Dependency Parsing with Undirected Graphs

Carlos Gómez-Rodríguez
Departamento de Computación
Universidade da Coruña
Campus de Elviña, 15071
A Coruña, Spain
carlos.gomez@udc.es

Daniel Fernández-González
Departamento de Informática
Universidade de Vigo
Campus As Lagoas, 32004
Ourense, Spain
danifg@uvigo.es

Abstract

We introduce a new approach to transition-based dependency parsing in which the parser does not directly construct a dependency structure, but rather an undirected graph, which is then converted into a directed dependency tree in a post-processing step. This alleviates error propagation, since undirected parsers do not need to observe the single-head constraint.

Undirected parsers can be obtained by simplifying existing transition-based parsers satisfying certain conditions. We apply this approach to obtain undirected variants of the planar and 2-planar parsers and of Covington's non-projective parser. We perform experiments on several datasets from the CoNLL-X shared task, showing that these variants outperform the original directed algorithms in most of the cases.

1 Introduction

Dependency parsing has proven to be very useful for natural language processing tasks. Data-driven dependency parsers such as those by Nivre et al. (2004), McDonald et al. (2005), Titov and Henderson (2007), Martins et al. (2009) or Huang and Sagae (2010) are accurate and efficient, they can be trained from annotated data without the need for a grammar, and they provide a simple representation of syntax that maps to predicate-argument structure in a straightforward way.

In particular, **transition-based** dependency parsers (Nivre, 2008) are a type of dependency parsing algorithms which use a model that scores transitions between parser states. Greedy deterministic search can be used to select the transition to be taken at each state, thus achieving linear or quadratic time complexity.

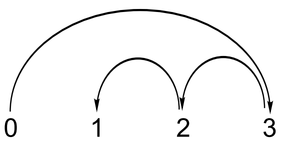

Figure 1: An example dependency structure where transition-based parsers enforcing the single-head constraint will incur in error propagation if they mistakenly build a dependency link $1 \rightarrow 2$ instead of $2 \rightarrow 1$ (dependency links are represented as arrows going from head to dependent).

It has been shown by McDonald and Nivre (2007) that such parsers suffer from **error propagation**: an early erroneous choice can place the parser in an incorrect state that will in turn lead to more errors. For instance, suppose that a sentence whose correct analysis is the dependency graph in Figure 1 is analyzed by any bottom-up or left-to-right transition-based parser that outputs dependency trees, therefore obeying the **single-head constraint** (only one incoming arc is allowed per node). If the parser chooses an erroneous transition that leads it to build a dependency link from 1 to 2 instead of the correct link from 2 to 1, this will lead it to a state where the single-head constraint makes it illegal to create the link from 3 to 2. Therefore, a single erroneous choice will cause two attachment errors in the output tree.

With the goal of minimizing these sources of errors, we obtain novel *undirected* variants of several parsers; namely, of the planar and 2-planar parsers by Gómez-Rodríguez and Nivre (2010) and the non-projective list-based parser described by Nivre (2008), which is based on Covington's algorithm (Covington, 2001). These variants work by collapsing the LEFT-ARC and

RIGHT-ARC transitions in the original parsers, which create right-to-left and left-to-right dependency links, into a single ARC transition creating an undirected link. This has the advantage that the single-head constraint need not be observed during the parsing process, since the directed notions of head and dependent are lost in undirected graphs. This gives the parser more freedom and can prevent situations where enforcing the constraint leads to error propagation, as in Figure 1.

On the other hand, these new algorithms have the disadvantage that their output is an undirected graph, which has to be post-processed to recover the direction of the dependency links and generate a valid dependency tree. Thus, some complexity is moved from the parsing process to this post-processing step; and each undirected parser will outperform the directed version only if the simplification of the parsing phase is able to avoid more errors than are generated by the post-processing. As will be seen in latter sections, experimental results indicate that this is in fact the case.

The rest of this paper is organized as follows: Section 2 introduces some notation and concepts that we will use throughout the paper. In Section 3, we present the undirected versions of the parsers by Gómez-Rodríguez and Nivre (2010) and Nivre (2008), as well as some considerations about the feature models suitable to train them. In Section 4, we discuss post-processing techniques that can be used to recover dependency trees from undirected graphs. Section 5 presents an empirical study of the performance obtained by these parsers, and Section 6 contains a final discussion.

2 Preliminaries

2.1 Dependency Graphs

Let $w = w_1 \dots w_n$ be an input string. A **dependency graph** for w is a directed graph $G = (V_w, E)$, where $V_w = \{0, \dots, n\}$ is the set of nodes, and $E \subseteq V_w \times V_w$ is the set of directed arcs. Each node in V_w encodes the position of a token in w, and each arc in E encodes a dependency relation between two tokens. We write $i \to j$ to denote a directed arc (i, j), which will also be called a **dependency link** from i to j.[1] We say that i is the **head** of j and, conversely, that j is a syntactic **dependent** of i.

[1]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.

