Sorting Out Dependency Parsing

Joakim Nivre

Uppsala University, Department of Linguistics and Philology
Växjö University, School of Mathematics and Systems Engineering
joakim.nivre@lingfil.uu.se
http://stp.lingfil.uu.se/~nivre/

Abstract. This paper explores the idea that non-projective dependency parsing can be conceived as the outcome of two interleaved processes, one that sorts the words of a sentence into a canonical order, and one that performs strictly projective dependency parsing on the sorted input. Based on this idea, a parsing algorithm is constructed by combining an online sorting algorithm with an arc-standard transition system for projective dependency parsing.

Keywords: parsing, sorting, non-projective dependency parsing.

1 Introduction

In syntactic parsing of natural language, we analyze sentences by constructing representations of their syntactic structure. Many different representations have been proposed for this purpose, but in this paper we will restrict our attention to *dependency graphs*. This form of representation, which comes out of a long tradition of theoretical work in dependency grammar [1,2,3,4], has recently enjoyed widespread interest in the computational linguistics community and have been used for applications as diverse as information extraction [5], machine translation [6], textual entailment [7], lexical ontology induction [8], and question answering [9]. We attribute this increase in interest to the fact that dependency graphs provide a transparent encoding of predicate-argument structure, which is useful for certain types of applications, together with the fact that they can be processed both efficiently and accurately, in particular using data-driven models that are induced from syntactically annotated corpora. Such models have recently been applied to a wide range of languages in connection with the CoNLL shared tasks on dependency parsing in 2006 and 2007 [10,11].

The dependency graph for a sentence is usually taken to be a directed tree, with nodes corresponding to the words of the sentence and with labeled arcs representing syntactic relations between words. For simplicity, it is often assumed that the single root of this tree is an artificial word ROOT prefixed to the sentence, as illustrated in Figure 1. One issue that is often debated is whether dependency graphs should also be assumed to be *projective*, that is, whether the yield of every subtree should be a continuous substring of the sentence. The dependency graph in Figure 1 fails to satisfy this condition, because the subtrees rooted at

A. Ranta (Ed.): GoTAL 2008, LNAI 5221, pp. 16–27, 2008.
© Springer-Verlag Berlin Heidelberg 2008

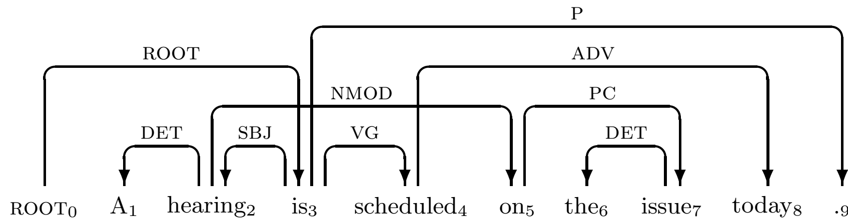

Fig. 1. Dependency graph for an English sentence (non-projective)

the words *hearing* and *scheduled* both have discontinuous yields (words 1, 2, 5, 6, 7 in the first case, words 4, 8 in the second). Most researchers today assume that, although projectivity is appealing from a computational point of view, it is too restrictive from a linguistic representational point of view, and most frameworks therefore allow non-projective dependency graphs for representing discontinuous linguistic constructions. This raises the question of how to parse such representations accurately and efficiently, given that most parsing algorithms proposed for natural language are limited to the derivation of continuous structures.

Current approaches to non-projective dependency parsing typically take one of two routes. Either they employ a non-standard parsing algorithm that is not limited to the derivation of continuous substructures, or they try to recover non-projective dependencies by post-processing the output of a strictly projective parser. The most well-known example of the former approach is the application of the Chu-Liu-Edmonds maximum spanning tree algorithm for directed graphs to dependency parsing [12], although other algorithms also exist [13,14]. The second approach is exemplified by pseudo-projective parsing [15], corrective modeling [16], and approximate second-order spanning tree parsing [17]. In this paper, we start exploring a third route, based on the idea that the parsing problem for dependency graphs can be decomposed into a *sorting problem*, where the input words need to be sorted into a canonical order, and a simpler *parsing problem*, where the ordered input is mapped to a strictly projective dependency graph.

The rest of the paper is structured as follows. Section 2 reviews the transition-based approach to projective dependency parsing, which is one of our building blocks. Section 3 introduces the idea of sorting the input words to facilitate parsing, defines the canonical sort order in terms of tree traversals, and presents a transition-based sorting algorithm. Section 4 puts the two building blocks together and presents an algorithm that simultaneously sorts the words in the input and constructs a projective dependency graph for the sorted input, a graph that may or may not be non-projective with respect to the original word order. Section 5 concludes and makes suggestions for future research.

