 Branch and Bound

Solving the Lagrangian dual problem may not always give an optimal solution to the original problem because of the potential duality gap. Still, we always obtain an upper bound on the optimal solution and if a dual solution satisfies constraints (4) and (5), its score with the original weights provides a lower bound.[7]

Moreover, the subgradient descent algorithm theoretically converges but we have no guarantee that this will happen in a realistic number of iterations. Therefore, in order to retrieve an optimal solution in all cases, we embed the Lagrangian Relaxation of the problem within a Branch-and-Bound procedure (Land and Doig, 1960).

The search space is recursively split according to an arc variable, creating two subspaces, one where it is fixed to 1 and the other to 0 (branching step). The procedure returns a candidate solution when all arc variables are fixed and constraints are satisfied, and the optimal solution is the highest-scoring candidate solution.

For each subspace, we estimate an upper bound using the Lagrangian Relaxation (bounding step). The recursive exploration of a subspace stops (pruning step) if (1) we can prove that all candidate solutions it contains have a score lower than the best found so far, or (2) we detect an unsastifiable constraint.

The branching strategy is built upon Lagrangian multipliers: we branch on the variable z_a with highest value $\theta_a - w_a$. Intuitively, if the branching step sets $z_a = 1$, it indicates that we add a *hard* constraint on an arc which has been strongly promoted by Lagrangian Relaxation. This strategy, compared to other variants, gave the best parsing time on development data.

6.1 Problem Reduction

The efficiency of the Branch-and-Bound procedure crucially depends on the number of free variables. To prune the search space, we rely on problem reduction (Beasley, 1993), once again based on duality and Lagrangian Relaxation, which provides certificates on optimal variable settings.

We fix a variable to 1 (resp. 0), and compute an upper bound on the optimal score with this new constraint. If it is lower than the score of the best solution found so far without this constraint, we can guarantee that this variable cannot (resp. must) be in the optimal solution and safely set it to 0 (resp. 1).

Problem reduction is performed at each node of the Branch-and-Bound tree after computing the upper bound with subgradient descent.

6.2 Fixing Variables to 1

Since a node in V^+ must have exactly one parent, fixing $z_{ij} = 1$ for an arc $a = (i, j)$ greatly reduces the problem size, as it will also fix $z_{hj} = 0, \forall h \neq i$. Among all arc variables that can be set to 1, promising candidates are the arcs in a solution of the unconstrained MSA and the arcs obtained in a solution after the subgradient descent.

There are exactly n such arcs in each set of candidates, so we test fixation for less than $2n$ variables. In this case, we are ready to pay the price of a quadratic computation for each of these arcs.

Hence, for each candidate arc we obtain an upper bound by seeking the (unconstrained) MSA on the graph where this arc is removed. If this upper bound is lower than the score of the best solution found so far, we can safely say that this arc is in the optimal solution.

6.3 Fixing Variables to 0

We could apply the same strategy for fixing variables to 0. However, this reduction is less rewarding and there are many more variables set to 0 than 1 in a MSA solution. Instead, we solve an easier problem, at the expense of a looser upper bound.

For each arc a which is not in the MSA, we compute a maximum directed graph that contains this arc and where all nodes but the root have one parent. Remark that if this graph is connected then it corresponds to a dependency tree. Therefore, the score of this directed graph provides an upper bound on a solution containing arc a. If this upper

[7] Because relaxed constraints are inequalities, constraint satisfaction does not guarantee optimality (Beasley, 1993).

bound is lower than the score of the best solution found so far, we can fix the variable z_a to 0.

Note that this whole fixing procedure can be done in $O(n^2)$.

7 Experiments

We ran a series of experiments to test our method in the case of unlabelled dependency parsing. Our prototype has been developped in Python with some parts in Cython and C++. We use the MSA implementation available in the LEMON library.[8]

7.1 Datasets

We ran experiments on 5 different corpora:

English: Dependencies were extracted from the WSJ part of the Penn Treebank (PTB) with additional NP bracketings (Vadas and Curran, 2007) with the *LTH* converter[9] (default options). Sections 02-21 are used for training, 22 for development and 23 for testing. POS tags were predicted by the Stanford tagger[10] trained with 10-jackkniffing.[11]

German: We used dependencies from the SPMRL dataset (Seddah et al., 2014), with predicted POS tags and the official split. We removed sentences of length greater than 100 in the test set.

Dutch, Spanish and Portuguese: We used the Universal Dependency Treebank 1.2 (Van der Beek et al., 2002; McDonald et al., 2013; Afonso et al., 2002) with gold POS tags and the official split. We removed sentences of length greater than 100 in the test sets.