Given a dependency graph $G = (V_w, E)$, we write $i \to^{\star} j \in E$ if there is a (possibly empty) directed path from i to j; and $i \leftrightarrow^{\star} j \in E$ if there is a (possibly empty) path between i and j in the undirected graph underlying G (omitting the references to E when clear from the context).

Most dependency-based representations of syntax do not allow arbitrary dependency graphs, instead, they are restricted to acyclic graphs that have at most one head per node. Dependency graphs satisfying these constraints are called dependency forests.

Definition 1 *A dependency graph G is said to be a* ***forest*** *iff it satisfies:*

1. *Acyclicity constraint: if $i \to^{\star} j$, then not $j \to i$.*
2. *Single-head constraint: if $j \to i$, then there is no $k \neq j$ such that $k \to i$.*

A node that has no head in a dependency forest is called a **root**. Some dependency frameworks add the additional constraint that dependency forests have only one root (or, equivalently, that they are connected). Such a forest is called a **dependency tree**. A dependency tree can be obtained from any dependency forest by linking all of its root nodes as dependents of a dummy root node, conventionally located in position 0 of the input.

2.2 Transition Systems

In the framework of Nivre (2008), transition-based parsers are described by means of a non-deterministic state machine called a **transition system**.

Definition 2 *A* ***transition system*** *for dependency parsing is a tuple $S = (C, T, c_s, C_t)$, where*

1. *C is a set of possible parser* ***configurations****,*
2. *T is a finite set of* ***transitions****, which are partial functions $t : C \to C$,*
3. *c_s is a total initialization function mapping each input string to a unique* ***initial configuration****, and*
4. *$C_t \subseteq C$ is a set of* ***terminal configurations****.*

To obtain a deterministic parser from a non-deterministic transition system, an **oracle** is used to deterministically select a single transition at each configuration. An oracle for a transition system $S = (C, T, c_s, C_t)$ is a function $o : C \to T$. Suitable oracles can be obtained in practice by training classifiers on treebank data (Nivre et al., 2004).

2.3 The Planar, 2-Planar and Covington Transition Systems

Our undirected dependency parsers are based on the planar and 2-planar transition systems by Gómez-Rodríguez and Nivre (2010) and the version of the Covington (2001) non-projective parser defined by Nivre (2008). We now outline these directed parsers briefly, a more detailed description can be found in the above references.

2.3.1 Planar

The planar transition system by Gómez-Rodríguez and Nivre (2010) is a linear-time transition-based parser for **planar** dependency forests, i.e., forests whose dependency arcs do not cross when drawn above the words. The set of planar dependency structures is a very mild extension of that of projective structures (Kuhlmann and Nivre, 2006).

Configurations in this system are of the form $c = \langle \Sigma, B, A \rangle$ where Σ and B are disjoint lists of nodes from V_w (for some input w), and A is a set of dependency links over V_w. The list B, called the **buffer**, holds the input words that are still to be read. The list Σ, called the **stack**, is initially empty and is used to hold words that have dependency links pending to be created. The system is shown at the top in Figure 2, where the notation $\Sigma \mid i$ is used for a stack with top i and tail Σ, and we invert the notation for the buffer for clarity (i.e., $i \mid B$ as a buffer with top i and tail B).

The system reads the input sentence and creates links in a left-to-right order by executing its four transitions, until it gets to a terminal configuration. A SHIFT transition moves the first (leftmost) node in the buffer to the top of the stack. Transitions LEFT-ARC and RIGHT-ARC create leftward or rightward link, respectively, involving the first node in the buffer and the topmost node in the stack. Finally, REDUCE transition is used to pop the top word from the stack when we have finished building arcs to or from it.

2.3.2 2-Planar

The 2-planar transition system by Gómez-Rodríguez and Nivre (2010) is an extension of the planar system that uses two stacks, allowing it to recognize 2-planar structures, a larger set of dependency structures that has been shown to cover the vast majority of non-projective structures in a number of treebanks (Gómez-Rodríguez and Nivre, 2010).

This transition system, shown in Figure 2, has configurations of the form $c = \langle \Sigma_0, \Sigma_1, B, A \rangle$, where we call Σ_0 the **active stack** and Σ_1 the **inactive stack**. Its SHIFT, LEFT-ARC, RIGHT-ARC and REDUCE transitions work similarly to those in the planar parser, but while SHIFT pushes the first word in the buffer to *both* stacks; the other three transitions only work with the top of the active stack, ignoring the inactive one. Finally, a SWITCH transition is added that makes the active stack inactive and vice versa.

2.3.3 Covington Non-Projective

Covington (2001) proposes several incremental parsing strategies for dependency representations and one of them can recover non-projective dependency graphs. Nivre (2008) implements a variant of this strategy as a transition system with configurations of the form $c = \langle \lambda_1, \lambda_2, B, A \rangle$, where λ_1 and λ_2 are **lists** containing partially processed words and B is the **buffer** list of unprocessed words.

