A Transition-Based Parser for 2-Planar Dependency Structures

Carlos Gómez-Rodríguez
Departamento de Computación
Universidade da Coruña, Spain
carlos.gomez@udc.es

Joakim Nivre
Department of Linguistics and Philology
Uppsala University, Sweden
joakim.nivre@lingfil.uu.se

Abstract

Finding a class of structures that is rich enough for adequate linguistic representation yet restricted enough for efficient computational processing is an important problem for dependency parsing. In this paper, we present a transition system for 2-planar dependency trees – trees that can be decomposed into at most two planar graphs – and show that it can be used to implement a classifier-based parser that runs in linear time and outperforms a state-of-the-art transition-based parser on four data sets from the CoNLL-X shared task. In addition, we present an efficient method for determining whether an arbitrary tree is 2-planar and show that 99% or more of the trees in existing treebanks are 2-planar.

1 Introduction

Dependency-based syntactic parsing has become a widely used technique in natural language processing, and many different parsing models have been proposed in recent years (Yamada and Matsumoto, 2003; Nivre et al., 2004; McDonald et al., 2005a; Titov and Henderson, 2007; Martins et al., 2009). One of the unresolved issues in this area is the proper treatment of non-projective dependency trees, which seem to be required for an adequate representation of predicate-argument structure, but which undermine the efficiency of dependency parsing (Neuhaus and Bröker, 1997; Buch-Kromann, 2006; McDonald and Satta, 2007).

Caught between the Scylla of linguistically inadequate projective trees and the Charybdis of computationally intractable non-projective trees, some researchers have sought a middle ground by exploring classes of *mildly* non-projective dependency structures that strike a better balance between expressivity and complexity (Nivre, 2006; Kuhlmann and Nivre, 2006; Kuhlmann and Möhl, 2007; Havelka, 2007). Although these proposals seem to have a very good fit with linguistic data, in the sense that they often cover 99% or more of the structures found in existing treebanks, the development of efficient parsing algorithms for these classes has met with more limited success. For example, while both Kuhlmann and Satta (2009) and Gómez-Rodríguez et al. (2009) have shown how well-nested dependency trees with bounded gap degree can be parsed in polynomial time, the best time complexity for lexicalized parsing of this class remains a prohibitive $O(n^7)$, which makes the practical usefulness questionable.

In this paper, we explore another characterization of mildly non-projective dependency trees based on the notion of multiplanarity. This was originally proposed by Yli-Jyrä (2003) but has so far played a marginal role in the dependency parsing literature, because no algorithm was known for determining whether an arbitrary tree was m-planar, and no parsing algorithm existed for any constant value of m. The contribution of this paper is twofold. First, we present a procedure for determining the minimal number m such that a dependency tree is m-planar and use it to show that the overwhelming majority of sentences in dependency treebanks have a tree that is at most 2-planar. Secondly, we present a transition-based parsing algorithm for 2-planar dependency trees, developed in two steps. We begin by showing how the stack-based algorithm of Nivre (2003) can be generalized from projective to planar structures. We then extend the system by adding a second stack and show that the resulting system captures exactly the set of 2-planar structures. Although the contributions of this paper are mainly theoretical, we also present an empirical evaluation of the 2-planar parser, showing that it outperforms the projective parser on four data sets from the CoNLL-X shared task (Buchholz and Marsi, 2006).

Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 1492–1501,
Uppsala, Sweden, 11-16 July 2010.

2 Preliminaries

2.1 Dependency Graphs

Let $w = w_1 \dots w_n$ be an input string.[1] An **interval** (with endpoints i and j) of the string w is a set of the form $[i, j] = \{w_k \mid i \leq k \leq j\}$.

Definition 1. *A **dependency graph** for w is a directed graph $G = (V_w, E)$, where $V_w = [1, n]$ and $E \subseteq V_w \times V_w$.*

We call an edge (w_i, w_j) in a dependency graph G a **dependency link**[2] from w_i to w_j. We say that w_i is the **parent** (or **head**) of w_j and, conversely, that w_j is a syntactic **child** (or **dependent**) of w_i. For convenience, we write $w_i \rightarrow w_j \in E$ if the link (w_i, w_j) exists; $w_i \leftrightarrow w_j \in E$ if there is a link from w_i to w_j or from w_j to w_i; $w_i \rightarrow^* w_j \in E$ if there is a (possibly empty) directed path from w_i to w_j; and $w_i \leftrightarrow^* w_j \in E$ if there is a (possibly empty) path between w_i and w_j in the undirected graph underlying G (omitting reference to E when clear from the context). The **projection** of a node w_i, denoted $\lfloor w_i \rfloor$, is the set of reflexive-transitive dependents of w_i: $\lfloor w_i \rfloor = \{w_j \in V \mid w_i \rightarrow^* w_j\}$.

Most dependency representations do not allow arbitrary dependency graphs but typically require graphs to be acyclic and have at most one head per node. Such a graph is called a dependency *forest*.

