Divisible Transition Systems and Multiplanar Dependency Parsing

Carlos Gómez-Rodríguez*
Universidade da Coruña

Joakim Nivre**
Uppsala University

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Transition-based parsing is a widely used approach for dependency parsing that combines high efficiency with expressive feature models. Many different transition systems have been proposed, often formalized in slightly different frameworks. In this article, we show that a large number of the known systems for projective dependency parsing can be viewed as variants of the same stack-based system with a small set of elementary transitions that can be composed into complex transitions and restricted in different ways. We call these systems divisible transition systems and prove a number of theoretical results about their expressivity and complexity. In particular, we characterize an important subclass called efficient divisible transition systems that parse planar dependency graphs in linear time. We go on to show, first, how this system can be restricted to capture exactly the set of planar dependency trees and, secondly, how the system can be generalized to k*-planar trees by making use of multiple stacks. Using the first known efficient test for* k*-planarity, we investigate the coverage of* k*-planar trees in available dependency treebanks and find a very good fit for 2-planar trees. We end with an experimental evaluation showing that our 2-planar parser gives significant improvements in parsing accuracy over the corresponding 1-planar and projective parsers for data sets with non-projective dependency trees and performs on a par with the widely used arc-eager pseudo-projective parser.*

1. Introduction

Syntactic parsing using dependency-based representations has attracted considerable interest in computational linguistics in recent years, both because it appears to provide a useful interface to downstream applications of parsing and because many dependency parsers combine competitive parsing accuracy with highly efficient processing. Among the most efficient systems available are transition-based dependency parsers, which perform a greedy search through a transition system, or abstract state machines, that map sentences to dependency trees, guided by statistical models trained on treebank

* Departamento de Computación, Universidade da Coruña, Facultad de Informtica, Campus de Elviña s/n, 15071 A Coruña, Spain. E-mail: cgomezr@udc.es.
** Department of Linguistics and Philology, Uppsala University, Box 635, 75126 Uppsala, Sweden. E-mail: joakim.nivre@lingfil.uu.se.

Submission received: 13 October 2011; revised submission received: 29 August 2012; accepted for publication: 7 November 2012.

doi:10.1162/COLI_a_00150

© 2013 Association for Computational Linguistics

data (Yamada and Matsumoto 2003; Nivre, Hall, and Nilsson 2004; Attardi 2006; Zhang and Clark 2008). Transition systems for dependency parsing come in many different varieties, and our aim in the first part of this article is to deepen our understanding of these systems by analyzing them in a uniform framework.

More precisely, we demonstrate that a number of well-known systems from the literature can all be viewed as variants of a stack-based system with five elementary transitions, where different variants are obtained by composing elementary transitions into complex transitions and by adding restrictions on their applicability. We call such systems **divisible** transition systems and prove a number of theoretical results about their expressivity (which classes of dependency graphs they can handle) and their complexity (what upper bounds exist on the length of transition sequences). In particular, we show that an important subclass called **efficient** divisible transition systems derive planar dependency graphs in time that is linear in the length of the sentence using standard inference methods for transition-based dependency parsing. Even though many of these results were already known for particular systems, the general framework allows us to derive these results from more general principles and thereby to establish connections between previously unrelated systems. We then go on to show that there are interesting cases of efficient divisible transition systems that have not yet been explored, notably a system that is sound and complete for **planar** dependency trees, a mild extension to the class of projective trees that are assumed in most existing systems.

In the second part of the article, we take the planar parsing system as our point of departure for addressing the problem of non-projective dependency parsing. Despite the impressive results obtained with dependency parsers limited to strictly projective dependency trees—that is, trees where every subtree has a contiguous yield—it is clear that most if not all languages have syntactic constructions whose analysis requires non-projective trees. It is also clear, however, that allowing arbitrary non-projective trees makes parsing computationally hard (McDonald and Satta 2007) and does not seem justified by the data in available treebanks (Kuhlmann and Nivre 2006; Nivre 2006a; Havelka 2007). This suggests that we should try to find a superset of projective trees that is permissive enough to encompass constructions found in natural language yet restricted enough to permit efficient parsing. Proposals for such a set include trees with bounded arc degree (Nivre 2006a, 2007), well-nested trees with bounded gap degree (Kuhlmann and Nivre 2006; Kuhlmann and Möhl 2007), as well as trees parsable by a particular transition system such as that proposed by Attardi (2006).

In the same vein, Yli-Jyrä (2003) introduced the concept of **multiplanarity**, which generalizes the simple notion of planarity by saying that a dependency tree is k-planar if it can be decomposed into at most k planar subgraphs, a proposal that remains largely unexplored because an efficient test for k-planarity has been lacking. In this article, we construct a test for k-planarity by reducing it to a graph coloring problem. Applying this test to a wide range of dependency treebanks, we show that, although simple planarity (or 1-planarity) is clearly insufficient (Kuhlmann and Nivre 2006), the set of 2-planar dependency trees gives a very good fit with the available data, better than many of the previously proposed superclasses of projective trees. We then demonstrate how the transition system for planar dependency parsing can be generalized to k-planarity by introducing additional stacks. In particular, we define a two-stack system for 2-planar dependency parsing that is provably correct and has linear complexity. Finally, we show that the 2-planar parser, when evaluated on data sets with a non-negligible proportion of non-projective trees, gives significant improvements in parsing accuracy over the corresponding 1-planar and projective parsers, and provides comparable accuracy to the widely used arc-eager pseudo-projective parser.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

The remainder of the article is structured as follows. Section 2 reviews basic concepts of dependency parsing and in particular the formalization of stack-based transition systems from Nivre (2008). Section 3 introduces our system of elementary transitions, uses it to analyze a number of parsing algorithms from the literature as divisible transition systems, proves a number of theoretical results about the expressivity and complexity of such systems, and finally introduces a divisible transition system for 1-planar dependency parsing. Section 4 reviews the notion of multiplanarity, introduces an efficient procedure for determining the smallest k for which a dependency tree is k-planar, and uses this procedure in an empirical investigation of available dependency treebanks. Section 5 shows how the divisible transition system framework and the 1-planar parser can be generalized to handle k-planar trees by introducing additional stacks, presents proofs of correctness and complexity for the 2-planar case, and reports the results of an experimental evaluation of projective, pseudo-projective, 1-planar and 2-planar dependency parsing. Section 6 reviews related work, and Section 7 concludes and makes suggestions for future research.

Part of the contributions in this article (namely, the test for multiplanarity and the 1-planar and 2-planar parsers) have been published previously by Gómez-Rodríguez and Nivre (2010); this article substantially revises and extends the ideas presented in that paper. The framework of divisible transition systems and all the derived theoretical results, including the properties and proofs regarding the 1-planar and 2-planar parsers, are entirely new contributions of this article.

2. Dependency Parsing

Dependency parsing is based on the idea that syntactic structure can be analyzed in terms of binary, asymmetric relations between the words of a sentence, an idea that has a long tradition in descriptive and theoretical linguistics (Tesnière 1959; Sgall, Hajičová, and Panevová 1986; Mel'čuk 1988; Hudson 1990). In computational linguistics, dependency structures have become increasingly popular in the interface to downstream applications of parsing, such as information extraction (Culotta and Sorensen 2004; Stevenson and Greenwood 2006; Buyko and Hahn 2010), question answering (Shen and Klakow 2006; Bikel and Castelli 2008), and machine translation (Quirk, Menezes, and Cherry 2005; Xu et al. 2009). And although dependency structures can easily be extracted from other syntactic representations, such as phrase structure trees, this has also led to an increased interest in statistical parsers that specifically produce dependency trees (Eisner 1996; Yamada and Matsumoto 2003; Nivre, Hall, and Nilsson 2004; McDonald, Crammer, and Pereira 2005).

Current approaches to statistical dependency parsing can be broadly grouped into **graph-based** and **transition-based** techniques (McDonald and Nivre 2007). Graph-based parsers parameterize the parsing problem by the structure of the dependency trees and learn models for scoring entire parse trees for a given sentence. Many of these models permit exact inference using dynamic programming (Eisner 1996; McDonald, Crammer, and Pereira 2005; Carreras 2007; Koo and Collins 2010), but recent work has explored approximate search methods in order to widen the scope of features especially when processing non-projective trees (McDonald and Pereira 2006; Riedel and Clarke 2006; Nakagawa 2007; Smith and Eisner 2008; Martins, Smith, and Xing 2009; Koo et al. 2010; Martins et al. 2010). Transition-based parsers parameterize the parsing problem by the structure of a transition system, or abstract state machine, for mapping sentences to dependency trees and learn models for scoring individual transitions from one state to the other. Traditionally, transition-based parsers have relied on local optimization and

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

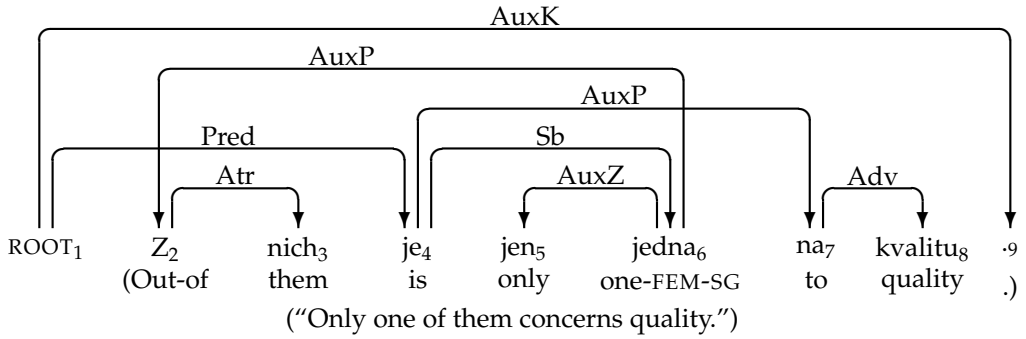

Figure 1
Dependency graph for a Czech sentence from the Prague Dependency Treebank.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

greedy, deterministic parsing (Yamada and Matsumoto 2003; Nivre, Hall, and Nilsson 2004; Attardi 2006; Nivre 2008), but globally trained models and non-greedy parsing methods such as beam search are increasingly used (Johansson and Nugues 2006; Titov and Henderson 2007; Zhang and Clark 2008; Huang, Jiang, and Liu 2009; Huang and Sagae 2010; Zhang and Nivre 2011). In empirical evaluations, the two main approaches to dependency parsing often achieve very similar accuracy, but transition-based parsers tend to be more efficient. In this article, we will be concerned exclusively with transition-based models.