The Covington non-projective transition system is shown at the bottom in Figure 2. At each configuration $c = \langle \lambda_1, \lambda_2, B, A \rangle$, the parser has to consider whether any dependency arc should be created involving the top of the buffer and the words in λ_1. A LEFT-ARC transition adds a link from the first node j in the buffer to the node in the head of the list λ_1, which is moved to the list λ_2 to signify that we have finished considering it as a possible head or dependent of j. The RIGHT-ARC transition does the same manipulation, but creating the symmetric link. A NO-ARC transition removes the head of the list λ_1 and inserts it at the head of the list λ_2 without creating any arcs: this transition is to be used where there is no dependency relation between the top node in the buffer and the head of λ_1, but we still may want to create an arc involving the top of the buffer and other nodes in λ_1. Finally, if we do not want to create any such arcs at all, we can execute a SHIFT transition, which advances the parsing process by removing the first node in the buffer B and inserting it at the head of a list obtained by concatenating

λ_1 and λ_2. This list becomes the new λ_1, whereas λ_2 is empty in the resulting configuration.

Note that the Covington parser has quadratic complexity with respect to input length, while the planar and 2-planar parsers run in linear time.

3 The Undirected Parsers

The transition systems defined in Section 2.3 share the common property that their LEFT-ARC and RIGHT-ARC have exactly the same effects except for the direction of the links that they create. We can take advantage of this property to define undirected versions of these transition systems, by transforming them as follows:

- Configurations are changed so that the arc set A is a set of undirected arcs, instead of directed arcs.
- The LEFT-ARC and RIGHT-ARC transitions in each parser are collapsed into a single ARC transition that creates an undirected arc.
- The preconditions of transitions that guarantee the single-head constraint are removed, since the notions of head and dependent are lost in undirected graphs.

By performing these transformations and leaving the systems otherwise unchanged, we obtain the undirected variants of the planar, 2-planar and Covington algorithms that are shown in Figure 3.

Note that the transformation can be applied to any transition system having LEFT-ARC and RIGHT-ARC transitions that are equal except for the direction of the created link, and thus collapsable into one. The above three transition systems fulfill this property, but not every transition system does. For example, the well-known arc-eager parser of Nivre (2003) pops a node from the stack when creating left arcs, and pushes a node to the stack when creating right arcs, so the transformation cannot be applied to it.[2]

3.1 Feature models

Some of the features that are typically used to train transition-based dependency parsers depend on the direction of the arcs that have been built up to a certain point. For example, two such features for the planar parser could be the POS tag associated with the head of the topmost stack node, or the label of the arc going from the first node in the buffer to its leftmost dependent.[3]

As the notion of head and dependent is lost in undirected graphs, this kind of features cannot be used to train undirected parsers. Instead, we use features based on undirected relations between nodes. We found that the following kinds of features worked well in practice as a replacement for features depending on arc direction:

- Information about the ith node linked to a given node (topmost stack node, topmost buffer node, etc.) on the left or on the right, and about the associated undirected arc, typically for $i = 1, 2, 3$,
- Information about whether two nodes are linked or not in the undirected graph, and about the label of the arc between them,
- Information about the first left and right "undirected siblings" of a given node, i.e., the first node q located to the left of the given node p such that p and q are linked to some common node r located to the right of both, and vice versa. Note that this notion of undirected siblings does not correspond exclusively to siblings in the directed graph, since it can also capture other second-order interactions, such as grandparents.

4 Reconstructing the dependency forest

The modified transition systems presented in the previous section generate undirected graphs. To obtain complete dependency parsers that are able to produce directed dependency forests, we will need a reconstruction step that will assign a direction to the arcs in such a way that the single-head constraint is obeyed. This reconstruction step can be implemented by building a directed graph with weighted arcs corresponding to both possible directions of each undirected edge, and then finding an optimum branching to reduce it to a directed

[2]One might think that the arc-eager algorithm could still be transformed by converting each of its arc transitions into an undirected transition, without collapsing them into one. However, this would result into a parser that violates the acyclicity constraint, since the algorithm is designed in such a way that acyclicity is only guaranteed if the single-head constraint is kept. It is easy to see that this problem cannot happen in parsers where LEFT-ARC and RIGHT-ARC transitions have the same effect: in these, if a directed graph is not parsable in the original algorithm, its underlying undirected graph cannot not be parsable in the undirected variant.

[3]These example features are taken from the default model for the planar parser in version 1.5 of MaltParser (Nivre et al., 2006).