Definition 2. *A dependency graph G for a string $w_1 \dots w_n$ is said to be a **forest** iff it satisfies:*

1. *Acyclicity: If $w_i \rightarrow^* w_j$, then not $w_j \rightarrow w_i$.*
2. *Single-head: If $w_j \rightarrow w_i$, then not $w_k \rightarrow w_i$ (for every $k \neq j$).*

Nodes in a forest that do not have a head are called **roots**. Some frameworks require that dependency forests have a unique root (i.e., are connected). Such a forest is called a **dependency tree**.

2.2 Projectivity

For reasons of computational efficiency, many dependency parsers are restricted to work with *projective* dependency structures, that is, forests in which the projection of each node corresponds to a contiguous substring of the input:

Definition 3. *A dependency forest G for a string $w_1 \dots w_n$ is **projective** iff $\lfloor w_i \rfloor$ is an interval for every word $w_i \in [1, n]$.*

Projective dependency trees correspond to the set of structures that can be induced from lexicalised context-free derivations (Kuhlmann, 2007; Gaifman, 1965). Like context-free grammars, projective dependency trees are not sufficient to represent all the linguistic phenomena observed in natural languages, but they have the advantage of being efficiently parsable: their parsing problem can be solved in cubic time with chart parsing techniques (Eisner, 1996; Gómez-Rodríguez et al., 2008), while in the case of general non-projective dependency forests, it is only tractable under strong independence assumptions (McDonald et al., 2005b; McDonald and Satta, 2007).

2.3 Planarity

The concept of *planarity* (Sleator and Temperley, 1993) is closely related to projectivity[3] and can be informally defined as the property of a dependency forest whose links can be drawn *above* the words without crossing.[4] To define planarity more formally, we first define **crossing links** as follows: let (w_i, w_k) and (w_j, w_l) be dependency links in a dependency graph G. Without loss of generality, we assume that $min(i, k) \leq min(j, l)$. Then, the links are said to be crossing if $min(i, k) < min(j, l) < max(i, k) < max(j, l)$.

Definition 4. *A dependency graph is **planar** iff it does not contain a pair of crossing links.*

2.4 Multiplanarity

The concept of planarity on its own does not seem to be very relevant as an extension of projectivity for practical dependency parsing. According to the results by Kuhlmann and Nivre (2006), most non-projective structures in dependency treebanks are also non-planar, so being able to parse planar structures will only give us a modest improvement in coverage with respect to a projective parser. However, our interest in planarity is motivated by the fact that it can be generalised to **multiplanarity** (Yli-Jyrä, 2003):

[1] For notational convenience, we will assume throughout the paper that all symbols in an input string are distinct, i.e., $i \neq j \Leftrightarrow w_i \neq w_j$. This can be guaranteed in practice by annotating each terminal symbol with its position in the input.

[2] In practice, dependency links are usually **labeled**, but to simplify the presentation we will ignore labels throughout most of the paper. However, all the results and algorithms presented can be applied to labeled dependency graphs and will be so applied in the experimental evaluation.

[3] For dependency forests that are extended with a unique artificial root located at position 0, as is commonly done, the two notions are equivalent.

[4] Planarity in the context of dependency structures is not to be confused with the homonymous concept in graph theory, which does not restrict links to be drawn above the nodes.

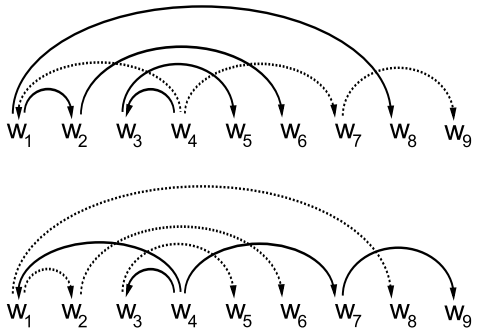

Figure 1: A 2-planar dependency structure with two different ways of distributing its links into two planes (represented by solid and dotted lines).

Definition 5. *A dependency graph $G = (V, E)$ is **m-planar** iff there exist planar dependency graphs $G_1 = (V, E_1), \ldots, G_m = (V, E_m)$ (called **planes**) such that $E = E_1 \cup \cdots \cup E_m$.*

Intuitively, we can associate planes with colours and say that a dependency graph G is m-planar if it is possible to assign one of m colours to each of its links in such a way that links with the same colour do not cross. Note that there may be multiple ways of dividing an m-planar graph into planes, as shown in the example of Figure 1.

3 Determining Multiplanarity

Several constraints on non-projective dependency structures have been proposed recently that seek a good balance between parsing efficiency and coverage of non-projective phenomena present in natural language treebanks. For example, Kuhlmann and Nivre (2006) and Havelka (2007) have shown that the vast majority of structures present in existing treebanks are well-nested and have a small gap degree (Bodirsky et al., 2005), leading to an interest in parsers for these kinds of structures (Gómez-Rodríguez et al., 2009). No similar analysis has been performed for m-planar structures, although Yli-Jyrä (2003) provides evidence that all except two structures in the Danish dependency treebank are at most 3-planar. However, his analysis is based on constraints that restrict the possible ways of assigning planes to dependency links, and he is not guaranteed to find the minimal number m for which a given structure is m-planar.