Those datasets contain different structure distributions as shown in Table 1. Fortunately, our method allows us to easily change the bounded degree constraint or toggle the well-nestedness one. For each language, we decided to use the most constrained combination of bounded block degree constraints and well-nestedness which covers over 99% of the data. Therefore, we chose to enforce 2-BBD and well-nestedness for English and Spanish, 3-BBD and well-nestedness for Dutch and Portuguese and 3-BBD only for German.

7.2 Decoding

In order to compare our methods with previous approaches, we tested five decoding strategies. (`MSA`) computes the best unconstrained dependency tree. (`Eisner`) computes the best projective tree. (`LR`) and (`B&B`) are the heuristic and the exact method presented in Sections 5.3 and 6 respectively.[12] Finally (`MSA/Eisner`) consists in running the MSA algorithm and, if the solution is invalid, returning the (`Eisner`) solution instead.

Our attempt to run the dynamic programming algorithm of (Gómez-Rodríguez et al., 2009) was unsuccessful. Even with heavy pruning we were not able to run it on sentences above 20 words. We also tried to use CPLEX on a compact ILP formulation based on multi-commodity flows (see footnote 4). Parsing time was also prohibitive: a total of 3473 seconds on English data without the well-nestedness constraint, 7298 for German.

We discuss the efficiency of our methods on data for English and German. Other languages give similar results. Optimality rate after the subgradient descent are reported in Figure 2. We see that Lagrangian Relaxation often returns optimal solutions but fails to give a certificate of their optimality. Table 2 shows parsing times. We see that (`LR`) and (`B&B`), while slower than (`MSA`), are fast in the majority of cases, below the third quartile. Inevitably, there are some rare cases where a large portion of the search space is explored, and thus their parsing time is high. Let us remark that these algorithms are run only when (`MSA`) returns an invalid structure, and so total time is very acceptable compared to the baseline.

Finally, we stress the importance of problem reduction as a pre-processing step in `B&B`: after subgradient descent is performed, it removes an average of 83.97% (resp. 76.59%) of arc variables in the English test set (resp. German test set).

7.3 Training

Feature weights are trained using the averaged structured perceptron (Collins, 2002) with 10 iterations where the best iteration is selected on the development set. We used the same feature set as in TurboParser (Martins et al., 2010), including features for lemma. For German, we additionally use morpho-syntactic features.

The decoding algorithm used at training time is the MSA. We experimented with Branch-and-Bound and Lagrangian Relaxation decoding dur-

[8] https://lemon.cs.elte.hu/trac/lemon

[9] http://nlp.cs.lth.se/software/treebank_converter/

[10] http://nlp.stanford.edu/software/tagger.shtml

[11] Prediction precision: 97.40%

[12] In both methods, the subgradient descent is stopped after a fixed maximum number of iterations. We chose 100 for English and 200 for other languages after tuning on the development set.

	English		German		Dutch		Spanish		Portuguese	
	WN	IL	WN	IL	WN	IL	WN	IL	WN	IL
Block degree 1	92.26	-	67.60	-	69.13	-	93.95	-	81.56	0.05
Block degree 2	7.58	0.12	27.12	0.79	28.50	0.08	5.99	0.04	13.92	0.02
Block degree 3	0.12	0.01	3.86	0.30	2.24	0.01	0.02	-	3.76	-
Block degree 4	-	-	0.19	<0.01	0.04	-	-	-	0.54	-
Block degree > 4	-	-	0.11	<0.01	-	-	-	-	0.14	-

Table 1: Distribution of dependency tree characteristics in datasets.

	English (96 sentences)			German (59 sentences)		
	MSA	LR	B&B	MSA	LR	B&B
Mean	0.02	0.26	0.53	0.04	0.51	0.71
Std.	0.01	0.20	0.86	0.02	0.41	1.39
Med.	0.02	0.21	0.27	0.03	0.47	0.47
3rd	0.03	0.34	0.53	0.05	0.71	0.80
Total	1.81	25.09	50.52	2.18	30.19	42.20

Table 2: Timings for strategies (see Section 7.2) on test for solutions which do not satisfy constraints after running MSA. We give (in seconds) average time, standard deviation, median time, time to parse up to the 3rd quartile and total time.