In the remainder of this background section, we first introduce the syntactic representations used by dependency parsers, starting from a general characterization of dependency graphs and discussing a number of different restrictions of this class that will be relevant for the analysis later on. We then go on to review the formalization of transition systems proposed by Nivre (2008), and in particular the class of stack-based systems that provides the framework for our discussion of existing and novel transition-based models. Finally, we discuss the implementation of efficient parsers based on these transition systems.

2.1 Dependency Graphs

In dependency parsing, the syntactic structure of a sentence is modeled by a **dependency graph**, which represents each token and its syntactic dependents through labeled, directed arcs. This is exemplified in Figure 1 for a Czech sentence taken from the Prague Dependency Treebank (Hajič et al. 2001; Böhmová et al. 2003), and in Figure 2 for an English sentence taken from the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993; Marcus et al. 1994).[1] In the former case, an artificial token ROOT has been inserted at the beginning of the sentence, serving as the unique root of the graph and ensuring that the graph is a tree even if more than one token is independent of all other tokens. In the latter case, no such device has been used, and we will not in general assume the existence of an artificial root node prefixed to the sentence, although all our models will be compatible with such a device.

1 The dependency graph has in this case been derived automatically from the constituency-based annotation in the treebank using standard head-finding rules and heuristics for inferring dependency labels.

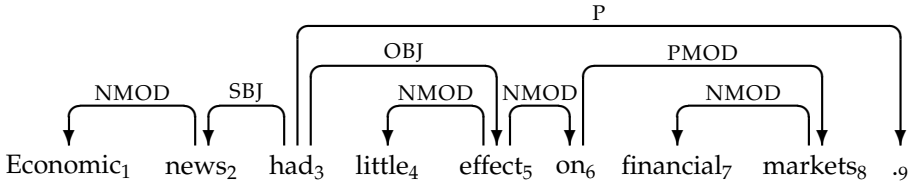

Figure 2
Dependency graph for an English sentence from the Penn Treebank.

Definition 1

A **dependency graph** for a sentence $x = w_1, \ldots, w_n$ is a directed graph $G = (V, A)$, where

1. $V = \{1, \ldots, n\}$ is a set of nodes,
2. $A \subseteq V \times V$ is a set of directed arcs, containing no loops (i.e., arcs of the form (v, v) are disallowed for all $v \in V$).

The set V of **nodes** (or **vertices**) is the set of positive integers up to and including n, each corresponding to the linear position of a token in the sentence (where the first token may or may not be the special token ROOT). The set A of **arcs** (or **directed edges**) is a set of pairs (i, j), where i and j are distinct nodes. Because arcs are used to represent dependency relations, we say that i is the **head** of j; conversely, we say that j is a **dependent** of i. A node with no incoming arcs is called a **root**.

We will say that two arcs (i, j) and (k, l) **cross** if $\min(i, j) < \min(k, l) < \max(i, j) < \max(k, l)$ or $\min(k, l) < \min(i, j) < \max(k, l) < \max(i, j)$, and that an arc (i, j) **covers** a node k if $\min(i, j) < k < \max(i, j)$.

Note that the dependency graphs defined by Definition 1 are **unlabeled** dependency graphs. Adding labels is straightforward by redefining arcs as triples (i, l, j), consisting of a head i, a label l, and a dependent j, but excluding labels for now will simplify the formal analysis without limiting the generality of the results. We will discuss the generalization to labeled dependency graphs whenever relevant, and the experiments reported in Section 5 all use labeled graphs.

Definition 2

Let $G = (V, A)$ be a dependency graph.

1. SINGLE-HEAD(G) $\Leftrightarrow$ every node in G has at most one incoming arc.
2. ACYCLIC(G) $\Leftrightarrow$ there are no (directed) cycles in G.
3. CONNECTED(G) $\Leftrightarrow$ G is weakly connected.
4. TREE(G) $\Leftrightarrow$ G is a directed tree.
5. PLANAR(G) $\Leftrightarrow$ there are no crossing arcs in G.
6. NO-COVERED-ROOTS(G) $\Leftrightarrow$ there is no root covered by an arc in G.
7. PROJECTIVE(G) $\Leftrightarrow$ PLANAR(G) and NO-COVERED-ROOTS(G).

Definition 2 lists a number of constraints that can be imposed on dependency graphs. The most common of these is the TREE constraint, which requires that there is a root from which all other nodes are reachable by a unique directed path, and which in turn entails SINGLE-HEAD, ACYCLIC, and CONNECTED. A dependency graph that satisfies the TREE constraint is called a **dependency tree**.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

The final three constraints are usually defined only for dependency trees, although we have extended them to apply to dependency graphs in general. The most common of these is the PROJECTIVE constraint, which for dependency trees is equivalent to the requirement that every subtree must have a contiguous yield and rules out both crossing arcs and covered roots. By contrast, the PLANAR constraint forbids crossing arcs but allows covered roots, which in the case of dependency trees is a very mild relaxation because there can be at most one covered root without violating the TREE constraint.

Example 1

Consider the dependency graphs depicted in Figures 1 and 2, ignoring labels for the time being:

$$\begin{array}{lrcl} \text{Figure 1:} & G_1 & = & (V_1, A_1) \\ & V_1 & = & \{1,2,3,4,5,6,7,8,9\} \\ & A_1 & = & \{(1,4),(1,9),(2,3),(4,6),(4,7),(6,2),(6,5),(7,8)\} \\ \text{Figure 2:} & G_2 & = & (V_2, A_2) \\ & V_2 & = & \{1,2,3,4,5,6,7,8,9\} \\ & A_2 & = & \{(2,1),(3,2),(3,5),(3,9),(5,4),(5,6),(6,8),(8,7)\} \end{array}$$

G_1 satisfies TREE (hence also SINGLE-HEAD, ACYCLIC, and CONNECTED) and NO-COVERED-ROOTS but violates PLANAR (hence also PROJECTIVE) because there are crossing arcs. By contrast, G_2 satisfies all constraints listed in Definition 2.

2.2 Transition Systems for Dependency Parsing

Transition-based dependency parsing is based on the notion of a **transition system**, or abstract state machine, for mapping sentences to dependency graphs. Such systems are nondeterministic in general and usually combined with heuristic search, guided by a treebank-induced function for scoring different transitions out of a given configuration. For the time being, we will ignore the details of the search procedure and concentrate on the underlying transition systems. We will adopt the general framework of Nivre (2008) but restricted to *stack-based* systems.[2]

Definition 3

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

1. C is a set of **configurations**, each of which contains a buffer β of (remaining) nodes and a set A of dependency arcs,
2. T is a set of **transitions**, each of which is a (partial) function $t : C \rightarrow C$,
3. c_s is an **initialization function**, mapping a sentence $x = w_1, \ldots, w_n$ to a configuration with $\beta = [1, \ldots, n]$,
4. $C_t \subseteq C$ is a set of **terminal configurations**.

2 In addition to stack-based systems, Nivre (2008) also investigates *list-based* systems, which make use of arbitrary lists instead of stacks that obey the last-in first-out constraint.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

In **stack-based** transition systems, a configuration takes the form of a triple $c = (\sigma, \beta, A)$, where σ is a stack of nodes, β is a buffer of nodes, and A is a set of dependency arcs; the initialization function is $c_s(x) = ([\,], [1, \ldots, n], \varnothing)$ (for $x = w_1, \ldots, w_n$); and the set of terminal configurations is $C_t = \{c \mid c = (\sigma, [\,], A) \text{ for any } \sigma, A\}$ (Nivre 2008).

We use the notation σ_c, β_c, and A_c to refer to the value of σ, β, and A in a given configuration c; we use $|\sigma|$ and $|\beta|$ to refer to the size of σ and β (i.e., the number of nodes), and we use $[\,]$ to denote an empty stack or buffer.

Definition 4

Let $S = (C, T, c_s, C_t)$ be a transition system. A **transition sequence**[3] for a sentence $x = w_1, \ldots, w_n$ in S is a sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ of configurations, such that

1. $c_0 = c_s(x)$,
2. $c_m \in C_t$,
3. for every i $(1 \leq i \leq m)$, $c_i = t(c_{i-1})$ for some $t \in T$.

The **parse** assigned to x by $C_{0,m}$ is the dependency graph $G_{c_m} = (\{1, \ldots, n\}, A_{c_m})$, where A_{c_m} is the set of dependency arcs in c_m.

Starting from the initial configuration for the sentence to be parsed, transitions will manipulate σ, β, and A until a terminal configuration is reached (β is empty). Because the node set V is given by the input sentence itself, the set A_{c_m} of dependency arcs in the terminal configuration will determine the output dependency graph $G_{c_m} = (V, A_{c_m})$.

Definition 5

Let $S = (C, T, c_s, C_t)$ be a transition system for dependency parsing.

1. S is **sound** for a class $\mathbb{G}$ of dependency graphs if and only if, for every sentence x and every transition sequence $C_{0,m}$ for x in S, the parse $G_{c_m} \in \mathbb{G}$.
2. S is **complete** for a class $\mathbb{G}$ of dependency graphs if and only if, for every sentence x and every dependency graph G_x for x in $\mathbb{G}$, there is a transition sequence $C_{0,m}$ for x in S such that $G_{c_m} = G_x$.
3. S is **correct** for a class $\mathbb{G}$ of dependency graphs if and only if it is sound and complete for $\mathbb{G}$.

As observed by Nivre (2008), soundness and completeness for transition systems are analogous to soundness and completeness for grammar parsing algorithms, according to which an algorithm is sound if it only derives parses licensed by the grammar and complete if it derives all such parses (Shieber, Schabes, and Pereira 1995).

3 Please note that, according to standard terminology both in transition-based dependency parsing and for transition systems more generally in computer science, a *transition sequence* is a sequence of *configurations*, not a sequence of *transitions*. We will later introduce the term *transition chain* for the corresponding sequence of *transitions*. We realize that these terms are potentially confusing but prefer not to deviate from the standard terminology.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Example 2

Nivre's (2008) arc-standard transition system uses three transitions:

$$\begin{array}{ll} \textsc{Shift}_{AS} & (\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A) \\ \textsc{Left-Arc}_{AS} & (\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i)\}) \\ \textsc{Right-Arc}_{AS} & (\sigma|i, j|\beta, A) \Rightarrow (\sigma, i|\beta, A \cup \{(i, j)\}) \end{array}$$

The unlabeled dependency graph in Figure 2 is derived by the transition sequence in Figure 3. For labeled dependency parsing, the LEFT-ARC and RIGHT-ARC transitions in addition have a parameter for the label l of the arc being added.