In this section, we provide a procedure for finding the minimal number m such that a dependency graph is m-planar and use it to show that the vast majority of sentences in dependency treebanks are

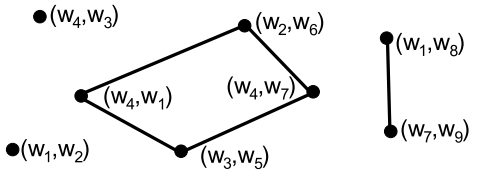

Figure 2: The crossings graph corresponding to the dependency structure of Figure 1.

at most 2-planar, with a coverage comparable to that of well-nestedness. The idea is to reduce the problem of determining whether a dependency graph $G = (V, E)$ is m-planar, for a given value of m, to a standard graph colouring problem. Consider first the following undirected graph:

$U(G) = (E, C)$ where
$C = \{\{e_i, e_j\} \mid e_i, e_j \text{ are crossing links in } G\}$

This graph, which we call the **crossings graph** of G, has one node corresponding to each link in the dependency graph G, with an undirected link between two nodes if they correspond to crossing links in G. Figure 2 shows the crossings graph of the 2-planar structure in Figure 1.

As noted in Section 2.4, a dependency graph G is m-planar if each of its links can be assigned one of m colours in such a way that links with the same colours do not cross. In terms of the crossings graph, this means that G is m-planar if each of the *nodes* of $U(G)$ can be assigned one of m colours such that no two neighbours have the same colour. This amounts to solving the well-known k-colouring problem for $U(G)$, where $k = m$.

For $k = 1$ the problem is trivial: a graph is 1-colourable only if it has no edges. For $k = 2$, the problem can be solved in time linear in the size of the graph by simple breadth-first search. Given a graph $U = (V, E)$, we pick an arbitrary node v and give it one of two colours. This forces us to give the other colour to all its neighbours, the first colour to the neighbours' neighbours, and so on. This process continues until we have processed all the nodes in the connected component of v. If this has resulted in assigning two different colours to the same node, the graph is not 2-colourable. Otherwise, we have obtained a 2-colouring of the connected component of U that contains v. If there are still unprocessed nodes, we repeat the process by arbitrarily selecting one of them, continue with the rest of the connected components, and in this way obtain a 2-colouring of the whole graph if it

Language	Structures	Non-Projective	Not Planar	Not 2-Planar	Not 3-Pl.	Not 4-pl.	Ill-nested
Arabic	2995	205 (6.84%)	158 (5.28%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	1 (0.03%)
Czech	87889	20353 (23.16%)	16660 (18.96%)	82 (0.09%)	0 (0.00%)	0 (0.00%)	96 (0.11%)
Danish	5512	853 (15.48%)	827 (15.00%)	1 (0.02%)	1 (0.02%)	0 (0.00%)	6 (0.11%)
Dutch	13349	4865 (36.44%)	4115 (30.83%)	162 (1.21%)	1 (0.01%)	0 (0.00%)	15 (0.11%)
German	39573	10927 (27.61%)	10908 (27.56%)	671 (1.70%)	0 (0.00%)	0 (0.00%)	419 (1.06%)
Portuguese	9071	1718 (18.94%)	1713 (18.88%)	8 (0.09%)	0 (0.00%)	0 (0.00%)	7 (0.08%)
Swedish	6159	293 (4.76%)	280 (4.55%)	5 (0.08%)	0 (0.00%)	0 (0.00%)	14 (0.23%)
Turkish	5510	657 (11.92%)	657 (11.92%)	10 (0.18%)	0 (0.00%)	0 (0.00%)	20 (0.36%)

Table 1: Proportion of dependency trees classified by projectivity, planarity, m-planarity and ill-nestedness in treebanks for Arabic (Hajič et al., 2004), Czech (Hajič et al., 2006), Danish (Kromann, 2003), Dutch (van der Beek et al., 2002), German (Brants et al., 2002), Portuguese (Afonso et al., 2002), Swedish (Nilsson et al., 2005) and Turkish (Oflazer et al., 2003; Atalay et al., 2003).

exists. Since this process can be completed by visiting each node and edge of the graph U once, its complexity is $O(V + E)$. The crossings graph of a dependency graph with n nodes can trivially be built in time $O(n^2)$ by checking each pair of dependency links to determine if they cross, and cannot contain more than n^2 edges, which means that we can check if the dependency graph for a sentence of length n is 2-planar in $O(n^2)$ time.

For $k > 2$, the k-colouring problem is known to be NP-complete (Karp, 1972). However, we have found this not to be a problem when measuring multiplanarity in natural language treebanks, since the effective problem size can be reduced by noting that each connected component of the crossings graph can be treated separately, and that nodes that are not part of a cycle need not be considered.[5] Given that non-projective sentences in natural language tend to have a small proportion of non-projective links (Nivre and Nilsson, 2005), the connected components of their crossings graphs are very small, and k-colourings for them can quickly be found by brute-force search.

By applying these techniques to dependency treebanks of several languages, we obtain the data shown in Table 1. As we can see, the coverage provided by the 2-planarity constraint is comparable to that of well-nestedness. In most of the treebanks, well over 99% of the sentences are 2-planar, and 3-planarity has almost total coverage. As we will see below, the class of 2-planar dependency structures not only has good coverage of linguistic phenomena in existing treebanks but is also efficiently parsable with transition-based parsing methods, making it a practically interesting subclass of non-projective dependency structures.