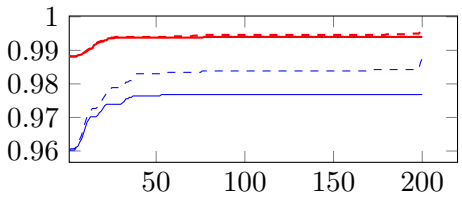

Figure 2: Optimality rate (y-axis) vs number of subgradient iterations (x-axis) for English (thin blue) and German (thick red). Solid line is the optimal rate with certificate, dashed is without.

ing training. It did not significantly improve accuracy and made training and decoding slower.

7.4 Parsing Results

Table 3 shows attachment score (UAS), percentage of valid dependency trees and relative time to (MSA) for different systems for our five decoding strategies. We can see (B&B) is on a par with (LR) on some corpora and more accurate on the others. The former takes more time, and the improvement is correlated with time difference for all corpora but the PTB. Dividing the five corpora in three cases, we can see that:

1. For English and Spanish, where projective dependency trees represent more than 90% of the data, (Eisner) outperforms (MSA). Our methods lie between the two. Here it is better to search for projective trees and (LR) and (B&B) are not interesting in terms of UAS. This is confirmed by the results of (MSA/Eisner).

2. For German and Dutch, where projective dependency trees represent less than 70% of the data, (MSA) outperforms (Eisner). For German, where well-nestedness is not required, our methods are as accurate as (MSA)[13], while for Dutch our methods seem to be useful, as (B&B) outperforms all systems. Moreover, our two methods guarantee validity.

3. For Portuguese, where projective dependency trees represent around 80% of the data, (MSA) is as accurate as (Eisner). In this case we see that, while our heuristic is below, the exact method is more accurate. This seems to be an edge case where neither unconstrained nor projective dependency trees seem to adequately capture the solution space. We also see that it is harder for our methods to give solutions (longer computation times, which tend to indicate that LR cannot guarantee optimality). Our methods are best fitted for this case.

In order to see how much well-nested and bounded block-degree structures are missed by a state-of-the-art parser, we compare our results with TurboParser.[14] We run the parser with three different feature sets: arc-factored, standard (second-order features), and full (third-order features). The results are shown in Table 4. Our model, by enforcing strict compliance to structural rules (100% valid dependency trees), is closer to the empirical distribution than TurboParser in arc-factored mode on all languages but German. Higher-order scoring functions manage to get more similar to the treebank data than our strict thresholds for all languages but Portuguese, at the expense of a significative computational burden.

[13] For German, we notice a small regression which we attribute to the representation of enumerations in the corpus: for enumerations of k elements, k-bounded block-degree subtrees must be generated.

[14] We used 2.1.0 and all defaults but the feature set.

		MSA	Eisner	LR	B&B	MSA/Eisner
English	UAS	89.45	**89.82**	89.54	89.53	89.60
	2-BBD/WN	96.02	–	–	–	–
	Relative Time	1	2.5	1.8	2.5	1.2
German	UAS	**87.79**	86.97	87.78	87.78	87.46
	3-BBD	98.81	–	–	–	–
	Relative Time	1	2.1	1.5	1.7	1.3
Dutch	UAS	77.30	76.62	76.96	**77.40**	76.79
	3-BBD/WN	94.82	–	–	–	–
	Relative Time	1	1.5	1.7	5	1.3
Spanish	UAS	83.37	**83.56**	83.37	83.44	83.48
	2-BBD/WN	92.62	–	–	–	–
	Relative Time	1	2.8	2.7	3	1.5
Portuguese	UAS	83.13	83.14	82.99	**83.21**	82.90
	3-BBD/WN	87.84	–	–	–	–
	Relative Time	1	2.7	5.7	19.7	1.7

Table 3: UAS, percentage of valid structure and decoding time for test data. Time is relative to MSA decoding. The percentage of valid structure is always 100% except for MSA decoding.

	English (99.84)			**German (99.27)**			**Dutch (99.87)**			**Spanish (99.94)**			**Portuguese (99.24)**		
Order	UAS	VDT	RT	UAS	VDT	RT	UAS	VDT	RT	UAS	VDT	RT	UAS	VDT	RT
1st	89.29	94.87	1	87.97	98.74	1	76.10	93.26	1	83.11	93.43	1	83.53	94.79	1
2nd	92.04	99.75	16	89.83	99.28	16	79.05	97.93	18	86.61	98.54	10	87.35	98.96	15
3rd	92.37	99.75	34	90.35	99.24	36	79.68	97.41	37	87.31	99.64	18	88.09	98.98	32

Table 4: UAS, percentage of valid dependency trees (VDT) and relative time (RT) obtained by Turboparser for different score functions on test sets. For each language we give the percentage of valid dependency structures in the data, according to the constraints postulated in Section 7.1.