2 Projective Dependency Parsing

The transition-based approach to dependency parsing has two key components. The first is a transition system for mapping sentences to dependency graphs;

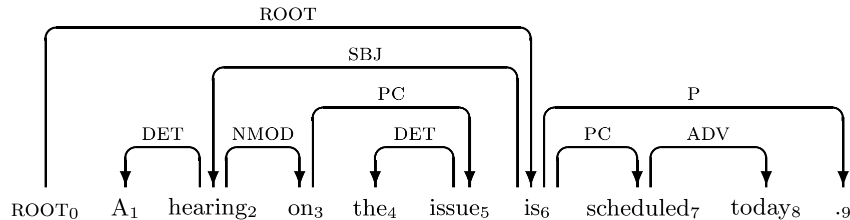

Fig. 2. Dependency graph for an English sentence (projective)

the second is a treebank-induced classifier for predicting the next transition for arbitrary configurations of the parser [18]. We will focus here on the first component and define a transition system that derives strictly projective dependency graphs, using a bottom-up, arc-standard parsing strategy, which is essentially a variant of the system described previously in [19,20,21]. But first of all, we need to define the notion of a dependency graph a little more precisely.

Given a set $L = \{l_1, \dots, l_{|L|}\}$ of dependency labels, a *dependency graph* for a sentence $S = w_0 w_1 \cdots w_n$ (where $w_0 = \text{ROOT}$) is a labeled directed graph $G = (V_S, A)$, where

1. $V_S = \{0, 1, \dots, n\}$ is a set of nodes;
2. $A \subseteq V_S \times L \times V_S$ is a set of labeled directed arcs;

The set V_S of *nodes* (or *vertices*) is the set of non-negative integers up to and including n, each corresponding to the linear position of a word in the sentence (including ROOT). The set A of *arcs* (or *directed edges*) is a set of ordered triples (i, l, j), where i and j are nodes and l is a dependency label. Since arcs are used to represent dependency relations, we will say that i is the *head* and l is the *dependency type* of j. Conversely, we say that j is a *dependent* of i.

For a dependency graph $G = (V_S, A)$ to be well-formed we in addition require that it is a *tree* rooted at the node 0. This implies that there is a unique directed path from the root node to every other node of the graph, and that every node except the root has exactly one incoming arc. By contrast, we do *not* require that G is projective with respect to the sentence S, i.e., that the yield of every subtree of G forms a continuous substring of S (where the yield of a subtree is the set of words corresponding to nodes in the subtree).

As already noted, the dependency graph depicted in Figure 1 is not projective, since the subtrees rooted at nodes 2 and 4 do not have continuous yields. Note, however, that projectivity is not a property of the dependency graph in isolation, but only of the graph in combination with the word order of a sentence. Thus, the dependency graph in Figure 2, while isomorphic to the graph in Figure 1, is projective because the words of the sentence occur in a different order. We will return to this observation in the next section, but first we will concentrate on parsing sentences with strictly projective dependency graphs.

A transition system for dependency parsing consists of a set of *configurations* and *transitions* between configurations. Given a sentence $S = w_0 w_1, \cdots w_n$, we

Transition		**Condition**
LEFT-ARC$_l$	$([\sigma\|w_i, w_j], \beta, A) \Rightarrow ([\sigma\|w_j], \beta, A \cup \{(j, l, i)\})$	$i \neq 0$
RIGHT-ARC$_l$	$([\sigma\|w_i, w_j], \beta, A) \Rightarrow ([\sigma\|w_i], \beta, A \cup \{(i, l, j)\})$	
SHIFT	$(\sigma, [w_i\|\beta], A) \Rightarrow ([\sigma\|w_i], \beta, A)$	

Fig. 3. Transitions for projective dependency parsing

take a *configuration* to be a triple $c = (\sigma, \beta, A)$, where σ is a stack of words $w_i \in S$, β is a buffer of words $w_i \in S$, and A is a set of labeled directed arcs $(i, l, j) \in V_S \times L \times V_S$. When necessary, we use σ_c, β_c and A_c to refer to the different components of a configuration c, and we use G_c to refer to the dependency graph $G = (V_S, A_c)$ defined by c. Both the stack and the buffer will be represented as lists, although the stack will have its head (or top) to the right for reasons of perspicuity. Thus, $[\sigma|w_i]$ represents a stack with top w_i and tail σ, while $[w_j|\beta]$ represents a buffer with head w_j and tail β. We use square brackets for enumerated lists, e.g., $[1, 2, \ldots, n]$, with $[\,]$ for the empty list as a special case.