[5] If we have a valid colouring for all the cycles in the graph, the rest of the nodes can be safely coloured by breadth-first search as in the $k = 2$ case.

4 Parsing 1-Planar Structures

In this section, we present a deterministic linear-time parser for planar dependency structures. The parser is a variant of Nivre's arc-eager projective parser (Nivre, 2003), modified so that it can also handle graphs that are planar but not projective. As seen in Table 1, this only gives a modest improvement in coverage compared to projective parsing, so the main interest of this algorithm lies in the fact that it can be generalised to deal with 2-planar structures, as shown in the next section.

4.1 Transition Systems

In the transition-based framework of Nivre (2008), a deterministic dependency parser is defined by a non-deterministic **transition system**, specifying a set of elementary operations that can be executed during the parsing process, and an **oracle** that deterministically selects a single transition at each choice point of the parsing process.

Definition 6. *A **transition system** for dependency parsing is a quadruple $S = (C, T, c_s, C_t)$ where*

1. *C is a set of possible parser **configurations**,*
2. *T is a set of **transitions**, each of which is a partial function $t : C \rightarrow C$,*
3. *c_s is a function that maps each input sentence w to an **initial configuration** $c_s(w) \in C$,*
4. *$C_t \subseteq C$ is a set of **terminal configurations**.*

Definition 7. *An **oracle** for a transition system $S = (C, T, c_s, C_t)$ is a function $o : C \rightarrow T$.*

An input sentence w can be parsed using a transition system $S = (C, T, c_s, C_t)$ and an oracle o by starting in the initial configuration $c_s(w)$, calling the oracle function on the current configuration c, and updating the configuration by applying the transition $o(c)$ returned by the oracle. This process is repeated until a terminal configuration is

Initial configuration: $c_s(w_1 \dots w_n) = \langle [], [w_1 \dots w_n], \emptyset \rangle$

Terminal configurations: $C_f = \{\langle \Sigma, [], A \rangle \in C\}$

Transitions:

SHIFT $\langle \Sigma, w_i|B, A\rangle \Rightarrow \langle \Sigma|w_i, B, A\rangle$

REDUCE $\langle \Sigma|w_i, B, A\rangle \Rightarrow \langle \Sigma, B, A\rangle$

LEFT-ARC $\langle \Sigma|w_i, w_j|B, A\rangle \Rightarrow \langle \Sigma|w_i, w_j|B, A \cup \{(w_j, w_i)\}\rangle$
only if $\nexists k|(w_k, w_i) \in A$ (single-head) and not $w_i \leftrightarrow^* w_j \in A$ (acyclicity).

RIGHT-ARC $\langle \Sigma|w_i, w_j|B, A\rangle \Rightarrow \langle \Sigma|w_i, w_j|B, A \cup \{(w_i, w_j)\}\rangle$
only if $\nexists k|(w_k, w_j) \in A$ (single-head) and not $w_i \leftrightarrow^* w_j \in A$ (acyclicity).

Figure 3: Transition system for planar dependency parsing.

reached, and the dependency analysis of the sentence is defined by the terminal configuration.

Each sequence of configurations that the parser can traverse from an initial configuration to a terminal configuration for some input w is called a **transition sequence**. If we associate each configuration c of a transition system $S = (C, T, c_s, C_t)$ with a dependency graph $g(c)$, we can say that S is **sound** for a class of dependency graphs $\mathcal{G}$ if, for every sentence w and transition sequence $(c_s(w), c_1, \dots, c_f)$ of S, $g(c_f)$ is in $\mathcal{G}$, and that S is **complete** for $\mathcal{G}$ if, for every sentence w and dependency graph $G \in \mathcal{G}$ for w, there is a transition sequence $(c_s(w), c_1, \dots, c_f)$ such that $g(c_f) = G$. A transition system that is sound and complete for $\mathcal{G}$ is said to be **correct** for $\mathcal{G}$.

Note that, apart from a correct transition system, a practical parser needs a good oracle to achieve the desired results, since a transition system only specifies how to reach all the possible dependency graphs that could be associated to a sentence, but not how to select the correct one. Oracles for practical parsers can be obtained by training classifiers on treebank data (Nivre et al., 2004).

4.2 A Transition System for Planar Structures

A correct transition system for the class of planar dependency forests can be obtained as a variant of the arc-eager projective system by Nivre (2003). As in that system, the set of configurations of the planar transition system is the set of all triples $c = \langle \Sigma, B, A\rangle$ such that Σ and B are disjoint lists of words from V_w (for some input w), and A is a set of dependency links over V_w. The list B, called the **buffer**, is initialised to the input string and is used to hold the words that are still to be read from the input. The list Σ, called the **stack**, is initially empty and holds words that have dependency links pending to be created. The system is shown in Figure 3, where we use the notation $\Sigma|w_i$ for a stack with top w_i and tail Σ, and we invert the notation for the buffer for clarity (i.e., $w_i|B$ is a buffer with top w_i and tail B).

The system reads the input from left to right and creates links in a left-to-right order by executing its four transitions:

1. SHIFT: pops the first (leftmost) word in the buffer, and pushes it to the stack.
2. LEFT-ARC: adds a link from the first word in the buffer to the top of the stack.
3. RIGHT-ARC: adds a link from the top of the stack to the first word in the buffer.
4. REDUCE: pops the top word from the stack, implying that we have finished building links to or from it.