2.3 Transition-Based Parsing

A transition system is an abstract machine that computes the mapping of a sentence to a dependency graph through a sequence of steps called transitions. In order to build a practical parsing system on top of this, we essentially need two additional components: a model for scoring transition sequences and an algorithm for finding the optimal transition sequence for a given sentence. Although many different scoring models are conceivable, practically all existing parsers use a linear model for scoring individual transitions whose scores are then added to get the score for an entire sequence:

$$\textsc{Score}(C_{0,m}) = \sum_{i=1}^{m} \mathbf{f}(c_{i-1}, t_i) \cdot \mathbf{w}$$

where $\mathbf{f}(c_{i-1}, t_i)$ is a feature vector representation of transition t_i out of configuration c_{i-1} and $\mathbf{w}$ is a corresponding weight vector. Finding the highest scoring transition

		(	[ ],	$[1, \ldots, 9]$,	$\varnothing$)
SHIFT	$\Rightarrow$	(	[1],	$[2, \ldots, 9]$,	$\varnothing$)
LEFT-ARC	$\Rightarrow$	(	[ ],	$[2, \ldots, 9]$,	$A_1 = \{(2, 1)\}$)
SHIFT	$\Rightarrow$	(	[2],	$[3, \ldots, 9]$,	A_1)
LEFT-ARC	$\Rightarrow$	(	[ ],	$[3, \ldots, 9]$,	$A_2 = A_1 \cup \{(3, 2)\}$)
SHIFT	$\Rightarrow$	(	[3],	$[4, \ldots, 9]$,	A_2)
SHIFT	$\Rightarrow$	(	[3, 4],	$[5, \ldots, 9]$,	A_2)
LEFT-ARC	$\Rightarrow$	(	[3],	$[5, \ldots, 9]$,	$A_3 = A_2 \cup \{(5, 4)\}$)
SHIFT	$\Rightarrow$	(	[3, 5],	$[6, \ldots, 9]$,	A_3)
SHIFT	$\Rightarrow$	(	[3, 5, 6],	$[7, 8, 9]$,	A_3)
SHIFT	$\Rightarrow$	(	$[3, \ldots, 7]$,	$[8, 9]$,	A_3)
LEFT-ARC	$\Rightarrow$	(	[3, 5, 6],	$[8, 9]$,	$A_4 = A_3 \cup \{(8, 7)\}$)
RIGHT-ARC	$\Rightarrow$	(	[3, 5],	$[6, 9]$,	$A_5 = A_4 \cup \{(6, 8)\}$)
RIGHT-ARC	$\Rightarrow$	(	[3],	$[5, 9]$,	$A_6 = A_5 \cup \{(5, 6)\}$)
RIGHT-ARC	$\Rightarrow$	(	[ ],	$[3, 9]$,	$A_7 = A_6 \cup \{(3, 5)\}$)
SHIFT	$\Rightarrow$	(	[3],	[9],	A_7)
RIGHT-ARC	$\Rightarrow$	(	[ ],	[3],	$A_8 = A_7 \cup \{(3, 9)\}$)
SHIFT	$\Rightarrow$	(	[3],	[ ],	A_8)

Figure 3
Arc-standard transition sequence for the (unlabeled) dependency graph in Figure 2.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

sequence under this model is a hard problem in general, and transition-based parsers therefore have to rely on heuristic search for the optimal transition sequence. Many systems simply use greedy 1-best search (Yamada and Matsumoto 2003; Nivre, Hall, and Nilsson 2004; Attardi 2006):

```
PARSE(x = (w_1, ..., w_n))
1  c ← c_s(x)
2  while c ∉ C_t
3        t* ← argmax_t f(c, t) · w
4        c ← t*(c)
5  return G_c
```

Another common approach is to use beam search with a fixed beam size (Johansson and Nugues 2006; Titov and Henderson 2007; Zhang and Clark 2008). In this case, lines 3 and 4 are replaced by an inner loop that expands all configurations in the current beam using all permissible transitions and then discards all except the k highest scoring configurations. The outer loop terminates when all configurations in the beam are terminal, and the dependency graph corresponding to the highest scoring configuration is returned. Setting the beam size to 1 makes this equivalent to greedy 1-best search.

The time complexity of transition-based parsing depends not only on the underlying transition system but also on the scoring model and the search algorithm. As long as the number of configurations considered by the search algorithm is bounded by a constant k and as long as every transition can be scored and executed in constant time relative to a fixed model, however, then the asymptotic time complexity of a parser using a transition system S is given by an upper bound on the length of transition sequences in S (Nivre 2008). Similarly, the space complexity is given by an upper bound on the size of a configuration $c \in C$, because at most k configurations need to be stored at any given time. For most of the systems considered in this article, we will see that the length of a transition sequence is $O(n)$, where n is the length of the input sentence, which translates into a linear bound on parsing time for transition-based parsers using beam search (with greedy 1-best search as a special case).

Transition-based dependency parsing using beam search has the advantage of low parsing complexity in combination with very few restrictions on feature representations, which enables fast and accurate parsing, but does not guarantee that the optimal transition sequence is found. Recent work on tabularization for transition-based parsing has shown that it is possible to use exact dynamic programming under certain conditions, but this leads either to very inefficient parsing or to very restricted feature representations. Thus, Huang and Sagae (2010) present a dynamic programming scheme for a feature-rich arc-standard parser, but the resulting parsing complexity is $O(n^7)$ and they therefore have to resort to beam search in practical parsing experiments. Conversely, Kuhlmann, Gómez-Rodríguez, and Satta (2011) show how to obtain cubic complexity for a tabularized arc-eager parser but only for very impoverished feature representations. Hence, for the remainder of this article, we will assume that transition sequence length is a relevant complexity bound, because it translates into a bound on running time for parsers that use beam search, as practically all state-of-the-art transition-based parsers currently do. This bound holds as long as every transition can be scored and executed in constant time, which is true even when including complex features like the valency features of Zhang and Nivre (2011), which are expensive to use in dynamic programming because of the combinatorial effect they have on parsing complexity.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

3. Divisible Transition Systems

In the last decade, several different dependency parsers have been defined as stack-based transition systems, which differ from each other in the order in which they add dependency arcs as well as in the constraints that they impose on output dependency graphs. In their original definitions, these differences arise from the fact that each algorithm uses a distinct set of transitions. In this section, we show how these algorithms can be expressed using a common set of transitions, which we call **elementary transitions**. Under this framework, the original transitions of each algorithm are viewed as combinations of one or more elementary transitions by means of the standard function operations of composition and restriction. A direct consequence of this is that each of the parsers expressed in this framework can be viewed as a restriction of the algorithm that uses elementary transitions directly, allowing any possible concatenation of elementary transitions. We call the systems that are analyzable within this framework **divisible transition systems**.

The elementary transitions in our framework represent five primitive operations that can be applied to stack-based configurations:

$$\begin{array}{ll}
\textsc{Shift} & (\sigma, j|\beta, A) \Rightarrow (\sigma|j, \beta, A) \\
\textsc{Unshift} & (\sigma|i, \beta, A) \Rightarrow (\sigma, i|\beta, A) \\
\textsc{Reduce} & (\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A) \\
\textsc{Left-Arc} & (\sigma|i, j|\beta, A) \Rightarrow (\sigma|i, j|\beta, A \cup \{(j, i)\}) \\
\textsc{Right-Arc} & (\sigma|i, j|\beta, A) \Rightarrow (\sigma|i, j|\beta, A \cup \{(i, j)\})
\end{array}$$

The first three operations modify the stack and/or buffer by moving a word from the buffer to the top of the stack (SHIFT), moving a word from the stack to the buffer (UNSHIFT), or popping a word from the stack (REDUCE). The remaining two operations create dependency arcs involving the top of the stack and the first word in the buffer (LEFT-ARC, RIGHT-ARC). We assume that LEFT-ARC and RIGHT-ARC only apply to configurations where the new arc is not already an element of the arc set A, an assumption that is needed in certain cases to guarantee termination (that is, to rule out transition sequences where the same arc is added an indefinite number of times). Note that, in the case of labeled dependency graphs, the LEFT-ARC and RIGHT-ARC transitions will have a label parameter, and this restriction should not prevent the addition of an arc with the same head and dependent as one or more existing arcs, as long as the label is different.

Different parsing algorithms can now be defined using *composition* of elementary transitions, which is defined as standard function composition.

Definition 6

Let $t_1, t_2 : C \rightarrow C$ be transitions. Their **composition** is the partial function $t_1; t_2 : C \rightarrow C$ mapping each $c \in C$ to $t_2(t_1(c))$.

Elementary transitions are defined as partial functions $t : C \rightarrow C$, and we use T_e to refer to the set of elementary transitions. In addition, we use function restriction to impose constraints on their domain, traditionally expressed in the literature as side conditions.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

For this purpose, we use the standard notation by which the **restriction** of a function $f : X \rightarrow Y$ to a subset $A \subseteq X$ is written as:

$$f|_A(x) = \begin{cases} f(x) & \text{if } x \in A \\ \text{undefined} & \text{if } x \notin A \end{cases}$$

Transition systems that can be defined using composition of elementary transitions with restrictions are said to be *divisible*.

Definition 7

A stack-based transition system $S = (C, T, c_s, C_t)$ is **divisible** if and only if every transition in T is of the form $t_1|_{s_1}; t_2|_{s_2}; \ldots; t_p|_{s_p}$, where $p > 0, t_i \in T_e, s_i \subseteq C$.

In other words, a stack-based transition system is divisible if and only if each of its transitions can be written as a composition of restrictions of the elementary transitions SHIFT, UNSHIFT, REDUCE, LEFT-ARC, and RIGHT-ARC. Note that the definition allows the use of unrestricted elementary transitions in the composition, because for any transition t, we have that $t|_C = t$.[4]

3.1 Examples of Divisible Transition Systems

In this section, we show that a number of transition-based parsers from the literature use divisible transition systems that can be defined using only elementary transitions. This includes the arc-eager and arc-standard projective parsers described in Nivre (2003) and Nivre (2008), the arc-eager and arc-standard parsers for directed acyclic graphs from Sagae and Tsujii (2008), the hybrid parser of Kuhlmann, Gómez-Rodríguez, and Satta (2011), and the easy-first parser of Goldberg and Elhadad (2010). We also give examples of transition systems that are not divisible (Attardi 2006; Nivre 2009).