Planar initial/terminal configurations:		$c_s(w_1 \dots w_n) = \langle [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \Sigma, [], A \rangle \in C\}$
Transitions:	SHIFT	$\langle \Sigma, i\|B, A \rangle \Rightarrow \langle \Sigma\|i, B, A \rangle$
	REDUCE	$\langle \Sigma\|i, B, A \rangle \Rightarrow \langle \Sigma, B, A \rangle$
	LEFT-ARC	$\langle \Sigma\|i, j\|B, A \rangle \Rightarrow \langle \Sigma\|i, j\|B, A \cup \{(j,i)\} \rangle$ only if $\nexists k \mid (k,i) \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).
	RIGHT-ARC	$\langle \Sigma\|i, j\|B, A \rangle \Rightarrow \langle \Sigma\|i, j\|B, A \cup \{(i,j)\} \rangle$ only if $\nexists k \mid (k,j) \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).
2-Planar initial/terminal configurations:		$c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \Sigma_0, \Sigma_1, [], A \rangle \in C\}$
Transitions:	SHIFT	$\langle \Sigma_0, \Sigma_1, i\|B, A \rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1\|i, B, A \rangle$
	REDUCE	$\langle \Sigma_0\|i, \Sigma_1, B, A \rangle \Rightarrow \langle \Sigma_0, \Sigma_1, B, A \rangle$
	LEFT-ARC	$\langle \Sigma_0\|i, \Sigma_1, j\|B, A \rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1, j\|B, A \cup \{j,i)\} \rangle$ only if $\nexists k \mid (k,i) \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).
	RIGHT-ARC	$\langle \Sigma_0\|i, \Sigma_1, j\|B, A \rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1, j\|B, A \cup \{(i,j)\} \rangle$ only if $\nexists k \mid (k,j) \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).
	SWITCH	$\langle \Sigma_0, \Sigma_1, B, A \rangle \Rightarrow \langle \Sigma_1, \Sigma_0, B, A \rangle$
Covington initial/term. configurations:		$c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \lambda_1, \lambda_2, [], A \rangle \in C\}$
Transitions:	SHIFT	$\langle \lambda_1, \lambda_2, i\|B, A \rangle \Rightarrow \langle \lambda_1 \cdot \lambda_2\|i, [], B, A \rangle$
	NO-ARC	$\langle \lambda_1\|i, \lambda_2, B, A \rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, B, A \rangle$
	LEFT-ARC	$\langle \lambda_1\|i, \lambda_2, j\|B, A \rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{(j,i)\} \rangle$ only if $\nexists k \mid (k,i) \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).
	RIGHT-ARC	$\langle \lambda_1\|i, \lambda_2, j\|B, A \rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{(i,j)\} \rangle$ only if $\nexists k \mid (k,j) \in A$ (single-head) and $i \leftrightarrow^* j \notin A$ (acyclicity).

Figure 2: Transition systems for planar, 2-planar and Covington non-projective dependency parsing.

Undirected Planar initial/term. conf.:		$c_s(w_1 \dots w_n) = \langle [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \Sigma, [], A \rangle \in C\}$
Transitions:	SHIFT	$\langle \Sigma, i\|B, A \rangle \Rightarrow \langle \Sigma\|i, B, A \rangle$
	REDUCE	$\langle \Sigma\|i, B, A \rangle \Rightarrow \langle \Sigma, B, A \rangle$
	ARC	$\langle \Sigma\|i, j\|B, A \rangle \Rightarrow \langle \Sigma\|i, j\|B, A \cup \{\{i,j\}\} \rangle$ only if $i \leftrightarrow^* j \notin A$ (acyclicity).
Undirected 2-Planar initial/term. conf.:		$c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \Sigma_0, \Sigma_1, [], A \rangle \in C\}$
Transitions:	SHIFT	$\langle \Sigma_0, \Sigma_1, i\|B, A \rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1\|i, B, A \rangle$
	REDUCE	$\langle \Sigma_0\|i, \Sigma_1, B, A \rangle \Rightarrow \langle \Sigma_0, \Sigma_1, B, A \rangle$
	ARC	$\langle \Sigma_0\|i, \Sigma_1, j\|B, A \rangle \Rightarrow \langle \Sigma_0\|i, \Sigma_1, j\|B, A \cup \{\{i,j\}\} \rangle$ only if $i \leftrightarrow^* j \notin A$ (acyclicity).
	SWITCH	$\langle \Sigma_0, \Sigma_1, B, A \rangle \Rightarrow \langle \Sigma_1, \Sigma_0, B, A \rangle$
Undirected Covington init./term. conf.:		$c_s(w_1 \dots w_n) = \langle [], [], [1 \dots n], \emptyset \rangle$, $C_f = \{\langle \lambda_1, \lambda_2, [], A \rangle \in C\}$
Transitions:	SHIFT	$\langle \lambda_1, \lambda_2, i\|B, A \rangle \Rightarrow \langle \lambda_1 \cdot \lambda_2\|i, [], B, A \rangle$
	NO-ARC	$\langle \lambda_1\|i, \lambda_2, B, A \rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, B, A \rangle$
	ARC	$\langle \lambda_1\|i, \lambda_2, j\|B, A \rangle \Rightarrow \langle \lambda_1, i\|\lambda_2, j\|B, A \cup \{\{i,j\}\} \rangle$ only if $i \leftrightarrow^* j \notin A$ (acyclicity).

Figure 3: Transition systems for undirected planar, 2-planar and Covington non-projective dependency parsing.

tree. Different criteria for assigning weights to arcs provide different variants of the reconstruction technique.