Given the notion of a parser configuration, we can now define a *transition* to be a (partial) function from configurations to configurations. The following set of transitions, defined more formally in Figure 3, are sufficient for projective dependency parsing:

1. The transition LEFT-ARC$_l$, parameterized for an arc label $l \in L$, updates a parser configuration with words w_i, w_j on top of the stack by adding the arc (j, l, i) to the arc set A and replacing w_i, w_j on the stack by w_j alone. This is a legal transition as long as $w_i \neq \text{ROOT}_0$.
2. The transition RIGHT-ARC$_l$, parameterized for an arc label $l \in L$, updates a parser configuration with words w_i, w_j on top of the stack by adding the arc (i, l, j) to the arc set A and replacing w_i, w_j on the stack by w_i alone.
3. The transition SHIFT updates a parser configuration with the word w_i as the first word of the buffer by removing w_i from the buffer and pushing it onto the stack.

The transition system defined in Figure 3 is *complete* for the set of well-formed projective dependency graphs in the sense that, for any sentence $S = w_0 w_1 \cdots w_n$ with projective dependency graph G, there is a transition sequence $(c_0, c_1, \ldots, c_m)$ such that:

1. $c_0 = ([w_0], [w_1, \ldots, w_n], \emptyset)$
2. $c_{i+1} = t_i(c_i)$ for some transition t_i $(0 \leq i < m)$
3. $G_{c_m} = G$

For example, the dependency graph for the sentence in Figure 2 is derived by the transition sequence given in Figure 4. Ideally, the system should also be *sound*

Transition	Stack (σ)	Buffer (β)	New Arc
	$[\text{R}_0]$	$[\text{A}_1, \ldots, ._9]$	
Shift	$[\text{R}_0, \text{A}_1]$	$[\text{hearing}_2, \ldots, ._9]$	
Shift	$[\text{R}_0, \text{A}_1, \text{hearing}_2]$	$[\text{on}_3, \ldots, ._9]$	
LA_{DET}	$[\text{R}_0, \text{hearing}_2]$	$[\text{on}_3, \ldots, ._9]$	$(2, \text{DET}, 1)$
Shift	$[\text{R}_0, \text{hearing}_2, \text{on}_3]$	$[\text{the}_4, \ldots, ._9]$	
Shift	$[\text{R}_0, \ldots, \text{on}_3, \text{the}_4]$	$[\text{issue}_5, \ldots, ._9]$	
Shift	$[\text{R}_0, \ldots, \text{the}_4, \text{issue}_5]$	$[\text{is}_6, \ldots, ._9]$	
LA_{DET}	$[\text{R}_0, \ldots, \text{on}_3, \text{issue}_5]$	$[\text{is}_6, \ldots, ._9]$	$(5, \text{DET}, 4)$
RA_{PC}	$[\text{R}_0, \text{hearing}_2, \text{on}_3]$	$[\text{is}_6, \ldots, ._9]$	$(3, \text{PC}, 5)$
RA_{NMOD}	$[\text{R}_0, \text{hearing}_2]$	$[\text{is}_6, \ldots, ._9]$	$(2, \text{NMOD}, 3)$
Shift	$[\text{R}_0, \text{hearing}_2, \text{is}_6]$	$[\text{scheduled}_7, \ldots, ._9]$	
LA_{SBJ}	$[\text{R}_0, \text{is}_6]$	$[\text{scheduled}_7, \ldots, ._9]$	$(6, \text{SBJ}, 2)$
Shift	$[\text{R}_0, \text{is}_6, \text{scheduled}_7]$	$[\text{today}_8, ._9]$	
Shift	$[\text{R}_0, \ldots, \text{scheduled}_7, \text{today}_8]$	$[._9]$	
RA_{ADV}	$[\text{R}_0, \text{is}_6, \text{scheduled}_7]$	$[._9]$	$(7, \text{ADV}, 8)$
RA_{VG}	$[\text{R}_0, \text{is}_6]$	$[._9]$	$(6, \text{VG}, 7)$
Shift	$[\text{R}_0, \text{is}_6, ._9]$	$[\,]$	
RA_{P}	$[\text{R}_0, \text{is}_6]$	$[\,]$	$(6, \text{P}, 9)$
RA_{ROOT}	$[\text{R}_0]$	$[\,]$	$(0, \text{ROOT}, 6)$

Fig. 4. Transition sequence for parsing the English sentence in Figure 2

with respect to the set of well-formed projective dependency graphs, in the sense that every transition sequence derives a well-formed graph, which unfortunately is not the case. However, every dependency graph derived by a transition sequence is guaranteed to be a forest (set of trees), which means that it can trivially be converted to a well-formed dependency graph by adding arcs from the node 0 to all (other) root nodes.[1]