First of all, we define four standard subsets of the configuration set C:

$$H_\sigma(C) = \{(\sigma|i, \beta, A) \in C \mid \exists j : (j, i) \in A\}$$
$$\overline{H_\sigma(C)} = \{(\sigma|i, \beta, A) \in C \mid \neg\exists j : (j, i) \in A\}$$
$$H_\beta(C) = \{(\sigma, i|\beta, A) \in C \mid \exists j : (j, i) \in A\}$$
$$\overline{H_\beta(C)} = \{(\sigma, i|\beta, A) \in C \mid \neg\exists j : (j, i) \in A\}$$

The set $H_\sigma(C)$ is the subset of configurations where the node on top of the stack has been assigned a head in A, and $\overline{H_\sigma(C)}$ is the subset where the top node has *not* been assigned a head in A. Similarly, the set $H_\beta(C)$ is the subset of configurations where the first node in the buffer has been assigned a head in A, and $\overline{H_\sigma(C)}$ is the subset where the first node has *not* been assigned a head in A. Note that there are configurations that are neither in $H_\sigma(C)$ nor in $\overline{H_\sigma(C)}$, namely, those where the stack is empty. There are also

4 It is worth noting that the assumption that LEFT-ARC and RIGHT-ARC only apply to configurations where the new arc is not already in the arc set A could be formally stated by restricting these transitions to the sets $LA = \{(\sigma|i, j|\beta, A) \mid (j, i) \notin A\}$ (for LEFT-ARC) and $RA = \{(\sigma|i, j|\beta, A) \mid (i, j) \notin A\}$ (for RIGHT-ARC). Because these restrictions are part of the definition of the elementary transitions themselves, however, we prefer to leave it implicit notationwise.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

configurations that are neither in $H_\beta(C)$ nor in $\overline{H_\beta(C)}$, because the buffer is empty, but these are all terminal configurations.

Example 3

Nivre's (2008) arc-standard parser, previously defined in Example 2, is a bottom–up parser for projective dependency trees. Its transitions can be defined in terms of elementary transitions as follows:

$$\begin{aligned} \textsc{Shift}_{AS} &= \textsc{Shift} \\ \textsc{Left-Arc}_{AS} &= \textsc{Left-Arc}; \textsc{Reduce} \\ \textsc{Right-Arc}_{AS} &= \textsc{Right-Arc}; \textsc{Shift}; \textsc{Reduce}; \textsc{Unshift} \end{aligned}$$

The $\textsc{Shift}_{AS}$ transition is the same as the elementary SHIFT transition. The $\textsc{Left-Arc}_{AS}$ transition composes the elementary LEFT-ARC transition with the REDUCE transition to ensure that the left dependent of the new arc is popped from the stack and therefore cannot be assigned more than one head. The $\textsc{Right-Arc}_{AS}$ transition, finally, composes four elementary transitions, where RIGHT-ARC is responsible for adding a left-headed arc, SHIFT and REDUCE jointly remove the dependent of the new arc from the buffer, and UNSHIFT moves the head of the new arc back to the buffer so that it can find a head to the left. It is worth noting that the arc-standard system for projective trees does not make use of restrictions.

Although this description of the arc-standard parser corresponds to its definition in Nivre (2008), where arcs are created involving the topmost stack node and the first buffer node, the system has also been presented in an equivalent form with arcs built between the two top nodes in the stack (Nivre 2004). This variant can also be described as a divisible transition system, with $\textsc{Left-Arc}_{AS'} = \textsc{Unshift}; \textsc{Left-Arc}; \textsc{Reduce}; \textsc{Shift}$ and $\textsc{Right-Arc}_{AS'} = \textsc{Unshift}; \textsc{Right-Arc}; \textsc{Shift}; \textsc{Reduce}$.[5]

Example 4

Nivre's (2003) arc-eager parser is a parser for projective dependency trees, which adds arcs in a strict left-to-right order using the following transitions:

$$\begin{aligned} \textsc{Shift}_{AE} &= \textsc{Shift} \\ \textsc{Reduce}_{AE} &= \textsc{Reduce}|_{H_\sigma(C)} \\ \textsc{Left-Arc}_{AE} &= \textsc{Left-Arc}|_{\overline{H_\sigma(C)}}; \textsc{Reduce} \\ \textsc{Right-Arc}_{AE} &= \textsc{Right-Arc}; \textsc{Shift} \end{aligned}$$

As in the first example, the $\textsc{Shift}_{AE}$ transition is equivalent to the elementary SHIFT transition, but the $\textsc{Right-Arc}_{AE}$ transition differs from $\textsc{Right-Arc}_{AS}$ by not popping

5 Additionally, it is also possible to define the divisible transition system framework itself in such a way that the LEFT-ARC and RIGHT-ARC elementary transitions themselves act upon the two topmost stack nodes, rather than on the topmost stack node and first buffer node. Although this definition can capture exactly the same set of parsers as the one we are using and makes it more natural to describe the mentioned arc-standard variant, we have not used it because it significantly complicates the definition of other algorithms, such as the arc-eager or 1-planar parsers.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

the right dependent from the stack after adding the arc and shifting. Instead, right dependents are removed from the stack in a separate transition REDUCE_{AE}, which is equivalent to the elementary transition REDUCE but restricted to $\overline{H_\sigma(C)}$ to ensure that unattached nodes are not removed. The LEFT-ARC_{AE} transition, finally, is the same as LEFT-ARC_{AS} but restricted to $H_\sigma(C)$, a restriction that is not needed in the arc-standard system where nodes on the stack can never have a head.

Example 5

The easy-first parser of Goldberg and Elhadad (2010) is a parser for projective trees that adds arcs in a bottom–up order but in a non-directional manner, trying to make the easier attachment decisions first regardless of the position of the corresponding words in the sentence. This parsing strategy corresponds to the following divisible transition system:

$$\begin{aligned}
\text{SHIFT}_{EF} &= \text{SHIFT} \\
\text{ATTACH-RIGHT}(i)_{EF} &= \text{SHIFT}^i; \text{LEFT-ARC}; \text{REDUCE}; \text{UNSHIFT}^{i-1} \\
\text{ATTACH-LEFT}(i)_{EF} &= \text{SHIFT}^i; \text{RIGHT-ARC}; \text{SHIFT}; \text{REDUCE}; \text{UNSHIFT}; \text{UNSHIFT}^{i-1}
\end{aligned}$$

where i is a strictly positive integer. Note that this means that the system has an infinite set of transitions. In practice, however, only the $\text{ATTACH-RIGHT}(i)_{EF}$ and $\text{ATTACH-LEFT}(i)_{EF}$ transitions such that $1 \leq i \leq n-1$ need to be considered when parsing a string of length n: Because the number of nodes in the buffer is bounded by n, transitions with $i \geq n$ will always be undefined because the buffer will become empty before the first $i+1$ elementary transitions can be applied. Therefore, to parse strings of length n we only need $2n-1$ transitions.

The purpose of an $\text{ATTACH-RIGHT}(i)_{EF}$ (or $\text{ATTACH-LEFT}(i)_{EF}$) is to create a rightward (or leftward) arc involving the ith and $(i+1)$th words in the input string, and then remove the dependent. This means that the system is not limited to building arcs in a predetermined order (such as left to right). Instead, it can generate the same tree in different orders depending on the criterion used to choose a transition at each configuration. In particular, the parser by Goldberg and Elhadad (2010) can be seen as an implementation of this transition system, which uses a training algorithm that assigns a weight to each of the $\text{ATTACH-RIGHT}(i)_{EF}$ and $\text{ATTACH-LEFT}(i)_{EF}$ transitions in such a way that "easier" (more reliable) attachments are performed first.

The $\text{ATTACH-LEFT}(i)_{EF}$ and $\text{ATTACH-RIGHT}(i)_{EF}$ transitions are essentially the same as LEFT-ARC_{AS} and RIGHT-ARC_{AS} in the arc-standard system, but preceded by i instances of SHIFT and succeeded by $i-1$ instances of UNSHIFT, which means that a separate SHIFT transition is needed only to reach a terminal configuration by pushing the final root(s) onto the stack. This analysis reveals that the two systems are similar in that they build dependency trees bottom–up but differ with respect to the order in which arcs are added. It is worth pointing out that using sequences of SHIFT and UNSHIFT transitions is not the most efficient way of implementing easy-first parsing in practice.

Example 6

The hybrid parser introduced by Kuhlmann, Gómez-Rodríguez, and Satta (2011) is a bottom–up projective transition system that builds each given dependency tree in a

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

unique order, rather than allowing each node to collect its dependents in different orders like the arc-standard or easy-first systems. Its transitions can be defined as follows:

$$\begin{aligned}
\textsc{Shift}_{HY} &= \textsc{Shift} \\
\textsc{Left-Arc}_{HY} &= \textsc{Left-Arc}; \textsc{Reduce} \\
\textsc{Right-Arc}_{HY} &= \textsc{Unshift}; \textsc{Right-Arc}; \textsc{Shift}; \textsc{Reduce}
\end{aligned}$$

Note that this parser creates leftward arcs between the first node in the buffer and the top node on the stack, just like arc-standard and arc-eager. Rightward arcs, however, are created by making the topmost stack node a dependent of the second topmost stack node, and removing the former from the stack.

Example 7

Sagae and Tsujii's (2008) arc-standard DAG parser performs bottom–up parsing without the common assumption that syntactic structures are represented as trees, allowing nodes to have multiple heads. It uses the following transitions:

$$\begin{aligned}
\textsc{Shift}_{DS} &= \textsc{Shift} \\
\textsc{Left-Reduce}_{DS} &= \textsc{Left-Arc}; \textsc{Reduce} \\
\textsc{Right-Reduce}_{DS} &= \textsc{Right-Arc}; \textsc{Shift}; \textsc{Reduce}; \textsc{Unshift} \\
\textsc{Left-Attach}_{DS} &= \textsc{Left-Arc}|_{\{(\sigma|i,j|\beta,A)\in C|\neg((i,j)\in A)\}} \\
\textsc{Right-Attach}_{DS} &= \textsc{Right-Arc}|_{\{(\sigma|i,j|\beta,A)\in C|\neg((j,i)\in A)\}}; \textsc{Unshift}
\end{aligned}$$

Whereas the first three transitions are exactly the same as in Nivre's (2008) arc-standard parser, the $\textsc{Left-Attach}_{DS}$ and $\textsc{Right-Attach}_{DS}$ transitions differ from $\textsc{Left-Reduce}_{DS}$ and $\textsc{Right-Reduce}_{DS}$ in that they do not remove the dependent of the new arc, thus allowing it to have additional incoming arcs. The restrictions on these transitions disallow the creation of both a left and a right arc between the same pair of nodes. Note that the class of dependency structures that can be output by this system does not exactly correspond to DAGs, however, because the system allows transition sequences that create dependency graphs with cycles. For example, starting from any configuration with at least two nodes on the stack and one node in the buffer and applying $\textsc{Right-Attach}_{DS}$, $\textsc{Right-Reduce}_{DS}$, $\textsc{Shift}_{DS}$, and $\textsc{Left-Attach}_{DS}$ gives rise to a cyclic structure.