We interpret this fact as an indication that adding higher order features into our system would make the relaxation method converge more often and faster.

8 Conclusion

We presented a novel characterization of dependency trees complying with two structural rules: bounded block degree and well-nestedness from which we derived two methods for arc-factored dependency parsing. The first one is a heuristic which relies on Lagrangian Relaxation and the Chu-Liu-Edmonds efficient maximum spanning arborescence algorithm. The second one is an exact Branch-and-Bound procedure where bounds are provided by Lagrangian Relaxation. We showed experimentally that these methods give results comparable with state-of-the-art arc-factored parsers, while enforcing constraints in all cases.

In this paper we focused on arc-factor models, but our method could be extended to higher order models, following the dual decomposition method presented in (Koo et al., 2010) in which the maximum-weight spanning arborescence component would be replaced by our constrained model.

Our method opens new perspectives for LTAG parsing, in particular using decomposition techniques where dependencies and templates are predicted separately. Moreover, while well-nested dependencies with 2-bounded block degree can represent LTAG derivations, toggling the well-nestedness or setting the block degree bound allows to express the whole range of derivations in lexicalized LCFRS, whether well-nested or with a bounded fan-out. Our algorithm can exactly represent these settings with a comparable complexity.

Acknowledgements

We thank the anonymous reviewers for their insightful comments which let us significantly improve the submitted paper. This work is supported by a public grant overseen by the French National Research Agency (ANR) as part of the Investissements d'Avenir program (ANR-10-LABX-0083).

References

[Afonso et al.2002] Susana Afonso, Eckhard Bick, Renato Haber, and Diana Santos. 2002. Floresta sintá (c) tica: A treebank for portuguese. In *LREC*.

[Attardi2006] Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 166–170. Association for Computational Linguistics.

[Ballesteros and Carreras2015] Miguel Ballesteros and Xavier Carreras. 2015. Transition-based spinal

parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 289–299, Beijing, China, July. Association for Computational Linguistics.

[Beasley1993] John E Beasley. 1993. Lagrangian relaxation. In *Modern heuristic techniques for combinatorial problems*, pages 243–303. John Wiley & Sons, Inc.

[Bertsekas1999] Dimitri P Bertsekas. 1999. *Nonlinear programming*. Athena scientific.

[Bodirsky et al.2009] Manuel Bodirsky, Marco Kuhlmann, and Mathias Möhl. 2009. Well-nested drawings as models of syntactic structure. In *Tenth Conference on Formal Grammar and Ninth Meeting on Mathematics of Language*, pages 195–203.

[Camerini et al.1975] Paolo M Camerini, Luigi Fratta, and Francesco Maffioli. 1975. On improving relaxation methods by modified gradient techniques. In *Nondifferentiable optimization*, pages 26–34. Springer.

[Carreras et al.2008] Xavier Carreras, Michael Collins, and Terry Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 9–16. Association for Computational Linguistics.

[Collins2002] Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics.

[Das et al.2012] Dipanjan Das, André FT Martins, and Noah A Smith. 2012. An exact dual decomposition algorithm for shallow semantic parsing with constraints. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 209–217. Association for Computational Linguistics.

[Du et al.2015] Yantao Du, Weiwei Sun, and Xiaojun Wan. 2015. A data-driven, factorization parser for CCG dependency structures. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference of Natural Language Processing of the Asian Federation of Natural Language Processing*, pages 1545–1555.

[Duchi et al.2011] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

[Eisner and Satta2000] Jason Eisner and Giorgio Satta. 2000. A faster parsing algorithm for lexicalized tree-adjoining grammars. In *Proceedings of the 5th Workshop on Tree-Adjoining Grammars and Related Formalisms (TAG+ 5)*, pages 14–19.

[Eisner2000] Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies*, pages 29–61. Springer.

[Fernández-González and Martins2015] Daniel Fernández-González and André F. T. Martins. 2015. Parsing as Reduction. In *Annual Meeting of the Association for Computational Linguistics (ACL'15)*, Beijing, China, July.

[Fischetti and Toth1992] Matteo Fischetti and Paolo Toth. 1992. An additive bounding procedure for the asymmetric travelling salesman problem. *Mathematical Programming*.

[Fisher1981] Marshall L Fisher. 1981. The lagrangian relaxation method for solving integer programming problems. *Management science*, 27.

[Gabow and Tarjan1984] Harold N. Gabow and Robert E. Tarjan. 1984. Efficient algorithms for a family of matroid intersection problems. *Journal of Algorithms*, 5(1):80–131.