We define an *oracle* o to be a function from configurations to transitions such that, for any sentence S with (projective) dependency graph G, if $(c_0, c_1, \ldots, c_m)$ is the transition sequence that derives G for S, then $o(c_i) = t_i$ (for every i such that $0 \leq i < m$). That is, for every configuration c_i, the oracle returns the correct transition t_i out of c_i. Given an oracle, projective dependency parsing can be performed deterministically using the following algorithm:

Parse($S = w_0 w_1 \cdots w_n$)
1 $c \leftarrow ([w_0], [w_1, \ldots, w_n], \emptyset)$
2 **while** $\beta_c \neq [\,]$
3   Shift(c)
4     $t \leftarrow o(c)$
5   **while** $t \in \{\text{Left-Arc}_l, \text{Right-Arc}_l\}$
6         $c \leftarrow t(c)$
7         $t \leftarrow o(c)$
8 **return** G_c

[1] For proofs of soundness and completeness for this transition system, see [20].

The parser is initialized to the configuration $c = ([w_0], [w_1, \ldots, w_n], \emptyset)$, where the stack σ_c contains the artificial root word ROOT, the buffer β_c contains all the real words of the sentence (in their linear order), and the arc set A_c is empty. The outer **while** loop is executed as long as there are words remaining in the buffer and starts by shifting the next word onto the stack after which it calls the oracle. The inner **while** loop is executed as long as the oracle predicts a LEFT-ARC$_l$ or RIGHT-ARC$_l$ transition and simply updates the configuration using the predicted transition and then calls the oracle again. After parsing is completed, the dependency graph G_c defined by the final configuration c is returned.

It is not hard to show that this algorithm terminates after at most $2n$ transitions, as it performs exactly n SHIFT transitions (one for each word initially in the buffer) and can perform at most n other transitions (since both LEFT-ARC$_l$ and RIGHT-ARC$_l$ reduce the size of the stack by 1). This means that, if oracle calls (lines 4 and 7) and transitions (lines 3 and 6) can be computed in constant time, then the time complexity of the parsing algorithm is $O(n)$ [20].

In order to build practical parsing systems, the oracle o has to be approximated by a classifier trained on data derived from a treebank. For every sentence S with dependency graph G, we construct a set of training instances of the form (c_i, t_i), where c_i is a parser configuration and t_i the correct transition out of c_i for the sentence. Training a classifier on such instances can be done using standard machine learning methods for discriminative classification, such as support vector machines or memory-based learning [22,23], and transition-based parsing using treebank-induced classifiers has been shown to give state-of-the-art parsing accuracy in several experimental evaluations [10,11,24]. For the rest of this paper, however, we will ignore the machine learning aspects and concentrate on the construction of a parsing algorithm that is not limited to projective graphs.

3 Sorting to Projective Order

As noted in the preceding section, the projectivity constraint on dependency graphs only holds in relation to a particular word order. And given a sentence $S = w_0 w_1 \cdots w_n$ with (non-projective) dependency graph G, it is always possible to find a permutation S' of S such that G is a projective dependency graph for S'. Moreover, since the graph structure remains the same, all the information about the syntactic structure encoded in G is preserved in this permutation. To take a concrete example, the sentence in Figure 1 can be permuted to the sentence in Figure 2 in order to make the dependency graph projective. In this section, we are going to explore the idea that this kind of permutation can be viewed as a sorting problem, which can be solved using standard sorting algorithms, and that this is a way of extending the transition-based dependency parsing method described in the preceding section to non-projective dependency graphs.

Let $S = w_0 w_1 \cdots w_n$ be a sentence with dependency graph $G = (V_S, A)$. We define the *projective order* of the words in S to be the order in which the corresponding nodes in V_S are visited in an inorder traversal of G starting at the root node 0, where the local order on a node and its children is given by the

Transition		Condition
SWAP	$(m, [\sigma\|w_i, w_j\|\sigma_m], \beta) \Rightarrow (m{+}1, [\sigma\|w_j, w_i\|\sigma_m], \beta)$	$i \neq 0$
SHIFT	$(m, \sigma, [w_i\|\beta]) \Rightarrow (0, [\sigma\|w_i], \beta)$	

Fig. 5. Transitions for sorting into projective order

arithmetic order $<$ on V_S induced by the original word order. The basic idea behind the notion of a projective order is to find a way to impose a linear order on the nodes of the dependency graph in such a way that we guarantee that every subtree has a continuous yield. This can be done in a variety of ways, but because we want to preserve as much as possible of the original word order, we choose an ordering that respects the original ordering of words corresponding to nodes that stand in a parent-child or sibling relation. We can exemplify this by returning to the sentence in Figure 1:

ROOT$_0$ A$_1$ hearing$_2$ is$_3$ scheduled$_4$ on$_5$ the$_6$ issue$_7$ today$_8$.$_9$

Given the dependency graph in Figure 1, the projective order of the words is the following (which corresponds to the word order of the sentence in Figure 2):

ROOT$_0$ A$_1$ hearing$_2$ on$_5$ the$_6$ issue$_7$ is$_3$ scheduled$_4$ today$_8$.$_9$

We now want to explore the idea that (non-projective) dependency parsing can be performed by sorting the words of a sentence into their projective order and deriving a strictly projective dependency graphs for the sorted input. In principle, we could use any one of the many algorithms that have been proposed for sorting, but our desire to combine sorting with a transition-based approach to parsing imposes certain constraints on the kind of algorithm that can be used. First of all, it should be an *online* algorithm, so that we can start sorting (and parsing) before having seen the end of the input, in an incremental left-to-right fashion. Secondly, it should be an *exchange sort*, which sorts by exchanging adjacent elements, so that sorting and parsing transitions can be defined on the same kinds of configurations. One algorithm that satisfies these constraints is *gnome sort*, which is similar to insertion sort, except that moving an element to its proper place is accomplished by a series of swaps, as in bubble sort. The worst-case time complexity of *gnome sort* is $O(n^2)$, but in practice the algorithm can run as fast as insertion sort and is very efficient on nearly sorted lists. This is an attractive property given that dependency graphs for natural language sentences tend to be very nearly projective, which means that the projective order will typically be close to the original word order [25,26].

In order to facilitate integration with the parser defined earlier, we first present a transition-based version of gnome sort, where a configuration is a triple $c = (m, \sigma, \beta)$, consisting of an index m and two lists σ and β, and where we use the two transitions defined in Figure 5. The idea is that the list β contains the list

Transition	m	List (σ)	Buffer (β)
	0	$[\text{R}_0]$	$[\text{A}_1, \ldots, ._9]$
SHIFT	0	$[\text{R}_0, \mathbf{A_1}]$	$[\text{hearing}_2, \ldots, ._9]$
SHIFT	0	$[\text{R}_0, \text{A}_1, \mathbf{hearing_2}]$	$[\text{is}_3, \ldots, ._9]$
SHIFT	0	$[\text{R}_0, \text{hearing}_2, \mathbf{is_3}]$	$[\text{scheduled}_4, \ldots, ._9]$
SHIFT	0	$[\text{R}_0, \ldots, \text{is}_3, \mathbf{scheduled_4}]$	$[\text{on}_5, \ldots, ._9]$
SHIFT	0	$[\text{R}_0, \ldots, \text{scheduled}_4, \mathbf{on_5}]$	$[\text{the}_6, \ldots, ._9]$
SWAP	1	$[\text{R}_0, \ldots, \text{is}_3, \mathbf{on_5}, \text{scheduled}_4]$	$[\text{the}_6, \ldots, ._9]$
SWAP	2	$[\text{R}_0, \text{hearing}_2, \mathbf{on_5}, \text{is}_3, \text{scheduled}_4]$	$[\text{the}_6, \ldots, ._9]$
SHIFT	0	$[\text{R}_0, \ldots, \text{scheduled}_4, \mathbf{the_6}]$	$[\text{issue}_7, \ldots, ._9]$
SWAP	1	$[\text{R}_0, \ldots, \text{is}_3, \mathbf{the_6}, \text{scheduled}_4]$	$[\text{issue}_7, \ldots, ._9]$
SWAP	2	$[\text{R}_0, \ldots, \text{on}_5, \mathbf{the_6}, \text{is}_3, \text{scheduled}_4]$	$[\text{issue}_7, \ldots, ._9]$
SHIFT	0	$[\text{R}_0, \ldots, \text{scheduled}_4, \mathbf{issue_7}]$	$[\text{today}_8, ._9]$
SWAP	1	$[\text{R}_0, \ldots, \text{is}_3, \mathbf{issue_7}, \text{scheduled}_4]$	$[\text{today}_8, ._9]$
SWAP	2	$[\text{R}_0, \ldots, \text{the}_6, \mathbf{issue_7}, \text{is}_3, \text{scheduled}_4]$	$[\text{today}_8, ._9]$
SHIFT	0	$[\text{R}_0, \ldots, \text{scheduled}_4, \mathbf{today_8}]$	$[._9]$
SHIFT	0	$[\text{R}_0, \ldots, \mathbf{._9}]$	$[]$

Fig. 6. Transition sequence for sorting the English sentence in Figure 1 ($\sigma[m]$ in bold)

of remaining words to be sorted, while the list σ contains the words sorted so far, with the index m referring to the position in σ of the word that is being inserted into its proper place (with the first position having index 0). The two transitions work as follows:

1. The SWAP transition swaps the mth and m+1th words in σ and increments the index to m+1 (the position of the word in mth position before the swap).
2. The SHIFT transition takes the next word from β, inserts it at the head of σ and sets the index m to 0 (the position of the newly inserted word).