Example 8

Sagae and Tsujii's (2008) arc-eager DAG parser allows nodes with multiple heads like the previous one but adds arcs in a strict left-to-right order like Nivre's (2003) arc-eager parser. The transition system can be defined as follows:

$$\begin{aligned}
\textsc{Shift}_{DE} &= \textsc{Shift} \\
\textsc{Reduce}_{DE} &= \textsc{Reduce}|_{H_\sigma(C)} \\
\textsc{Left-Arc}_{DE} &= \textsc{Left-Arc}|_{\{(\sigma|i,j|\beta,A)\in C|\neg((i,j)\in A)\}} \\
\textsc{Right-Arc}_{DE} &= \textsc{Right-Arc}|_{\{(\sigma|i,j|\beta,A)\in C|\neg((j,i)\in A)\}}
\end{aligned}$$

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Here the first two transitions are the same as in Nivre's (2003) arc-eager parser, whereas the LEFT-ARC$_{DE}$ and RIGHT-ARC$_{DE}$ transitions differ from their counterparts LEFT-ARC$_{AE}$ and RIGHT-ARC$_{AE}$ by not removing the dependent of the new arc. Like the previous arc-standard system, this system can produce cyclic dependency graphs. For example, starting from any configuration with at least one node in the stack and two nodes in the buffer and applying RIGHT-ARC$_{DE}$, SHIFT$_{DE}$, RIGHT-ARC$_{DE}$, REDUCE$_{DE}$, and LEFT-ARC$_{DE}$ creates a cycle of length 3.

Before we close this section we will briefly consider two transition systems that are not divisible. Attardi's (2006) non-projective parser extends the arc-standard system of Nivre (2004) with transitions that directly add non-projective arcs like the following:

$$\text{LEFT-ARC2} \quad (\sigma|i|k,j|\beta,A) \Rightarrow (\sigma|k,j|\beta,A \cup \{(j,i)\})$$
$$\text{RIGHT-ARC2} \quad (\sigma|i|k,j|\beta,A) \Rightarrow (\sigma|i,k|\beta,A \cup \{(i,j)\})$$

Nivre's (2009) non-projective parser constructs non-projective arcs in an indirect fashion by first swapping the order of two adjacent nodes on the stack:

$$\text{SWAP} \quad (\sigma|i|j,\beta,A) \Rightarrow (\sigma|j,i|\beta,A)$$

Neither of these systems can be formalized using only elementary transitions as defined herein, although they both represent straightforward extensions of the basic system.

3.2 Properties of Divisible Transition Systems

The elementary transition framework not only allows us to describe a wide range of transition-based parsers in a clear and concise way, but it can also easily be used to prove formal properties of transition systems. To do so, we consider the successions of transitions allowed by these algorithms, and break their transitions up into chains of elementary transitions.

Definition 8

Let $C_{0,m} = (c_0, c_1, \ldots, c_m)$ be a transition sequence for a sentence x under a transition system S. The **standard transition chain** associated with $C_{0,m}$ is the sequence of transitions $T_{0,m} = (t_1, t_2, \ldots, t_m)$ such that $t_i(c_{i-1}) = c_i$ for each $i \in [1, m]$.

Definition 9

Let $T_{0,m} = (t_1, t_2, \ldots, t_m)$ be the standard transition chain for a transition sequence $C_{0,m}$ under a divisible transition system S. Its associated **elementary transition chain** is the sequence of elementary transitions

$$E_{0,m} = (t_{1,1}, t_{1,2}, \ldots, t_{1,p_1}, t_{2,1}, t_{2,2}, \ldots, t_{2,p_2}, \ldots, t_{m,1}, t_{m,2}, \ldots, t_{m,p_m})$$

such that $t_1 = t_{1,1}|_{s_{1,1}}; t_{1,2}|_{s_{1,2}}; \ldots; t_{1,p_1}|_{s_{1,p_1}}, \ldots, t_m = t_{m,1}|_{s_{m,1}}; t_{m,2}|_{s_{m,2}}; \ldots; t_{m,p_m}|_{s_{m,p_m}}$ for some values of the restrictions $s_{1,1}, \ldots, s_{1,p_1}, \ldots, s_{m,1}, \ldots, s_{m,p_m}$.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Definition 10

Let $E_{0,m} = (e_1, e_2, \ldots, e_q)$ be the elementary transition chain for some transition sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$. Then:

- The **computation function** associated with $C_{0,m}$ is the function $e_1; e_2; \ldots; e_q$, resulting from composing the elementary transitions in the chain. Note that the same function could also be obtained from composing the transitions in the standard transition chain associated with $C_{0,m}$, and that this function will always map c_0 to c_m.
- The **elementary transition sequence** associated with $C_{0,m}$ is the sequence of configurations $C'_{0,m} = (c'_0, c'_1, \ldots, c'_q)$ such that $c'_0 = c_0$, and $c'_i = e_i(c'_{i-1})$ for all $i \in [1, q]$. Note that c'_q will always equal c_m. We will say that e_i is **applied** to the configuration c_{i-1} in $C'_{0,m}$.

After these preliminaries, we will now prove a number of results about divisible transition systems, first about the classes of dependency graphs that they can derive (Section 3.2.1) and secondly about termination and parsing complexity (Section 3.2.2).

3.2.1 Constraints on Dependency Graphs. Here we consider properties related to the graph constraints NO-COVERED-ROOTS, SINGLE-HEAD, ACYCLICITY, and PLANAR.

Proposition 1

If all elementary REDUCE transitions in the elementary transition chains under S are applied to configurations in $H_\sigma(C)$, then no dependency graph generated by S contains covered roots.

This property implies that algorithms where REDUCE transitions are restricted to the set $H_\sigma(C)$ always satisfy the NO-COVERED-ROOTS constraint. Note that this restriction may be expressed explicitly in the transition definitions (as in Example (4)), but it may also be implicit. For example, in Example (3), we defined $\text{LEFT-ARC}_{AS} = \text{LEFT-ARC}; \text{REDUCE}$. Although we did not explicitly write $\text{LEFT-ARC}; \text{REDUCE}_{|H_\sigma(C)}$, the LEFT-ARC transition always produces configurations that are trivially in $H_\sigma(C)$ (because the transition gives the topmost stack node a head), so the REDUCE transition in this algorithm is implicitly restricted to $H_\sigma(C)$. The same observation can be applied to subsequent properties.

Proof

To prove this proposition, we first make some simple observations about divisible transition systems that will be useful for this and subsequent proofs.

Lemma 1

In every configuration in an (elementary) transition sequence under a divisible transition system S, elements in the stack and buffer are ordered, that is, if the configuration is of the form $([s_1, \ldots, s_k], [b_1, \ldots, b_l], A)$, then we know that $s_1 < \ldots < s_k < b_1 < \ldots < b_l$. This can be easily seen by induction. It holds in initial configurations, because the stack is empty and the buffer is ordered, and all of the elementary transitions preserve the order of the nodes. Note that this lemma implies that a node cannot be in both the stack and the buffer of the same configuration.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Lemma 2

We will call $\Pi(c)$ the set of elements that are present either in the stack or in the buffer in a configuration c. Let $C'_{0,m} = (c'_0, c'_1, \ldots, c'_q)$ be an (elementary) transition sequence under a divisible transition system S. Then, we have that $\Pi(c'_q) \subseteq \Pi(c'_{q-1}) \subseteq \ldots \subseteq \Pi(c'_0) = \{1, \ldots, n\}$. This means that the set Π monotonically decreases in the course of an (elementary) transition sequence or, in plain language, that a node that is removed from the stack and buffer can never be placed back there by elementary transitions. This can be easily seen by observing that the transitions SHIFT, UNSHIFT, LEFT-ARC, and RIGHT-ARC leave the set Π unchanged, whereas the REDUCE transition removes one element from it by popping the stack.

Lemma 3

Let $E_{0,m} = (e_0, e_1, \ldots, e_q)$ and $C'_{0,m} = (c'_0, c'_1, \ldots, c'_q)$ be an elementary transition chain and its corresponding elementary transition sequence under a divisible transition system S. If $v \notin \Pi(c'_i)$ for some $v \in [0, n]$ and $i \in [0, q]$, then there exists some $j \in [0, i]$ such that $e_j =$ REDUCE and c'_{j-1} has v on the top of the stack. This amounts to saying that the only way an element can be removed from the set Π in a divisible system is by a REDUCE transition, as observed earlier. Thus, whenever a token v is not present in $\Pi(c'_i)$ for a given configuration c'_i, we can assume that it was previously popped by a REDUCE transition applied to a configuration that had v on the top of the stack.

With these observations, it is easy to show that if a graph generated by a transition sequence in a divisible transition system S has at least one covered root, then the transition sequence applies at least one REDUCE transition to a configuration that is not in $H_\sigma(C)$. Let G be a dependency graph in which the node j is a root, covered by an arc connecting the nodes i and k ($i < k$). If a transition sequence $C_{0,m}$ generates G, then it must apply a LEFT-ARC or RIGHT-ARC transition to a configuration having i at the top of the stack and k as the first element in the buffer, which is the only way of adding the arc involving i and k. By Lemma 1, we know that in that configuration c, $j \notin \Pi(c)$. By Lemma 3, we know that there must thus be a previous application of a REDUCE transition with j on the top of the stack. Because j is a root, by definition this configuration is not in $H_\sigma(C)$, and the proposition is proved. ■

Proposition 2

If all the elementary LEFT-ARC transitions in the elementary transition chains under S are applied to configurations in $\overline{H_\sigma(C)}$, and all the RIGHT-ARC elementary transitions are applied to configurations in $\overline{H_\beta(C)}$, then all the dependency graphs generated by S obey the SINGLE-HEAD constraint.

Proof

The proof of this proposition is straightforward. Because elementary transitions either leave the generated dependency graph as it is or add one dependency arc to it, an elementary transition sequence will generate a graph violating the SINGLE-HEAD constraint if and only if it contains a LEFT-ARC or RIGHT-ARC transition that adds an incoming arc to a node that already has a head in the graph. ■

Proposition 3

If all the elementary LEFT-ARC and RIGHT-ARC transitions in the elementary transition chains under S are applied to configurations $(\sigma|i, j|\beta, A) \in C$ where i and j belong to

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

different connected components of the undirected graph underlying A, then the undirected graphs underlying all the dependency graphs generated by S are acyclic (i.e., the dependency graphs generated by S have no undirected cycles). Note that this in turn implies ACYCLICITY.