Note that the planar parser's transitions are more fine-grained than those of the arc-eager projective parser by Nivre (2003), which pops the stack as part of its LEFT-ARC transition and shifts a word as part of its RIGHT-ARC transition. Forcing these actions after creating dependency links rules out structures whose root is covered by a dependency link, which are planar but not projective. In order to support these structures, we therefore simplify the ARC transitions (LEFT-ARC and RIGHT-ARC) so that they only create an arc. For the same reason, we remove the constraint in Nivre's parser by which words without a head cannot be reduced. This has the side effect of making the parser able to output cyclic graphs. Since we are interested in planar dependency *forests*, which do not contain cycles, we only apply ARC transitions after checking that there is no undirected path between the nodes to be linked. This check can be done without affecting the linear-time complexity of the

parser by storing the weakly connected component of each node in $g(c)$.

The fine-grained transitions used by this parser have also been used by Sagae and Tsujii (2008) to parse DAGs. However, the latter parser differs from ours in the constraints, since it does not allow the reduction of words without a head (disallowing forests with covered roots) and does not enforce the acyclicity constraint (which is guaranteed by post-processing the graphs to break cycles).

4.3 Correctness and Complexity

For reasons of space, we can only give a sketch of the correctness proof. We wish to prove that the planar transition system is sound and complete for the set $\mathcal{F}_p$ of all planar dependency forests. To prove soundness, we have to show that, for every sentence w and transition sequence $(c_s(w), c_1, \ldots, c_f)$, the graph $g(c_f)$ associated with c_f is in $\mathcal{F}_p$. We take the graph associated with a configuration $c = (\Sigma, B, A)$ to be $g(c) = (V_w, A)$. With this, we prove the stronger claim that $g(c) \in \mathcal{F}_p$ for every configuration c that belongs to some transition sequence starting with $c_s(w)$. This amounts to showing that in every configuration c reachable from $c_s(w)$, $g(c)$ meets the following three conditions that characterise a planar dependency forest: (1) $g(c)$ does not contain nodes with more than one head; (2) $g(c)$ is acyclic; and (3) $g(c)$ contains no crossing links. (1) is trivially guaranteed by the single-head constraint; (2) follows from (1) and the acyclicity constraint; and (3) can be established by proving that there is no transition sequence that will invoke two ARC transitions on node pairs that would create crossing links. At the point when a link from w_i to w_j is created, we know that all the words strictly located between w_i and w_j are not in the stack or in the buffer, so no links can be created to or from them.

To prove completeness, we show that every planar dependency forest $G = (V, E) \in \mathcal{F}_p$ for a sentence w can be produced by applying the oracle function that maps a configuration $\langle \Sigma | w_i, w_j | B, A \rangle$ to:

1. LEFT-ARC if $w_j \rightarrow w_i \in (E \setminus A)$,
2. RIGHT-ARC if $w_i \rightarrow w_j \in (E \setminus A)$,
3. REDUCE if $\exists x_{[x<i]}[w_x \leftrightarrow w_j \in (E \setminus A)]$,
4. SHIFT otherwise.

We show completeness by setting the following invariants on transitions traversed by the application of the oracle:

1. $\forall a, b_{[a,b<j]}[w_a \leftrightarrow w_b \in E \Rightarrow w_a \leftrightarrow w_b \in A]$
2. $[w_i \leftrightarrow w_j \in A \Rightarrow$
   $\forall k_{[i<k<j]}[w_k \leftrightarrow w_j \in E \Rightarrow w_k \leftrightarrow w_j \in A]]$
3. $\forall k_{[k<j]}[w_k \notin \Sigma \Rightarrow$
   $\forall l_{[l>k]}[w_k \leftrightarrow w_l \in E \Rightarrow w_k \leftrightarrow w_l \in A]]$

We can show that each branch of the oracle function keeps these invariants true. When we reach a terminal configuration (which always happens after a finite number of transitions, since every transition generating a configuration $c = \langle \Sigma, B, A \rangle$ decreases the value of the variant function $|E| + |\Sigma| + 2|B| - |A|$), it can be deduced from the invariant that $A = E$, which proves completeness.

The worst-case complexity of a deterministic transition-based parser is given by an upper bound on transition sequence length (Nivre, 2008). For the planar system, like its projective counterpart, the length is clearly $O(n)$ (where n is the number of input words), since there can be no more than n SHIFT transitions, n REDUCE transitions, and n ARC transitions in a transition sequence.

5 Parsing 2-Planar Structures

The planar parser introduced in the previous section can be extended to parse all 2-planar dependency structures by adding a second stack to the system and making REDUCE and ARC transitions apply to only one of the stacks at a time. This means that the set of links created in the context of each individual stack will be planar, but pairs of links created in different stacks are allowed to cross. In this way, the parser will build a 2-planar dependency forest by using each of the stacks to construct one of its two planes.