To describe these variants, we first introduce preliminary definitions. Let $U = (V_w, E)$ be an undirected graph produced by an undirected parser for some string w. We define the following sets of arcs:

$$A_1(U) = \{(i,j) \mid j \neq 0 \wedge \{i,j\} \in E\},$$
$$A_2(U) = \{(0,i) \mid i \in V_w\}.$$

Note that $A_1(U)$ represents the set of arcs obtained from assigning an orientation to an edge in U, except arcs whose dependent is the dummy root, which are disallowed. On the other hand, $A_2(U)$ contains all the possible arcs originating from the dummy root node, regardless of whether their underlying undirected edges are in U or not; this is so that reconstructions are allowed to link unattached tokens to the dummy root.

The reconstruction process consists of finding a minimum branching (i.e. a directed minimum spanning tree) for a weighted directed graph obtained from assigning a cost $c(i,j)$ to each arc (i,j) of the following directed graph:

$$D(U) = \{V_w, A(U) = A_1(U) \cup A_2(U)\}.$$

That is, we will find a dependency tree $T = (V_w, A_T \subseteq A(U))$ such that the sum of costs of the arcs in A_T is minimal. In general, such a minimum branching can be calculated with the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). Since the graph $D(U)$ has $O(n)$ nodes and $O(n)$ arcs for a string of length n, this can be done in $O(n \log n)$ if implemented as described by Tarjan (1977).

However, applying these generic techniques is not necessary in this case: since our graph U is acyclic, the problem of reconstructing the forest can be reduced to choosing a root word for each connected component in the graph, linking it as a dependent of the dummy root and directing the other arcs in the component in the (unique) way that makes them point away from the root.

It remains to see how to assign the costs $c(i,j)$ to the arcs of $D(U)$: different criteria for assigning scores will lead to different reconstructions.

4.1 Naive reconstruction

A first, very simple reconstruction technique can be obtained by assigning arc costs to the arcs in $A(U)$ as follows:

$$c(i,j) \begin{cases} 1 & \text{if } (i,j) \in A_1(U), \\ 2 & \text{if } (i,j) \in A_2(U) \wedge (i,j) \notin A_1(U). \end{cases}$$

This approach gives the same cost to all arcs obtained from the undirected graph U, while also allowing (at a higher cost) to attach any node to the dummy root. To obtain satisfactory results with this technique, we must train our parser to explicitly build undirected arcs from the dummy root node to the root word(s) of each sentence using arc transitions (note that this implies that we need to represent forests as trees, in the manner described at the end of Section 2.1). Under this assumption, it is easy to see that we can obtain the correct directed tree T for a sentence if it is provided with its underlying undirected tree U: the tree is obtained in $O(n)$ as the unique orientation of U that makes each of its edges point away from the dummy root.

This approach to reconstruction has the advantage of being very simple and not adding any complications to the parsing process, while guaranteeing that the correct directed tree will be recovered if the undirected tree for a sentence is generated correctly. However, it is not very robust, since the direction of all the arcs in the output depends on which node is chosen as sentence head and linked to the dummy root. Therefore, a parsing error affecting the undirected edge involving the dummy root may result in many dependency links being erroneous.

4.2 Label-based reconstruction

To achieve a more robust reconstruction, we use labels to encode a preferred direction for dependency arcs. To do so, for each pre-existing label X in the training set, we create two labels X_l and X_r. The parser is then trained on a modified version of the training set where leftward links originally labelled X are labelled X_l, and rightward links originally labelled X are labelled X_r. Thus, the output of the parser on a new sentence will be an undirected graph where each edge has a label with an annotation indicating whether the reconstruction process should prefer to link the pair of nodes with a leftward or a rightward arc. We can then assign costs to our minimum branching algorithm so that it will return a tree agreeing with as many such annotations as possible.

To do this, we call $A_{1+}(U) \subseteq A_1(U)$ the set of arcs in $A_1(U)$ that agree with the annotations, i.e., arcs $(i,j) \in A_1(U)$ where either $i < j$ and i,j is labelled X_r in U, or $i > j$ and i,j is labelled X_l in U. We call $A_{1-}(U)$ the set of arcs in $A_1(U)$ that disagree with the annotations, i.e., $A_{1-}(U) = A_1(U) \backslash A_{1+}(U)$. And we assign costs as follows:

$$c(i,j) \begin{cases} 1 & \text{if } (i,j) \in A_{1+}(U), \\ 2 & \text{if } (i,j) \in A_{1-}(U), \\ 2n & \text{if } (i,j) \in A_2(U) \wedge (i,j) \notin A_1(U). \end{cases}$$

where n is the length of the string.

With these costs, the minimum branching algorithm will find a tree which agrees with as many annotations as possible. Additional arcs from the root not corresponding to any edge in the output of the parser (i.e. arcs in $A_2(U)$ but not in $A_1(U)$) will be used only if strictly necessary to guarantee connectedness, this is implemented by the high cost for these arcs.