Proof

Again, this proposition is straightforward, because a cycle can only be created in the undirected graph underlying the generated dependency graph if an arc is added between nodes that are already connected. ■

Proposition 4

All dependency graphs generated by a divisible system S are planar.

Proof

To prove this proposition, we observe that a graph is non-planar if and only if it contains two arcs (i,j) and (k,l) such that $\min(i,j) < \min(k,l) < \max(i,j) < \max(k,l)$. We can show that there is no elementary transition chain that creates such a pair of arcs.

- An elementary transition chain that first adds the arc (i,j) and later the arc (k,l) must apply a LEFT-ARC or RIGHT-ARC transition to a configuration having $\min(i,j)$ at the top of the stack and $\max(i,j)$ as the first element in the buffer, which is the only way of adding the first arc. By Lemma 1, we know that in that configuration c, $\min(k,l) \notin \Pi(c)$; and by Lemma 2, we know that $\min(k,l) \notin \Pi(c')$ for every subsequent configuration c' in the sequence. Given that an arc involving $\min(k,l)$ and $\max(k,l)$ can only be built from a configuration having $\min(k,l)$ in the stack, we conclude that after adding the arc (i,j) to the arc set, the parser will never be able to reach a configuration allowing it to add the arc (k,l).
- An elementary transition chain that first adds the arc (k,l) and later the arc (i,j) is not possible. The reasoning is analogous, but in this case $\max(i,j)$ is the node that gets removed from the set Π when the arc (k,l) is added, making it impossible to add the arc (i,j) afterwards.

Therefore, divisible systems can only generate planar dependency graphs. ■

The properties considered in this section can be used as a tool set for easily proving the soundness of transition systems with respect to different sets of dependency graphs, as well as for designing new transition systems. We exemplify the former in Example (9) and the latter in Section 3.3.

Example 9

Consider the transition set of the arc-eager parser in Example (4), repeated here for convenience:

$$\begin{aligned}
\text{SHIFT}_{AE} &= \text{SHIFT} \\
\text{REDUCE}_{AE} &= \text{REDUCE}|_{H_\sigma(C)} \\
\text{LEFT-ARC}_{AE} &= \text{LEFT-ARC}|_{\overline{H_\sigma(C)}}; \text{REDUCE} \\
\text{RIGHT-ARC}_{AE} &= \text{RIGHT-ARC}; \text{SHIFT}
\end{aligned}$$

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

We can easily conclude the following:

- The algorithm enforces the NO-COVERED-ROOTS constraint by Proposition 1, because REDUCE transitions are restricted to $H_\sigma(c)$.
- The algorithm enforces the SINGLE-HEAD constraint by Proposition 2, because LEFT-ARC elementary transitions are explicitly restricted to $\overline{H_\sigma(C)}$ and RIGHT-ARC transitions are implicitly restricted to $\overline{H_\beta(C)}$. (Trivially, none of the transitions can produce a configuration outside $\overline{H_\beta(C)}$.)
- The algorithm enforces the ACYCLICITY constraint by Proposition 3, because by construction none of the transitions can produce a configuration c where the first node in the buffer is connected to any node in $\Pi(c)$.
- The graphs it generates are planar by Proposition 4.
- The algorithm generates only projective dependency graphs, because the combination of PLANAR, ACYCLICITY, SINGLE-HEAD, and NO-COVERED-ROOTS implies PROJECTIVE.

3.2.2 Termination and Complexity. In general, there are two ways in which a transition-based parser may fail to parse a given input sentence. On the one hand, it may terminate in a non-terminal configuration where no transition can be applied. On the other hand, it may fail to terminate at all, because the system allows an infinite sequence of transitions. We say that a system is **robust** if it can never get stuck in a non-terminal configuration and **bounded** if it does not permit infinite loops.

Definition 11
A divisible transition system $S = (C, T, c_s, C_t)$ is **robust** if and only if, for every non-terminal configuration $c \in C \setminus C_t$, there is some transition $t \in T$ such that $t(c) \in C$.

Definition 12
A divisible transition system $S = (C, T, c_s, C_t)$ is **bounded** if and only if there exists no non-terminal configuration $c \in C \setminus C_t$ and (non-empty) sequence of transitions $t_1, \ldots t_k$ ($t_i \in T$) such that $t_1; \ldots; t_k(c) = c$.

In this section, we first provide sufficient conditions for robustness and boundedness and then go on to discuss the parsing complexity for a subset of divisible systems that are guaranteed to be robust and bounded.

Proposition 5
Let $S = (C, T, c_s, C_t)$ be a divisible transition system. If SHIFT $\in T$, then S is robust.

Proof
It is clear that SHIFT $\in T$ is sufficient for robustness, because it applies to every configuration that has a non-empty buffer β, which by definition includes every non-terminal configuration. ■

Thus, in order to guarantee robustness, it is enough that a divisible transition system includes the elementary SHIFT transition. This is the case for all the divisible systems

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

exemplified in Section 3.1. Before we go on to characterize bounded systems, it is convenient to introduce three auxiliary functions that characterize the effect a transition t has on an arbitrary configuration c:

- $A(t) = |A_{t(c)}| - |A_c|$
- $\Pi(t) = |\Pi(c)| - |\Pi(t(c))|$
- $\beta(t) = |\beta_c| - |\beta_{t(c)}|$

$A(t)$ is the *increase* in size of the arc set A, which is always non-negative as there are no elementary transitions that remove arcs. $\Pi(t)$ is the *decrease* in size of the set of nodes that are on the stack σ or in the buffer β, which is also non-negative as there are no elementary transitions that add new nodes. $\beta(t)$ is the *decrease* in the size of the buffer β, which can be negative as well as positive (or zero).

Proposition 6

Let $S = (C, T, c_s, C_t)$ be a divisible transition system. If every transition $t \in T$ is such that $A(t) > 0$ or $\Pi(t) > 0$ or $\beta(t) > 0$, then S is bounded.

Proof

To see why the disjunctive condition excludes looping transition sequences, consider an arbitrary configuration c and an arbitrary transition t for which the condition holds. If $A(t) > 0$ or $\Pi(t) > 0$, then c is clearly not reachable from $t(c)$, because there are no transitions that delete arcs (first case) or insert nodes (second case). If $A(t) = 0$ and $\Pi(t) = 0$, then $\beta(t) > 0$ and c could be reachable from $t(c)$ only if there is a transition t' such that $\beta(t') < 0$ (that is, a transition that puts nodes back in the buffer). But any such transition t' would have to have either $A(t) > 0$ or $\Pi(t) > 0$, which would again rule out the possibility of a loop. We may therefore conclude that there is no sequence of transitions $t_1, \ldots, t_2$ such that $t_1; \ldots; t_k(c) = c$ and, hence, that S is bounded. ■

Example 10

The condition of Proposition 6 does not hold for the elementary transition system, because $A(\text{UNSHIFT}) = 0$, $\Pi(\text{UNSHIFT}) = 0$, and $\beta(\text{UNSHIFT}) = -1$. In fact, this system is not bounded, because we can have an unbounded number of alternating SHIFT and UNSHIFT transitions without reaching a terminal configuration.

By contrast, the arc-eager system from Examples (4) and (9) is bounded, which can be seen by observing that $\beta(\text{SHIFT}_{AE}) = 1$, $\Pi(\text{REDUCE}_{AE}) = 1$, $A(\text{LEFT-ARC}_{AE}) = 1$, and $A(\text{LEFT-ARC}_{AE}) = 1$. The same reasoning can be applied to show that all the transition systems introduced in Examples (3)–(8) are bounded.

As already stated, the running time of a transition-based parser that only explores a constant number of transition sequences (such as a greedy deterministic parser or a beam-search parser with a constant-size beam) is given by an upper bound on the length of a transition sequence. To prove such bounds for divisible transition systems, we will first prove a linear bound on the number of arcs in planar graphs.

Lemma 4

A planar dependency graph with n nodes ($n > 1$) has no more than $4n - 6$ arcs.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Proof

For $n = 2$, we can trivially have at most two arcs, $(1, 2)$ and $(2, 1)$, and thus the lemma holds because $2 = 4 \cdot 2 - 6$. For the induction step, let $n > 2$. We will show that if the lemma holds for graphs with less than n nodes, then it also holds for graphs with n nodes.

To do so, we first give some preliminary definitions. We will say that the **length** of an arc (i, j) is $\ell(i, j) = \max(i, j) - \min(i, j)$. We will call the **domain** of an arc (i, j) the set $\delta(i, j) = \{\min(i, j), \min(i, j) + 1, \ldots, \max(i, j) - 1\}$. Note that the number of elements in the domain of an arc equals its length. We will say that an arc (i, j) **covers** an arc (k, l) if $(i, j) \neq (k, l)$ and $\min(i, j) \leq \min(k, l) < \max(k, l) \leq \max(i, j)$. Note that an arc (i, j) covers an arc (k, l) if and only if $\delta(k, l) \subset \delta(i, j)$, and a pair of distinct arcs (i, j) and (k, l) cross (as defined in Section 2.1) if and only if none of them covers the other and $\delta(k, l) \cap \delta(i, j) \neq \varnothing$. Thus, we conclude that a pair of distinct arcs that do not cross or cover each other have disjoint domains.