The 2-planar transition system, shown in Figure 4, has configurations of the form $\langle \Sigma_0, \Sigma_1, B, A \rangle$, where we call Σ_0 the **active stack** and Σ_1 the **inactive stack**, and the following transitions:

1. SHIFT: pops the first (leftmost) word in the buffer, and pushes it to *both* stacks.
2. LEFT-ARC: adds a link from the first word in the buffer to the top of the *active* stack.
3. RIGHT-ARC: adds a link from the top of the *active* stack to the first word in the buffer.
4. REDUCE: pops the top word from the *active* stack, implying that we have added all links to or from it on the plane tied to that stack.
5. SWITCH: makes the active stack inactive and vice versa, changing the plane the parser is working with.

Initial configuration: $c_s(w_1 \dots w_n) = \langle [], [], [w_1 \dots w_n], \emptyset \rangle$

Terminal configurations: $C_f = \{\langle \Sigma_0, \Sigma_1, [], A \rangle \in C\}$

Transitions:

SHIFT $\langle \Sigma_0, \Sigma_1, w_i|B, A\rangle \Rightarrow \langle \Sigma_0|w_i, \Sigma_1|w_i, B, A\rangle$

REDUCE $\langle \Sigma_0|w_i, \Sigma_1, B, A\rangle \Rightarrow \langle \Sigma_0, \Sigma_1, B, A\rangle$

LEFT-ARC $\langle \Sigma_0|w_i, \Sigma_1, w_j|B, A\rangle \Rightarrow \langle \Sigma_0|w_i, \Sigma_1, w_j|B, A \cup \{(w_j, w_i)\}\rangle$
only if $\nexists k \mid (w_k, w_i) \in A$ (single-head) and not $w_i \leftrightarrow^* w_j \in A$ (acyclicity).

RIGHT-ARC $\langle \Sigma_0|w_i, \Sigma_1, w_j|B, A\rangle \Rightarrow \langle \Sigma_0|w_i, \Sigma_1, w_j|B, A \cup \{(w_i, w_j)\}\rangle$
only if $\nexists k|(w_k, w_j) \in A$ (single-head) and not $w_i \leftrightarrow^* w_j \in A$ (acyclicity).

SWITCH $\langle \Sigma_0, \Sigma_1, B, A\rangle \Rightarrow \langle \Sigma_1, \Sigma_0, B, A\rangle$

Figure 4: Transition system for 2-planar dependency parsing.

5.1 Correctness and Complexity

As in the planar case, we provide a brief sketch of the proof that the transition system in Figure 4 is correct for the set $\mathcal{F}_{2p}$ of 2-planar dependency forests. Soundness follows from a reasoning analogous to the planar case, but applying the proof of planarity separately to each stack. In this way, we prove that the sets of dependency links created by linking to or from the top of each of the two stacks are always planar graphs, and thus their union (which is the dependency graph stored in A) is 2-planar. This, together with the single-head and acyclicity constraints, guarantees that the dependency graphs associated with reachable configurations are always 2-planar dependency forests.

For completeness, we assume an extended form of the transition system where transitions take the form $\langle \Sigma_0, \Sigma_1, B, A, p\rangle$, where p is a flag taking values in $\{0, 1\}$ which equals 0 for initial configurations and gets flipped by each application of a SWITCH transition. Then we show that every 2-planar dependency forest $G \in \mathcal{F}_{2p}$, with planes $G_0 = (V, E_0)$ and $G_1 = (V, E_1)$, can be produced by this system by applying the oracle function that maps a configuration $\langle \Sigma_0|w_i, \Sigma_1, w_j|B, A, p\rangle$ to:

1. LEFT-ARC if $w_j \rightarrow w_i \in (E_p \setminus A)$,
2. RIGHT-ARC if $w_i \rightarrow w_j \in (E_p \setminus A)$,
3. REDUCE if $\exists x_{[x<i]}[w_x \leftrightarrow w_j \in (E_p \setminus A) \wedge \neg\exists y_{[x<y\leq i]}[w_y \leftrightarrow w_j \in (E_{\overline{p}} \setminus A)]]$,
4. SWITCH if $\exists x < j : (w_x, w_j)$ or $(w_j, w_x) \in (E_{\overline{p}} \setminus A)$,
5. SHIFT otherwise.

This can be shown by employing invariants analogous to the planar case, with the difference that the third invariant applies to each stack and its corresponding plane: if Σ_y is associated with the plane E_x,[6] we have:

3. $\forall k_{[k<j]}[w_k \notin \Sigma_y] \Rightarrow$
   $\forall l_{[l>k]}[w_k \leftrightarrow w_l \in E_x] \Rightarrow [w_k \leftrightarrow w_l \in A]$

Since the presence of the flag p in configurations does not affect the set of dependency graphs generated by the system, the completeness of the system extended with the flag p implies that of the system in Figure 4.

We can show that the complexity of the 2-planar system is $O(n)$ by the same kind of reasoning as for the 1-planar system, with the added complication that we must constrain the system to prevent two adjacent SWITCH transitions. In fact, without this restriction, the parser is not even guaranteed to terminate.

5.2 Implementation

In practical settings, oracles for transition-based parsers can be approximated by classifiers trained on treebank data (Nivre, 2008). To do this, we need an oracle that will generate transition sequences for gold-standard dependency graphs. In the case of the planar parser of Section 4.2, the oracle of 4.3 is suitable for this purpose. However, in the case of the 2-planar parser, the oracle used for the completeness proof in Section 5.1 cannot be used directly, since it requires the gold-standard trees to be divided into two planes in order to generate a transition sequence.