While this may be the simplest cost assignment to implement label-based reconstruction, we have found that better empirical results are obtained if we give the algorithm more freedom to create new arcs from the root, as follows:

$$c(i,j) \begin{cases} 1 & \text{if } (i,j) \in A_{1+}(U) \wedge (i,j) \notin A_2(U), \\ 2 & \text{if } (i,j) \in A_{1-}(U) \wedge (i,j) \notin A_2(U), \\ 2n & \text{if } (i,j) \in A_2(U). \end{cases}$$

While the cost of arcs from the dummy root is still $2n$, this is now so even for arcs that are in the output of the undirected parser, which had cost 1 before. Informally, this means that with this configuration the postprocessor does not "trust" the links from the dummy root created by the parser, and may choose to change them if it is convenient to get a better agreement with label annotations (see Figure 4 for an example of the difference between both cost assignments). We believe that the better accuracy obtained with this criterion probably stems from the fact that it is biased towards changing links from the root, which tend to be more problematic for transition-based parsers, while respecting the parser output for links located deeper in the dependency structure, for which transition-based parsers tend to be more accurate (McDonald and Nivre, 2007).

Note that both variants of label-based reconstruction have the property that, if the undirected parser produces the correct edges and labels for a given sentence, then the obtained directed tree is guaranteed to be correct (as it will simply be the tree obtained by decoding the label annotations).

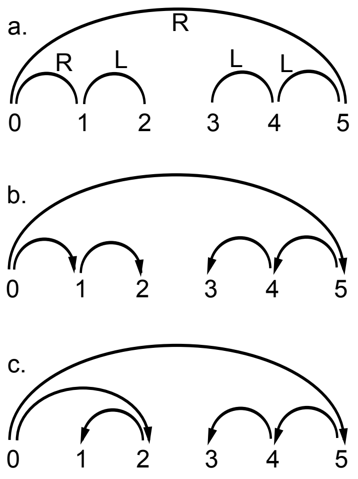

Figure 4: a) An undirected graph obtained by the parser with the label-based transformation, b) and c) The dependency graph obtained by each of the variants of the label-based reconstruction (note how the second variant moves an arc from the root).

5 Experiments

In this section, we evaluate the performance of the undirected planar, 2-planar and Covington parsers on eight datasets from the CoNLL-X shared task (Buchholz and Marsi, 2006).

Tables 1, 2 and 3 compare the accuracy of the undirected versions with naive and label-based reconstruction to that of the directed versions of the planar, 2-planar and Covington parsers, respectively. In addition, we provide a comparison to well-known state-of-the-art projective and non-projective parsers: the planar parsers are compared to the arc-eager projective parser by Nivre (2003), which is also restricted to planar structures; and the 2-planar parsers are compared with the arc-eager parser with pseudo-projective transformation of Nivre and Nilsson (2005), capable of handling non-planar dependencies.

We use SVM classifiers from the LIBSVM package (Chang and Lin, 2001) for all the languages except Chinese, Czech and German. In these, we use the LIBLINEAR package (Fan et al., 2008) for classification, which reduces training time for these larger datasets; and feature models adapted to this system which, in the case of German, result in higher accuracy than published results using LIBSVM.

The LIBSVM feature models for the arc-eager projective and pseudo-projective parsers are the same used by these parsers in the CoNLL-X shared task, where the pseudo-projective version of MaltParser was one of the two top performing systems (Buchholz and Marsi, 2006). For the 2-planar parser, we took the feature models from Gómez-Rodríguez and Nivre (2010) for the languages included in that paper. For all the algorithms and datasets, the feature models used for the undirected parsers were adapted from those of the directed parsers as described in Section 3.1.[4]

The results show that the use of undirected parsing with label-based reconstruction clearly improves the performance in the vast majority of the datasets for the planar and Covington algorithms, where in many cases it also improves upon the corresponding projective and non-projective state-of-the-art parsers provided for comparison. In the case of the 2-planar parser the results are less conclusive, with improvements over the directed versions in five out of the eight languages.

The improvements in LAS obtained with label-based reconstruction over directed parsing are statistically significant at the .05 level[5] for Danish, German and Portuguese in the case of the planar parser; and Czech, Danish and Turkish for Covington's parser. No statistically significant decrease in accuracy was detected in any of the algorithm/dataset combinations.

As expected, the good results obtained by the undirected parsers with label-based reconstruction contrast with those obtained by the variants with root-based reconstruction, which performed worse in all the experiments.

6 Discussion

We have presented novel variants of the planar and 2-planar transition-based parsers by Gómez-Rodríguez and Nivre (2010) and of Covington's non-projective parser (Covington, 2001; Nivre, 2008) which ignore the direction of dependency links, and reconstruction techniques that can be used to recover the direction of the arcs thus produced. The results obtained show that this idea of undirected parsing, together with the label-based reconstruction technique of Section 4.2, improves parsing accuracy on most of the tested dataset/algorithm combinations, and it can outperform state-of-the-art transition-based parsers.