Note that we use the notation $[\sigma|w_i, w_j|\sigma_m]$ to refer to a list (with its head to the right) with a prefix of m words, followed by the words w_i and w_j and a tail σ of unspecified length.

Assume now that we have an *oracle* o, which maps each configuration to the correct transition (SWAP or SHIFT) in order to sort the words of a sentence into their projective order. Then sorting can be performed using an algorithm that is very similar to the parsing algorithm described in the previous section:

```
SORT(S = w_0 w_1 ··· w_n)
1 c ← (0, [w_0], [w_1, ···, w_n])
2 while β_c ≠ []
3     SHIFT(c)
4     t ← o(c)
5     while t = SWAP
6         c ← t(c)
7         t ← o(c)
8 return σ_c
```

Transition		Condition
SWAP	$(m, [\sigma\|w_i, w_j\|\sigma_m], \beta, A) \Rightarrow (m{+}1, [\sigma\|w_j, w_i\|\sigma_m], \beta, A)$	$i \neq 0$
LEFT-ARC$_l$	$(m, [\sigma\|w_i, w_j\|\sigma_m], \beta, A) \Rightarrow (m, [\sigma\|w_j\|\sigma_m], \beta, A\cup\{(j, l, i)\})$	$i \neq 0$
RIGHT-ARC$_l$	$(m, [\sigma\|w_i, w_j\|\sigma_m], \beta, A) \Rightarrow (m, [\sigma\|w_i\|\sigma_m], \beta, A\cup\{(i, l, j)\})$	
SHIFT	$(m, \sigma, [w_i\|\beta], A) \Rightarrow (0, [\sigma\|w_i], \beta, A)$	

Fig. 7. Transitions for integrated sorting and parsing

The outer **while** loop is executed once for each word to be inserted into its place in the projective order, while the inner **while** loop is executed as many times as the word needs to be swapped with its neighbor in order to reach its place. To illustrate how this sort procedure works, Figure 6 shows the transition sequence for sorting the words of the sentence in Figure 1 into their projective order.

4 Integrated Sorting and Parsing

In the two previous sections, we have shown how to perform projective dependency parsing and how to sort the words of a sentence into their projective order, in both cases relying on oracles for predicting the next transition, which in practice can be approximated by classifiers trained on syntactically annotated sentences. In this section, we will put the two pieces together and define an algorithm that simultaneously sorts the words of a sentence into their projective order and derives a projective dependency graph for the sorted input, which may or may not be non-projective in relation to the original word order.

We let a *configuration* be a quadruple $c = (m, \sigma, \beta, A)$, where m, σ, and β are as in section 3, and where A is a set of dependency arcs as in section 2; we use the transitions in Figure 7, where SWAP and SHIFT are exactly as in section 3, and where LEFT-ARC$_l$ and RIGHT-ARC$_l$ have been modified to apply to the mth and $m{+}1$th word in σ instead of the first and second; and we use the following algorithm:

SORTPARSE($S = w_0 w_1 \cdots w_n$)
1 $c \leftarrow (0, [w_0], [w_1, \ldots, w_n], \emptyset)$
2 **while** $\beta_c \neq [\,]$
3   SHIFT(c)
4     $t \leftarrow o(c)$
5   **while** $t =$ SWAP
6         $c \leftarrow t(c)$
7         $t \leftarrow o(c)$
8   **while** $t \in \{$LEFT-ARC$_l$, RIGHT-ARC$_l\}$
9         $c \leftarrow t(c)$
10        $t \leftarrow o(c)$
11 **return** G_c