Of course, it is possible to use the algorithm presented in Section 3 to obtain a division of sentences into planes. However, for training purposes and to obtain a robust behaviour if non-2-planar

[6] The plane corresponding to each stack in a configuration changes with each SWITCH transition: Σ_x is associated with E_x in configurations where $p = 0$, and with $E_{\overline{x}}$ in those where $p = 1$.

Parser	Czech				Danish				German				Portuguese			
	LAS	UAS	NPP	NPR	LAS	UAS	NPP	NPR	LAS	UAS	NPP	NPR	LAS	UAS	NPP	NPR
2-planar	79.24	85.30	68.9	**60.7**	**83.81**	88.50	**66.7**	20.0	**86.50**	**88.84**	57.1	**45.8**	87.04	**90.82**	82.8	33.8
Malt P	78.18	84.12	–	–	83.31	88.30	–	–	85.36	88.06	–	–	86.60	90.20	–	–
Malt PP	**79.80**	**85.70**	**76.7**	56.1	83.67	**88.52**	41.7	**25.0**	85.76	88.66	**58.1**	40.7	**87.08**	90.66	**83.3**	**46.2**

Table 2: Parsing accuracy for 2-planar parser in comparison to MaltParser with (PP) and without (P) pseudo-projective transformations. LAS = labeled attachment score; UAS = unlabeled attachment score; NPP = precision on non-projective arcs; NPR = recall on non-projective arcs.

sentences are found, it is more convenient that the oracle can distribute dependency links into the planes incrementally, and that it produces a distribution of links that only uses SWITCH transitions when it is strictly needed to account for non-planarity. Thus we use a more complex version of the oracle which performs a search in the crossings graph to check if a dependency link can be built on the plane of the active stack, and only performs a switch when this is not possible. This has proved to work well in practice, as will be observed in the results in the next section.

6 Empirical Evaluation

In order to get a first estimate of the empirical accuracy that can be obtained with transition-based 2-planar parsing, we have evaluated the parser on four data sets from the CoNLL-X shared task (Buchholz and Marsi, 2006): Czech, Danish, German and Portuguese. As our baseline, we take the strictly projective arc-eager transition system proposed by Nivre (2003), as implemented in the freely available MaltParser system (Nivre et al., 2006a), with and without the pseudo-projective parsing technique for recovering non-projective dependencies (Nivre and Nilsson, 2005). For the two baseline systems, we use the parameter settings used by Nivre et al. (2006b) in the original shared task, where the pseudo-projective version of MaltParser was one of the two top performing systems (Buchholz and Marsi, 2006). For our 2-planar parser, we use the same kernelized SVM classifiers as MaltParser, using the LIBSVM package (Chang and Lin, 2001), with feature models that are similar to MaltParser but extended with features defined over the second stack.[7]

In Table 2, we report labeled (LAS) and unlabeled (UAS) attachment score on the four languages for all three systems. For the two systems that are capable of recovering non-projective dependencies, we also report precision (NPP) and recall (NPR) specifically on non-projective dependency arcs. The results show that the 2-planar parser outperforms the strictly projective variant of MaltParser on all metrics for all languages, and that it performs on a par with the pseudo-projective variant with respect to both overall attachment score and precision and recall on non-projective dependencies. These results look very promising in view of the fact that very little effort has been spent on optimizing the training oracle and feature model for the 2-planar parser so far.

It is worth mentioning that the 2-planar parser has two advantages over the pseudo-projective parser. The first is simplicity, given that it is based on a single transition system and makes a single pass over the input, whereas the pseudo-projective parsing technique involves preprocessing of training data and post-processing of parser output (Nivre and Nilsson, 2005). The second is the fact that it parses a well-defined class of dependency structures, with known coverage[8], whereas no formal characterization exists of the class of structures parsable by the pseudo-projective parser.

7 Conclusion

In this paper, we have presented an efficient algorithm for deciding whether a dependency graph is 2-planar and a transition-based parsing algorithm that is provably correct for 2-planar dependency forests, neither of which existed in the literature before. In addition, we have presented empirical results showing that the class of 2-planar dependency forests includes the overwhelming majority of structures found in existing treebanks and that a deterministic classifier-based implementation of the 2-planar parser gives state-of-the-art accuracy on four different languages.

[7] Complete information about experimental settings can be found at http://stp.lingfil.uu.se/ nivre/exp/.

[8] If more coverage is desired, the 2-planar parser can be generalised to m-planar structures for larger values of m by adding additional stacks. However, this comes at the cost of more complex training models, making the practical interest of increasing m beyond 2 dubious.

Acknowledgments

The first author has been partially supported by Ministerio de Educación y Ciencia and FEDER (HUM2007-66607-C04) and Xunta de Galicia (PGIDIT07SIN005206PR, Rede Galega de Procesamento da Linguaxe e Recuperación de Información, Rede Galega de Lingüística de Corpus, Bolsas Estadías INCITE/FSE cofinanced).