The accuracy improvements achieved by relaxing the single-head constraint to mitigate error propagation were able to overcome the errors generated in the reconstruction phase, which were few: we observed empirically that the differences between the undirected LAS obtained from the undirected graph before the reconstruction and the final directed LAS are typically below 0.20%. This is true both for the naive and label-based transformations, indicating that both techniques are able to recover arc directions accurately, and the accuracy differences between them come mainly from the differences in training (e.g. having tentative arc direction as part of feature information in the label-based reconstruction and not in the naive one) rather than from the differences in the reconstruction methods themselves.

The reason why we can apply the undirected simplification to the three parsers that we have used in this paper is that their LEFT-ARC and RIGHT-ARC transitions have the same effect except for the direction of the links they create. The same transformation and reconstruction techniques could be applied to any other transition-based dependency parsers sharing this property. The reconstruction techniques alone could potentially be applied to any dependency parser (transition-based or not) as long as it can be somehow converted to output undirected graphs.

The idea of parsing with undirected relations between words has been applied before in the work on Link Grammar (Sleator and Temperley, 1991), but in that case the formalism itself works with undirected graphs, which are the final output of the parser. To our knowledge, the idea of using an undirected graph as an intermediate step towards obtaining a dependency structure has not been explored before.

Acknowledgments

This research has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER (projects TIN2010-18552-C03-01 and TIN2010-18552-C03-02), Ministry of Education (FPU Grant Program) and Xunta de Galicia (Rede Galega de Recursos Lingüísticos para unha Soc. do Coñec.). The experiments were conducted with the help of computing resources provided by the Supercomputing Center of Galicia (CESGA). We thank Joakim Nivre for helpful input in the early stages of this work.

[4]All the experimental settings and feature models used are included in the supplementary material and also available at `http://www.grupolys.org/~cgomezr/exp/`.

[5]Statistical significance was assessed using Dan Bikel's randomized comparator: `http://www.cis.upenn.edu/~dbikel/software.html`

	Planar		UPlanarN		UPlanarL		MaltP	
Lang.	LAS(p)	UAS(p)	LAS(p)	UAS(p)	LAS(p)	UAS(p)	LAS(p)	UAS(p)
Arabic	**66.93** (**67.34**)	**77.56** (**77.22**)	65.91 (66.33)	77.03 (76.75)	66.75 (67.19)	77.45 (**77.22**)	66.43 (66.74)	77.19 (76.83)
Chinese	84.23 (84.20)	88.37 (88.33)	83.14 (83.10)	87.00 (86.95)	84.51* (84.50*)	88.37 (88.35*)	**86.42** (**86.39**)	**90.06** (**90.02**)
Czech	77.24 (77.70)	83.46 (83.24)	75.08 (75.60)	81.14 (81.14)	**77.60*** (**77.93***)	**83.56*** (**83.41***)	77.24 (77.57)	83.40 (83.19)
Danish	83.31 (82.60)	88.02 (86.64)	82.65 (82.45)	87.58 (86.67*)	**83.87*** (**83.83***)	**88.94*** (**88.17***)	83.31 (82.64)	88.30 (86.91)
German	84.66 (83.60)	87.02 (85.67)	83.33 (82.77)	85.78 (84.93)	**86.32*** (**85.67***)	**88.62*** (**87.69***)	86.12 (85.48)	88.52 (87.58)
Portug.	86.22 (83.82)	89.80 (86.88)	85.89 (83.82)	89.68 (87.06*)	86.52* (**84.83***)	**90.28*** (**88.03***)	**86.60** (84.66)	90.20 (87.73)
Swedish	**83.01** (82.44)	88.53 (87.36)	81.20 (81.10)	86.50 (85.86)	82.95 (**82.66***)	88.29 (87.45*)	82.89 (82.44)	**88.61** (**87.55**)
Turkish	62.70 (71.27)	73.67 (78.57)	59.83 (68.31)	70.15 (75.17)	**63.27*** (**71.63***)	**73.93*** (**78.72***)	62.58 (70.96)	73.09 (77.95)

Table 1: Parsing accuracy of the undirected planar parser with naive (UPlanarN) and label-based (UPlanarL) postprocessing in comparison to the directed planar (Planar) and the MaltParser arc-eager projective (MaltP) algorithms, on eight datasets from the CoNLL-X shared task (Buchholz and Marsi, 2006): Arabic (Hajič et al., 2004), Chinese (Chen et al., 2003), Czech (Hajič et al., 2006), Danish (Kromann, 2003), 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). We show labelled (LAS) and unlabelled (UAS) attachment score excluding and including punctuation tokens in the scoring (the latter in brackets). Best results for each language are shown in boldface, and results where the undirected parser outperforms the directed version are marked with an asterisk.