Transition	m	**List** (σ)	**Buffer** (β)	**New Arc**
	0	$[\text{R}_0]$	$[\text{A}_1, \ldots, ._9]$	
SHIFT	0	$[\text{R}_0, \mathbf{A}_1]$	$[\text{hearing}_2, \ldots, ._9]$	
SHIFT	0	$[\text{R}_0, \text{A}_1, \mathbf{hearing}_2]$	$[\text{is}_3, \ldots, ._9]$	
LA$_{\text{DET}}$	0	$[\text{R}_0, \mathbf{hearing}_2]$	$[\text{is}_3, \ldots, ._9]$	$(2, \text{DET}, 1)$
SHIFT	0	$[\text{R}_0, \text{hearing}_2, \mathbf{is}_3]$	$[\text{scheduled}_4, \ldots, ._9]$	
SHIFT	0	$[\text{R}_0, \ldots, \text{is}_3, \mathbf{scheduled}_4]$	$[\text{on}_5, \ldots, ._9]$	
SHIFT	0	$[\text{R}_0, \ldots, \text{scheduled}_4, \mathbf{on}_5]$	$[\text{the}_6, \ldots, ._9]$	
SWAP	1	$[\text{R}_0, \ldots, \text{is}_3, \mathbf{on}_5, \text{scheduled}_4]$	$[\text{the}_6, \ldots, ._9]$	
SWAP	2	$[\text{R}_0, \text{hearing}_2, \mathbf{on}_5, \text{is}_3, \text{scheduled}_4]$	$[\text{the}_6, \ldots, ._9]$	
SHIFT	0	$[\text{R}_0, \ldots, \text{scheduled}_4, \mathbf{the}_6]$	$[\text{issue}_7, \ldots, ._9]$	
SWAP	1	$[\text{R}_0, \ldots, \text{is}_3, \mathbf{the}_6, \text{scheduled}_4]$	$[\text{issue}_7, \ldots, ._9]$	
SWAP	2	$[\text{R}_0, \ldots, \text{on}_5, \mathbf{the}_6, \text{is}_3, \text{scheduled}_4]$	$[\text{issue}_7, \ldots, ._9]$	
SHIFT	0	$[\text{R}_0, \ldots, \text{scheduled}_4, \mathbf{issue}_7]$	$[\text{today}_8, ._9]$	
SWAP	1	$[\text{R}_0, \ldots, \text{is}_3, \mathbf{issue}_7, \text{scheduled}_4]$	$[\text{today}_8, ._9]$	
SWAP	2	$[\text{R}_0, \ldots, \text{the}_6, \mathbf{issue}_7, \text{is}_3, \text{scheduled}_4]$	$[\text{today}_8, ._9]$	
LA$_{\text{DET}}$	2	$[\text{R}_0, \ldots, \text{on}_5, \mathbf{issue}_7, \text{is}_3, \text{scheduled}_4]$	$[\text{today}_8, ._9]$	$(7, \text{DET}, 6)$
RA$_{\text{PC}}$	2	$[\text{R}_0, \text{hearing}_2, \mathbf{on}_5, \text{is}_3, \text{scheduled}_4]$	$[\text{today}_8, ._9]$	$(5, \text{PC}, 7)$
RA$_{\text{NMOD}}$	2	$[\text{R}_0, \mathbf{hearing}_2, \text{is}_3, \text{scheduled}_4]$	$[\text{today}_8, \ldots, ._9]$	$(2, \text{NMOD}, 5)$
SHIFT	0	$[\text{R}_0, \ldots, \text{scheduled}_4, \mathbf{today}_8]$	$[._9]$	
RA$_{\text{ADV}}$	0	$[\text{R}_0, \ldots, \text{is}_3, \mathbf{scheduled}_4]$	$[._9]$	$(4, \text{ADV}, 8)$
RA$_{\text{VG}}$	0	$[\text{R}_0, \text{hearing}_2, \mathbf{is}_3]$	$[._9]$	$(3, \text{VG}, 4)$
LA$_{\text{SBJ}}$	0	$[\text{R}_0, \mathbf{is}_3]$	$[._9]$	$(3, \text{SBJ}, 2)$
SHIFT	0	$[\text{R}_0, \text{is}_3, \mathbf{.}_9]$	$[]$	
RA$_{\text{P}}$	0	$[\text{R}_0, \mathbf{is}_3]$	$[]$	$(3, \text{P}, 9)$
RA$_{\text{ROOT}}$	0	$[\text{R}_0]$	$[]$	$(0, \text{ROOT}, 3)$

Fig. 8. Transition sequence for parsing the English sentence in Figure 1 ($\sigma[m]$ in bold)

As before, the outer **while** loop is executed once for each word w_i ($1 \leq i \leq n$), which is inserted at the head of the list σ. The first inner **while** loop inserts w_i in its proper place, by performing the required number of SWAP transitions, and the second inner **while** loop adds the required number of arcs before the next word is shifted to σ. The parsing procedure is exemplified in Figure 8, which shows the transition sequence for parsing the sentence in Figure 1.

Provided that oracle predictions and transitions can both be performed in constant time,[2] the time complexity of the algorithm is $O(n^2)$ in the worst case but $O(n)$ in the best case where the input words are already sorted in the projective order. Since dependency graphs for natural language sentences tend to be very nearly projective, the algorithm can therefore be expected to be very efficient in practice.