References

Susana Afonso, Eckhard Bick, Renato Haber, and Diana Santos. 2002. "Floresta sintá(c)tica": a treebank for Portuguese. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2002)*, pages 1968–1703, Paris, France. ELRA.

Nart B. Atalay, Kemal Oflazer, and Bilge Say. 2003. The annotation process in the Turkish treebank. In *Proceedings of EACL Workshop on Linguistically Interpreted Corpora (LINC-03)*, pages 243–246, Morristown, NJ, USA. Association for Computational Linguistics.

Leonoor van der Beek, Gosse Bouma, Robert Malouf, and Gertjan van Noord. 2002. The Alpino dependency treebank. In *Language and Computers, Computational Linguistics in the Netherlands 2001. Selected Papers from the Twelfth CLIN Meeting*, pages 8–22, Amsterdam, the Netherlands. Rodopi.

Manuel Bodirsky, Marco Kuhlmann, and Mathias Möhl. 2005. Well-nested drawings as models of syntactic structure. In *10th Conference on Formal Grammar and 9th Meeting on Mathematics of Language*, Edinburgh, Scotland, UK.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The tiger treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories, September 20-21*, Sozopol, Bulgaria.

Matthias Buch-Kromann. 2006. *Discontinuous Grammar: A Model of Human Parsing and Language Acquisition*. Ph.D. thesis, Copenhagen Business School.

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)*, pages 149–164.

Chih-Chung Chang and Chih-Jen Lin, 2001. *LIBSVM: A Library for Support Vector Machines*. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, San Francisco, CA, USA, August. ACL / Morgan Kaufmann.

Haim Gaifman. 1965. Dependency systems and phrase-structure systems. *Information and Control*, 8:304–337.

Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2008. A deductive approach to dependency parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL'08:HLT)*, pages 968–976, Morristown, NJ, USA. Association for Computational Linguistics.

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 (EACL)*, pages 291–299.

Jan Hajič, Otakar Smrž, Petr Zemánek, Jan Šnaidauf, and Emanuel Beška. 2004. Prague Arabic dependency treebank: Development in data and tools. In *Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools*, pages 110–117.

Jan Hajič, Jarmila Panevová, Eva Hajičová, Jarmila Panevová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, and Marie Mikulová. 2006. Prague Dependency Treebank 2.0. CDROM CAT: LDC2006T01, ISBN 1-58563-370-4. Linguistic Data Consortium.

Jiri Havelka. 2007. Beyond projectivity: Multilingual evaluation of constraints and measures on non-projective structures. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 608–615.

Richard M. Karp. 1972. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press.

Matthias T. Kromann. 2003. The Danish dependency treebank and the underlying linguistic theory. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 217–220, Växjö, Sweden. Växjö University Press.

Marco Kuhlmann and Mathias Möhl. 2007. Mildly context-sensitive dependency languages. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 160–167.

Marco Kuhlmann and Joakim Nivre. 2006. Mildly non-projective dependency structures. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 507–514.

Marco Kuhlmann and Giorgio Satta. 2009. Treebank grammar techniques for non-projective dependency parsing. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 478–486.

Marco Kuhlmann. 2007. *Dependency Structures and Lexicalized Grammars*. Doctoral dissertation, Saarland University, Saarbrücken, Germany.

Andre Martins, Noah Smith, and Eric 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-IJCNLP)*, pages 342–350.

Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*, pages 122–131.

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

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *HLT/EMNLP 2005: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530, Morristown, NJ, USA. Association for Computational Linguistics.

Peter Neuhaus and Norbert Bröker. 1997. The complexity of recognition of linguistically adequate dependency grammars. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL) and the 8th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 337–343.

Jens Nilsson, Johan Hall, and Joakim Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *Proceedings of NODALIDA 2005 Special Session on Treebanks*, pages 119–132. Samfundslitteratur, Frederiksberg, Denmark, May.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *ACL '05: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 99–106, Morristown, NJ, USA. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 49–56, Morristown, NJ, USA. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006a. MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 2216–2219.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006b. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Joakim Nivre. 2006. Constraints on non-projective dependency graphs. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 73–80.

Joakim Nivre. 2008. Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553.

Kemal Oflazer, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür. 2003. Building a Turkish treebank. In A. Abeille (ed.), *Building and Exploiting Syntactically-annotated Corpora*, pages 261–277, Dordrecht, the Netherlands. Kluwer.

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *COLING '08: Proceedings of the 22nd International Conference on Computational Linguistics*, pages 753–760, Morristown, NJ, USA. Association for Computational Linguistics.

Daniel Sleator and Davy Temperley. 1993. Parsing English with a Link Grammar. In *Proceedings of the Third International Workshop on Parsing Technologies (IWPT'93)*, pages 277–292. ACL/SIGPARSE.

Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT)*, pages 144–155.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.

Anssi Mikael Yli-Jyrä. 2003. Multiplanarity – a model for dependency structures in treebanks. In Joakim Nivre and Erhard Hinrichs, editors, *TLT 2003. Proceedings of the Second Workshop on Treebanks and Linguistic Theories*, volume 9 of *Mathematical Modelling in Physics, Engineering and Cognitive Sciences*, pages 189–200, Växjö, Sweden, 14-15 November. Växjö University Press.