	2Planar		U2PlanarN		U2PlanarL		MaltPP	
Lang.	LAS(p)	UAS(p)	LAS(p)	UAS(p)	LAS(p)	UAS(p)	LAS(p)	UAS(p)
Arabic	**66.73** (**67.19**)	**77.33** (**77.11**)	66.37 (66.93)	77.15 (77.09)	66.13 (66.52)	76.97 (76.70)	65.93 (66.02)	76.79 (76.14)
Chinese	84.35 (84.32)	88.31 (88.27)	83.02 (82.98)	86.86 (86.81)	84.45* (84.42*)	88.29 (88.25)	86.42 (86.39)	90.06 (90.02)
Czech	77.72 (77.91)	83.76 (83.32)	74.44 (75.19)	80.68 (80.80)	78.00* (**78.59***)	84.22* (**84.21***)	**78.86** (78.47)	**84.54** (83.89)
Danish	**83.81** (83.61)	88.50 (87.63)	82.00 (81.63)	86.87 (85.80)	83.75 (**83.65***)	**88.62*** (**87.82***)	83.67 (83.54)	88.52 (87.70)
German	86.28 (85.76)	88.68 (87.86)	82.93 (82.53)	85.52 (84.81)	86.52* (85.99*)	88.72* (87.92*)	**86.94** (**86.62**)	**89.30** (**88.69**)
Portug.	87.04 (**84.92**)	**90.82** (**88.14**)	85.61 (83.45)	89.36 (86.65)	86.70 (84.75)	90.38 (87.88)	**87.08** (84.90)	90.66 (87.95)
Swedish	83.13 (**82.71**)	88.57 (**87.59**)	81.00 (80.71)	86.54 (85.68)	82.59 (82.25)	88.19 (87.29)	**83.39** (82.67)	**88.59** (87.38)
Turkish	61.80 (70.09)	72.75 (77.39)	58.10 (67.44)	68.03 (74.06)	61.92* (70.64*)	72.18 (77.46*)	**62.80** (**71.33**)	**73.49** (**78.44**)

Table 2: Parsing accuracy of the undirected 2-planar parser with naive (U2PlanarN) and label-based (U2PlanarL) postprocessing in comparison to the directed 2-planar (2Planar) and MaltParser arc-eager pseudo-projective (MaltPP) algorithms. The meaning of the scores shown is as in Table 1.

	Covington		UCovingtonN		UCovingtonL	
Lang.	LAS(p)	UAS(p)	LAS(p)	UAS(p)	LAS(p)	UAS(p)
Arabic	65.17 (65.49)	75.99 (**75.69**)	63.49 (63.93)	74.41 (74.20)	**65.61*** (**65.81***)	**76.11*** (75.66)
Chinese	85.61 (85.61)	89.64 (89.62)	84.12 (84.02)	87.85 (87.73)	**86.28*** (**86.17***)	**90.16*** (**90.04***)
Czech	78.26 (77.43)	84.04 (83.15)	74.02 (74.78)	79.80 (79.92)	**78.42*** (**78.69***)	**84.50*** (**84.16***)
Danish	83.63 (82.89)	88.50 (87.06)	82.00 (81.61)	86.55 (85.51)	**84.27*** (**83.85***)	**88.82*** (**87.75***)
German	**86.70** (85.69)	**89.08** (87.78)	84.03 (83.51)	86.16 (85.39)	86.50 (**85.90***)	88.84 (**87.95***)
Portug.	84.73 (82.56)	89.10 (86.30)	83.83 (81.71)	87.88 (85.17)	**84.95*** (**82.70***)	**89.18*** (**86.31***)
Swedish	**83.53** (**82.76**)	**88.91** (**87.61**)	81.78 (81.47)	86.78 (85.96)	83.09 (82.73)	88.11 (87.23)
Turkish	64.25 (72.70)	74.85 (79.75)	63.51 (72.08)	74.07 (79.10)	**64.91*** (**73.38***)	**75.46*** (**80.40***)

Table 3: Parsing accuracy of the undirected Covington non-projective parser with naive (UCovingtonN) and label-based (UCovingtonL) postprocessing in comparison to the directed algorithm (Covington). The meaning of the scores shown is as in Table 1.

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.

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.

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.

K. Chen, C. Luo, M. Chang, F. Chen, C. Chen, C. Huang, and Z. Gao. 2003. Sinica treebank: Design criteria, representational issues and implementation. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, chapter 13, pages 231–248. Kluwer.

Y. J. Chu and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1492–1501, Stroudsburg, PA, USA. Association for Computational Linguistics.

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*.

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.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1077–1086, Stroudsburg, PA, USA. Association for Computational Linguistics.

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

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 Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 523–530.

Jens Nilsson, Johan Hall, and Joakim Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from Antiquity. In Peter Juel Henrichsen, editor, *Proceedings of the NODALIDA Special Session on Treebanks*.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

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. 2006. 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. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

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 Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, pages 261–277. Kluwer.

Daniel Sleator and Davy Temperley. 1991. Parsing English with a link grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University, Computer Science.

R. E. Tarjan. 1977. Finding optimum branchings. *Networks*, 7:25–35.

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.