[Gómez-Rodríguez et al.2009] Carlos Gómez-Rodríguez, David Weir, and John Carroll. 2009. Parsing mildly non-projective dependency structures. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 291–299. Association for Computational Linguistics.

[Gómez-Rodríguez et al.2011] Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2011. Dependency parsing schemata and mildly non-projective dependency parsing. *Computational linguistics*, 37(3):541–586.

[Havelka2007] Jiří Havelka. 2007. Relationship between non-projective edges, their level types, and well-nestedness. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 61–64. Association for Computational Linguistics.

[Joshi and Schabes1997] Aravind K Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In *Handbook of formal languages*, pages 69–123. Springer.

[Kallmeyer and Kuhlmann2012] Laura Kallmeyer and Marco Kuhlmann. 2012. A formal model for plausible dependencies in lexicalized tree adjoining grammar. In *Proceedings of TAG*, volume 11, pages 108–116.

[Koo and Collins2010] Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of ACL*.

[Koo et al.2010] Terry Koo, Alexander M Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298. Association for Computational Linguistics.

[Land and Doig1960] Ailsa H. Land and Alison G. Doig. 1960. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, 28(3):497–520.

[Lemaréchal2001] Claude Lemaréchal. 2001. Lagrangian relaxation. In *Computational combinatorial optimization*, pages 112–156. Springer.

[Lucena2005] Abilio Lucena. 2005. Non delayed relax-and-cut algorithms. *Annals of Operations Research*, 140(1):375–410.

[Lucena2006] Abilio Lucena. 2006. Lagrangian relax-and-cut algorithms. In *Handbook of Optimization in Telecommunications*, pages 129–145. Springer.

[Martins et al.2009] André FT 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: Volume 1-Volume 1*, pages 342–350. Association for Computational Linguistics.

[Martins et al.2010] André F. T. Martins, Noah A. Smith, Eric P. Xing, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP 2010, 9-11 October 2010, MIT Stata Center, Massachusetts, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 34–44.

[McDonald and Pereira2006] Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL*.

[McDonald et al.2005] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics.

[McDonald et al.2013] Ryan T McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith B Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. 2013. Universal dependency annotation for multilingual parsing. In *ACL (2)*, pages 92–97. Citeseer.

[Möhl2006] Mathias Möhl. 2006. *Drawings as models of syntactic structure: Theory and algorithms*. Ph.D. thesis, Saarland University.

[Nivre2003] Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*.

[Pitler and McDonald2015] Emily Pitler and Ryan McDonald. 2015. A linear-time transition system for crossing interval trees. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 662—-671.

[Pitler et al.2012] Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2012. Dynamic programming for higher order parsing of gap-minding trees. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 478–488. Association for Computational Linguistics.

[Qian and Liu2013] Xian Qian and Yang Liu. 2013. Branch and bound algorithm for dependency parsing with non-local features. *Transactions of the Association for Computational Linguistics*, 1:37–48.

[Riedel et al.2012] Sebastian Riedel, David Smith, and Andrew McCallum. 2012. Parse, price and cut: delayed column and row generation for graph based parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 732–743. Association for Computational Linguistics.

[Riedel2009] Sebastian Riedel. 2009. Cutting plane map inference for markov logic. In *SRL 2009*.

[Rush and Collins2012] Alexander M Rush and Michael Collins. 2012. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*.

[Rush et al.2013] Alexander M. Rush, Yin-Wen Chang, and Michael Collins. 2013. Optimal beam search for machine translation. In *Proceedings of EMNLP*.

[Satta and Kuhlmann2014] Giorgio Satta and Marco Kuhlmann. 2014. Efficient parsing for head-split dependency trees. *Transactions of the Association for Computational Linguistics*, 1:267–278.

[Schrijver2003] A. Schrijver. 2003. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer.

[Seddah et al.2014] Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the spmrl 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109.

[Shen and Joshi2007] Libin Shen and Aravind K Joshi. 2007. Bidirectional ltag dependency parsing. Technical report, Technical Report 07-02, IRCS, University of Pennsylvania.

[Vadas and Curran2007] David Vadas and James R. Curran. 2007. Adding noun phrase structure to the penn treebank. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-07)*, pages 240–247.

[Van der Beek et al.2002] Leonoor Van der Beek, Gosse Bouma, Rob Malouf, and Gertjan Van Noord. 2002. The alpino dependency treebank. *Language and Computers*, 45(1):8–22.

[Zeiler2012] Matthew D Zeiler. 2012. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.