[2] The time taken to compute the oracle prediction depends heavily on the time of classifier used but does not in general depend on the length of the input sentence. It can therefore be regarded as a constant in this context, corresponding to the grammar constant in grammar-based approaches to parsing.

5 Conclusion

In this paper, we have explored the idea that the general parsing problem for dependency graphs can be decomposed into a sorting problem and a simpler parsing problem restricted to projective dependency graphs. Based on this idea, we have constructed a parsing algorithm for non-projective dependency graphs by combining an online sorting algorithm with a projective parsing algorithm. The next important step in the exploration of this approach is to develop a practical parsing system by training classifiers to approximate the oracle used to predict the next transition. This methodology has previously proven successful for strictly projective dependency parsing, but it is an open question how well it will perform for the more complex problem of integrated sorting and parsing. Finally, it is worth emphasizing that the projective order and sorting algorithm proposed in this paper only define one of many conceivable realizations of the basic idea of integrated sorting and parsing. Exploring alternative orders and sorting strategies is another important area for future research.

References

1. Tesnière, L.: Éléments de syntaxe structurale. Editions Klincksieck (1959)
2. Sgall, P., Hajičová, E., Panevová, J.: The Meaning of the Sentence in Its Pragmatic Aspects. Reidel (1986)
3. Mel'čuk, I.: Dependency Syntax: Theory and Practice. State University of New York Press (1988)
4. Hudson, R.A.: English Word Grammar. Blackwell, Malden (1990)
5. Culotta, A., Sorensen, J.: Dependency tree kernels for relation extraction. In: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL), pp. 423–429 (2004)
6. Ding, Y., Palmer, M.: Synchronous dependency insertion grammars: A grammar formalism for syntax based statistical MT. In: Proceedings of the Workshop on Recent Advances in Dependency Grammar, pp. 90–97 (2004)
7. Haghighi, A., Ng, A., Manning, C.D.: Robust textual inference via graph matching. In: Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP), pp. 387–394 (2005)
8. Snow, R., Jurafsky, D., Ng, A.Y.: Learning syntactic patterns for automatic hypernym discovery. In: Advances in Neural Information Processing Systems (NIPS) (2005)
9. Wang, M., Smith, N.A., Mitamura, T.: What is the Jeopardy Model? A quasi-synchronous grammar for QA. In: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pp. 22–32 (2007)
10. Buchholz, S., Marsi, E.: CoNLL-X shared task on multilingual dependency parsing. In: Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL), pp. 149–164 (2006)
11. Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., Yuret, D.: The CoNLL 2007 shared task on dependency parsing. In: Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007, pp. 915–932 (2007)

12. McDonald, R., Pereira, F., Ribarov, K., Hajič, J.: 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), pp. 523–530 (2005)
13. Covington, M.A.: A fundamental algorithm for dependency parsing. In: Proceedings of the 39th Annual ACM Southeast Conference, pp. 95–102 (2001)
14. Nivre, J.: Incremental non-projective dependency parsing. In: Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT), pp. 396–403 (2007)
15. Nivre, J., Nilsson, J.: Pseudo-projective dependency parsing. In: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL), pp. 99–106 (2005)
16. Hall, K., Novák, V.: Corrective modeling for non-projective dependency parsing. In: Proceedings of the 9th International Workshop on Parsing Technologies (IWPT), pp. 42–52 (2005)
17. McDonald, R., Pereira, F.: Online learning of approximate dependency parsing algorithms. In: Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL), pp. 81–88 (2006)
18. Nivre, J.: Inductive Dependency Parsing. Springer, Heidelberg (2006)
19. Nivre, J.: Incrementality in deterministic dependency parsing. In: Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL), pp. 50–57 (2004)
20. Nivre, J.: Algorithms for deterministic incremental dependency parsing. Computational Linguistics (to appear)
21. Attardi, G.: Experiments with a multilanguage non-projective dependency parser. In: Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL), pp. 166–170 (2006)
22. Yamada, H., Matsumoto, Y.: Statistical dependency analysis with support vector machines. In: Proceedings of the 8th International Workshop on Parsing Technologies (IWPT), pp. 195–206 (2003)
23. Nivre, J., Hall, J., Nilsson, J.: Memory-based dependency parsing. In: Proceedings of the 8th Conference on Computational Natural Language Learning, pp. 49–56 (2004)
24. Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryiğit, G., Kübler, S., Marinov, S., Marsi, E.: Maltparser: A language-independent system for data-driven dependency parsing. Natural Language Engineering 13, 95–135 (2007)
25. Nivre, J.: Constraints on non-projective dependency graphs. In: Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL), pp. 73–80 (2006)
26. Kuhlmann, M., Nivre, J.: Mildly non-projective dependency structures. In: Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions, pp. 507–514 (2006)