Let G be a planar dependency graph $G = (V = \{1, \ldots, n\}, A)$. Let $A_c = \{a_1, \ldots, a_m\}$ be the set of arcs in A with length strictly smaller than $n - 1$, and that are not covered by any arc in A with length strictly smaller than $n - 1$. By definition of A_c, we know that

$$\ell(a_i) < \ell(1, n) = n - 1 \tag{1}$$

On the other hand, because G is planar, a pair of arcs in A_c cannot cross each other. Furthermore, by definition of A_c, an arc in A_c cannot cover another arc in A_c. Therefore, the domains of $a_1, \ldots, a_m$ are disjoint subsets of $\{1, \ldots, n\}$, and thus

$$\ell(a_1) + \ldots + \ell(a_i) \leq \ell(1, n) = n - 1 \tag{2}$$

By definition of A_c, every arc in A is either (i) an arc of length at least $n - 1$ (i.e., $(1, n)$ or $(n, 1)$), or (ii) an arc $a_i \in A_c$, or (iii) an arc covered by some arc $a_i \in A_c$. For each given $i \in \{1, \ldots, m\}$, the arcs of types (ii) and (iii) form a subgraph of G with $\ell(a_i) + 1$ nodes. Because $\ell(a_i) + 1 < n$, we can apply the induction hypothesis to conclude that there are at most $4(\ell(a_i) + 1) - 6$ arcs of this type for each value of i. Combining this with Equations (1) and (2), we conclude that the total amount of arcs in A is bounded by

$$\max_{\substack{a_1, \ldots, a_m \\ \text{s.t. } \ell(a_i) < n-1 \\ \text{and } \sum_{i=1}^{m} \ell(a_i) \leq n-1}} 2 + \sum_{i=1}^{m} [4(\ell(a_i) + 1) - 6]$$

$$= 2 + \max_{\substack{a_1, \ldots, a_m \\ \text{s.t. } \ell(a_i) < n-1 \\ \text{and } \sum_{i=1}^{m} \ell(a_i) \leq n-1}} (-2m + 4 \sum_{i=1}^{m} \ell(a_i))$$

It is easy to see that the expression is maximized for $m = 2$, and in that case the value of the expression is bounded by $2 - 2 \cdot 2 + 4(n - 1) = 4n - 6$. This proves the induction step and thus concludes the proof of Lemma 4. ■

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Thanks to the result in Lemma 4, we can now proceed to prove bounds on the length of transition sequences in divisible systems that are guaranteed to be robust and bounded, that is, systems that satisfy the conditions of Propositions 5 and 6. We call such systems *efficient* divisible transition systems.

Definition 13

A divisible transition system $S = (C, T, c_s, C_t)$ is **efficient** if and only if SHIFT $\in T$ and, for every $t \in T$, $A(t) > 0$ or $\Pi(t) > 0$ or $\beta(t) > 0$.

We give three increasingly tight bounds for (i) arbitrary efficient divisible transition systems, (ii) systems that in addition have a constant bound on the growth of the buffer, and (iii) systems that have a constant bound on the number of elementary transitions that a composite transition can contain.

Proposition 7

Let $S = (C, T, c_s, C_t)$ be an efficient divisible transition system. Then the length of a transition sequence for a sentence x of length n in S is $O(n^2)$.

Proof

Consider an arbitrary transition sequence $C_{0,m} = (c_s(x), \ldots, c_m)$ in S for a sentence x of length n and the corresponding transition chain $T_{1,m} = (t_1, \ldots, t_m)$. The following must hold:

- The number of transitions t in $T_{1,m}$ for which $A(t) > 0$ is bounded by the maximum number of arcs in a planar dependency graph, which is $4n - 6$ (by Lemma 4).
- The number of transitions t in $T_{1,m}$ for which $\Pi(t) > 0$ is according to Lemma 2 bounded by the number of nodes in the initial configuration $c_s(x)$, which is n.
- The longest contiguous subsequence $T_{i,k} = (t_i, \ldots, t_k)$ of $T_{1,m}$ such that all transitions $t_j \in T_{i,k}$ have $A(t) = 0$, $\Pi(t) = 0$ and $\beta(t) > 0$ is bounded by the maximum size of the buffer, which again according to Lemma 2 is bounded by the number n of nodes in the initial configuration $c_s(c)$.

Hence, $T_{1,m}$ can contain at most $O(n)$ transitions of the first two types and at most $O(n^2)$ transitions of the third type, because there are at most $O(n)$ transitions that increase the size of the buffer. ■

Proposition 8

Let $S = (C, T, c_s, C_t)$ be an efficient divisible transition system such that, for every transition $t \in T$, $\beta(t) > k$ for some constant k. Then the length of every transition sequence for a sentence x of length n in S is $O(n)$.

Proof

This follows from the same kind of considerations as in the proof of Proposition 7 together with the observation that the total number of transitions t for which $A(t) = 0$,

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

$\Pi(t) = 0$, and $\beta(t) > 0$ is now bounded by kn instead of n^2, because each of the $O(n)$ transitions that may increase the size of the buffer can only do so by at most k. ■

Proposition 9

Let $S = (C, T, c_s, C_t)$ be an efficient divisible transition system such that, for every transition $t = t_1|_{s_1}; \ldots; t_m|_{s_m}$ ($t \in T$, $t_i \in T_e$), $m \leq k$ for some constant k. The length of every elementary transition sequence for a sentence x of length n in S is $O(n)$.

Proof

This follows from Proposition 8 together with the constant bound on the number of elementary transitions in a composite transition. ■

Example 11

All the systems defined in Section 3.1 satisfy the condition of Proposition 8 and therefore have a linear bound on the length of their transition sequences. In addition, all the systems except the easy-first parser satisfy the condition of Proposition 9 and therefore also have a linear bound on the number of elementary transitions. To see why this fails for the easy-first parser, note that the number of elementary transitions in $\text{ATTACH-LEFT}(i)_{EF}$ and $\text{ATTACH-RIGHT}(i)_{EF}$ depends on i, which can grow with the size of the sentence. Nevertheless, $\beta(\text{ATTACH-LEFT}(i)_{EF}) = \beta(\text{ATTACH-RIGHT}(i)_{EF}) = 1$ (for all values of i), which guarantees the linear bound on composite transitions.

3.3 Planar Dependency Parsing

So far in this section, we have shown how a number of well-known transition systems from the literature can be formulated and studied as divisible transition systems, that is, as restrictions of the same generic system based on five elementary transitions. In this section, we show how this formulation can also be used to define a novel algorithm. Specifically, we can obtain a transition system that will be able to parse any planar dependency graph (regardless of projectivity) if we use all elementary transitions except UNSHIFT directly as the transitions of the system. On top of this system, we can use Propositions 1–4 to add optional restrictions to the system in order to enforce the SINGLE-HEAD, ACYCLICITY, and NO-COVERED-ROOTS constraints. In addition, we can use Propositions 5–9 to show that there is a linear bound on the length of elementary transition sequences in this system. In this way, we obtain an efficient parser for planar dependency graphs, optionally restricted to trees, which is a novel contribution in itself. More importantly, however, we will show in Section 5 how this system can be generalized to a system capable of handling non-planar, hence non-projective, dependency trees using the concept of multiplanarity (to be introduced in Section 4).

First, we define a **planar** transition system as the divisible transition system S_P having the following transitions:

$$
\begin{aligned}
\text{SHIFT}_P &= \text{SHIFT} \\
\text{REDUCE}_P &= \text{REDUCE} \\
\text{LEFT-ARC}_P &= \text{LEFT-ARC} \\
\text{RIGHT-ARC}_P &= \text{RIGHT-ARC}
\end{aligned}
$$

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

3.3.1 Correctness. This transition system can parse all the planar dependency graphs. To prove its correctness, we must show soundness (all the graphs produced by the system are planar) and completeness (all the planar graphs can be obtained by the system). Soundness is trivial given Property 4, so we only need to prove completeness. To do so, we prove the stronger claim in Lemma 5.

Lemma 5

Let $G = (V, A)$ be a planar dependency graph for a sentence $w_1 \dots w_n$. Then there is a transition sequence in S_P ending in a terminal configuration of the form $(\sigma, [\,], A)$ such that all the nodes that are not covered by any dependency arc in A are in σ.

Proof

To prove this lemma, we proceed by induction on the length n of the sentence. In the case where $n = 1$, the only possible planar dependency graph is the graph $G_0 = (\{1\}, \varnothing)$ with a single node and no arcs. It is easy to see that the transition sequence that applies a single SHIFT transition meets the required conditions, because it ends in a terminal configuration $([1], [\,], \varnothing)$.

For the induction step, we assume that the lemma holds for sentences of length n and prove that it then also holds for sentences of length $n + 1$, for any $n \geq 1$. Let $G_{n+1} = (V_{n+1}, A_{n+1})$ be a planar dependency graph for a sentence $w_1 \dots w_{n+1}$. We denote by L_{n+1} the set of arcs

$$L_{n+1} = \{(n+1, i) \in A_{n+1}\} \cup \{(j, n+1) \in A_{n+1}\}$$

that is, the set of incoming and outgoing arcs from the node $n + 1$ in G_{n+1}, and we denote by G_n the graph

$$G_n = (V_n = V_{n+1} \setminus \{n+1\}, A_n = A_{n+1} \setminus L_{n+1})$$

that is, the graph obtained by removing the node $n + 1$ and all its incoming and outgoing arcs from G_{n+1}. By the induction hypothesis, there exists a transition sequence C_n whose final configuration is of the form $(\sigma_n, [\,], A_n)$, such that σ_n contains all the nodes that are not covered by any dependency arc in A_n. From this transition sequence C_n, we will obtain a transition sequence C_{n+1} meeting the conditions asserted by the lemma for the graph G_{n+1}. To do so, we first observe that the planarity of the graph G_{n+1} implies that the left endpoints of the arcs in L_{n+1} cannot be covered by any arc in A_n, because this would mean that the arc in L_{n+1} and the covering arc would cross. Therefore, by the induction hypothesis, we know that all the left endpoints of the arcs in L_{n+1} are in σ_n. Thus, if the left endpoints of the arcs in L_{n+1} are $i_1, i_2, \dots, i_e$; then the stack σ_n (which is ordered, by Lemma 1) is of the form

$$\sigma_n = [s_1, \dots, s_{z_1} = i_1, \dots, s_{z_2} = i_2, \dots, s_{z_e} = i_e, \dots, s_m]$$

With this in mind, we can obtain the transition sequence C_{n+1} from C_n by adding the following extra transitions at the end of its associated transition chain:

$$\text{REDUCE}^{m-z_e}; arcs(i_e); \text{REDUCE}^{z_e - z_{e-1}}; arcs(i_{e-1}); \dots; \text{REDUCE}^{z_2 - z_1}; arcs(i_1); \text{SHIFT}$$

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

where we use the notation *arcs*(i) as shorthand for:

- LEFT-ARC, if $(n+1, i) \in L_{n+1} \wedge (i, n+1) \notin L_{n+1}$,
- RIGHT-ARC, if $(n+1, i) \notin L_{n+1} \wedge (i, n+1) \in L_{n+1}$,
- LEFT-ARC; RIGHT-ARC, if $(n+1, i) \in L_{n+1} \wedge (i, n+1) \in L_{n+1}$.

The final configuration of the transition sequence obtained by applying these transitions at the end of C_n is of the form (σ, β, A), where:

- $\beta = [\,]$; since the nodes $1, \ldots, n$ are removed from the buffer by C_n, and $n+1$ is removed by the extra SHIFT transition,
- $A = A_{n+1}$, because $A_{n+1} = A_n \cup L_{n+1}$, the arcs in A_n are added to the set by C_n, and all the arcs in L_{n+1} are added by *arcs*(i_e), ..., *arcs*(i_1),
- All the nodes that are not covered by arcs in A_{n+1} are in σ, because they were in σ_n (a node not covered by arcs in A_{n+1} is trivially not covered by arcs in A_n) and the REDUCE transitions applied after C_n only remove nodes to the right of i_1, which are covered by the arc $(n+1, i_1)$ or $(i_1, n+1)$.[6]

This proves the induction step for Lemma 5, and thus correctness is proved. ■

3.3.2 Constraints on Planar Dependency Parsing. As we have just proved, the transition system S_P is able to parse all planar dependency graphs. In many practical applications, however, it is convenient to exclude some subset of those graphs, for example, those that have cycles or more than one head per node. The results obtained in Section 3.2 can be used to easily add common constraints to the planar parser. The constraints can be added individually or jointly, so that we can obtain a variant of the planar parser with the SINGLE-HEAD, ACYCLICITY, and NO-COVERED-ROOTS constraint, or with any combination of them.

Single-Head Constraint. To add the SINGLE-HEAD constraint to the S_P transition system, we restrict the LEFT-ARC$_P$ transition to $\overline{H_\sigma(C)}$, and the RIGHT-ARC$_P$ transition to $\overline{H_\beta(C)}$:

$$\begin{aligned} \text{LEFT-ARC}_{P-Sh} &= \text{LEFT-ARC}|_{\overline{H_\sigma(C)}} \\ \text{RIGHT-ARC}_{P-Sh} &= \text{RIGHT-ARC}|_{\overline{H_\beta(C)}} \end{aligned}$$

The soundness of this variant for the set of planar dependency graphs that meet the SINGLE-HEAD constraint is trivially given by Proposition 2. Completeness is also straightforward, because, as discussed in Proposition 2, applying a LEFT-ARC transition to a configuration of $H_\sigma(C)$ or a RIGHT-ARC transition to a configuration of $H_\beta(C)$ will *always* generate a graph violating the SINGLE-HEAD constraint. Therefore, any graph that meets the SINGLE-HEAD constraint and can be obtained using the S_P transition system (which has been proven complete) can also be generated by this one.

6 This assumes that at least one arc was created to or from node $n+1$ (i.e., that $e > 0$). In the case where $e = 0$, it is trivial to show that all nodes not covered by arcs in G_{n+1} are in σ, because in that case no REDUCE transitions are applied at all after C_n.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Acyclicity Constraint. Analogously to the case for the SINGLE-HEAD constraint, we can add the ACYCLICITY constraint to the S_P transition system by applying Proposition 3. To do so, we restrict the LEFT-ARC$_P$ and RIGHT-ARC$_P$ transitions as follows:

$$\begin{aligned} \text{LEFT-ARC}_{P-Ac} &= \text{LEFT-ARC}|_{\{(\sigma|i,j|\beta,A)\in C|\neg(i\leftrightarrow^* j\in A)\}} \\ \text{RIGHT-ARC}_{P-Ac} &= \text{RIGHT-ARC}|_{\{(\sigma|i,j|\beta,A)\in C|\neg(i\leftrightarrow^* j\in A)\}} \end{aligned}$$

The soundness of this variant for the set of acyclic planar dependency graphs is trivially implied by Proposition 3. This variant is not complete for acyclic planar dependency graphs, because it actually enforces a stronger variant of ACYCLICITY, namely, it will only accept dependency graphs that have no undirected cycles. We can combine this acyclicity check with the SINGLE-HEAD constraint by intersecting the restrictions:

$$\begin{aligned} \text{LEFT-ARC}_{P-ShAc} &= \text{LEFT-ARC}|_{\overline{H_\sigma(C)}\cap\{(\sigma|i,j|\beta,A)\in C|\neg(i\leftrightarrow^* j\in A)\}} \\ \text{RIGHT-ARC}_{P-ShAc} &= \text{RIGHT-ARC}|_{\overline{H_\beta(C)}\cap\{(\sigma|i,j|\beta,A)\in C|\neg(i\leftrightarrow^* j\in A)\}} \end{aligned}$$

We then obtain a parser that is sound and complete for the set of planar dependency graphs that meet the SINGLE-HEAD and ACYCLICITY constraints. The reason is that, under the SINGLE-HEAD constraint, standard ACYCLICITY and undirected acyclicity are equivalent, because every undirected cycle is also a directed cycle. If we need a parser that enforces only directed ACYCLICITY but allows nodes with multiple heads, this can also be achieved. Instead of checking $\neg(i \leftrightarrow^* j \in A)$, the restrictions must check that the arc does not create a directed cycle (that is, $\neg(i \rightarrow^* j \in A)$ for LEFT-ARC and $\neg(j \rightarrow^* i \in A)$ for RIGHT-ARC). Although the check for undirected cycles can be implemented in constant time if the parser implementation keeps track of the connected component of each node in A, the check for directed cycles is more computationally costly, however.[7]

No-Covered-Roots Constraint. Similarly to the other constraints, we can add the NO-COVERED-ROOTS constraint to S_P by applying Proposition 1. To do so, we restrict the REDUCE$_P$ transition as follows:

$$\text{REDUCE}_{P-Nc} = \text{REDUCE}|_{H_\sigma(C)}$$

The soundness of the resulting parser with respect to planar dependency graphs complying with the NO-COVERED-ROOTS constraint is directly given by Proposition 1. To prove completeness, we observe that the transition sequences that we build for each graph in the proof of Lemma 5 only reduce nodes that are then covered by an arc. Therefore, given a graph G that satisfies the NO-COVERED-ROOTS constraint, we know that the transition sequence built as in that proof will never reduce a root node. Therefore, all of its REDUCE transitions will be applied to configurations in $H_\sigma(C)$ and, hence, that same transition sequence will also parse G in this variant of the transition system, which proves completeness. The NO-COVERED-ROOTS restriction can be combined with any combination of the other two restrictions. Note that the result of applying the NO-COVERED-ROOTS restriction alone is equivalent to the arc-eager parser by Sagae and

7 Strictly speaking, for undirected cycles using the techniques of path compression and union by rank for disjoint sets, the amortized time per operation is $O(\alpha(n))$, where n is the number of nodes and $\alpha(n)$ is the inverse of the Ackermann function, which means that $\alpha(n)$ is less than 5 for all remotely practical values of n and is effectively a small constant (Cormen, Leiserson, and Rivest 1990).

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Tsujii (2008). If the SINGLE-HEAD, ACYCLICITY, and NO-COVERED-ROOTS restrictions are applied at the same time, together with the PLANAR constraint that is implicit in the algorithm itself, we obtain a projective parser different from the projective parsers described in Section 3.1.

3.3.3 Complexity of Planar Dependency Parsing. To study the runtime complexity of the planar parser, it suffices to observe that the planar transition system in any of its variants (with or without constraints) satisfies the following:

- It is efficient, by Definition 13, because it contains the elementary SHIFT transition and $\beta(\text{SHIFT}_P) = 1$, $\Pi(\text{REDUCE}_P) = 1$, $A(\text{LEFT-ARC}_P) = 1$, and $A(\text{RIGHT-ARC}_P) = 1$. This implies that the system is robust (by Proposition 5) and bounded (by Proposition 6).
- The length of every transition sequence in the planar parser is $O(n)$ (by Proposition 8) and the same holds for elementary transition sequences (by Proposition 9).

We also note that each transition in any of the variants of the planar parser can be executed in constant time, because their execution only requires us to keep track of a constant number of nodes at the top of the stack and at the beginning of the buffer. The exception is the check for ACYCLICITY if this restriction is required. As explained in Section 3.3.2, however, this can be implemented in constant time if the SINGLE-HEAD constraint is present by keeping track of the undirected connected component of each node in the generated dependency graph. Therefore, as explained in Section 2.3, the complexity of the planar parser with beam search is $O(n)$, both for the unrestricted version and for the variant that enforces the SINGLE-HEAD and ACYCLICITY constraints.

3.4 Beyond Planarity

Although the divisible transition system framework introduced in Section 3 can be used to represent and study a wide range of parsers, we have seen by Proposition 4 that it is limited to parsers that generate planar dependency graphs. As already noted, planarity is a very mild relaxation of the better known projectivity constraint, the only difference being that planarity allows graphs with covered roots (see Definition 2), and studies of natural language treebanks have shown the vast majority of non-projective structures to be non-planar as well (Kuhlmann and Nivre 2006; Havelka 2007).[8] Therefore, being able to parse planar dependency graphs only provides a modest improvement in practical coverage with respect to projective parsing. To increase this coverage further, we need to be able to handle dependency graphs with crossing arcs.

To be able to build such graphs, several stack-based transition systems have been proposed in the literature that introduce extra flexibility by allowing actions that fall outside the divisible transition system framework, like the systems by Attardi (2006) and Nivre (2009) shown at the end of Section 3.1. Because these parsers use diverse strategies to support different subsets of non-planar structures—allowing arcs to be built to or from nodes deep in the stack in the case of Attardi (2006), adding transitions able to reorder stack nodes in the case of (Nivre 2009)—it seems unlikely that a simple

8 This is true in particular if dependency graphs are restricted to trees that have their roots at the periphery, as in Figure 1, in which case the two notions become equivalent.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

extension of the framework can encompass all of them in a natural way. We can, however, extend the framework individually for each approach by adding the respective new transitions as elementary transitions, but the details and properties of each of these extensions fall outside the scope of this article.

Instead, in the next sections we will focus on introducing a different extension of the framework that is achieved by adding additional stacks, giving support to a generalization of the planar transition system described in Section 3.3 that can parse a large set of non-planar graphs.

4. Multiplanar Dependency Graphs

Because it has been shown that exact parsing becomes computationally intractable when arbitrary non-projective dependency graphs are allowed (McDonald and Satta 2007), a substantial amount of research in recent years has been devoted to finding a superset of projective dependency graphs that is rich enough to cover the non-projective phenomena found in natural language and restricted enough to allow for simple and efficient parsing, that is, a suitable set of **mildly non-projective dependency structures**. To this end, different sets of dependency trees have been proposed, such as trees with bounded arc degree (Nivre 2006a, 2007), well-nested trees with bounded gap degree (Kuhlmann and Nivre 2006; Kuhlmann and Möhl 2007), mildly ill-nested trees with bounded gap degree (Gómez-Rodríguez, Weir, and Carroll 2009), or the operationally defined set of trees parsed by the transition system of Attardi (2006).

In the same vein, a straightforward way to relax the planarity constraint to obtain richer sets of non-projective dependency graphs is the notion of **multiplanarity**, or k-planarity, originally introduced by Yli-Jyrä (2003). Quite simply, a dependency graph is said to be k-planar if it can be decomposed into k planar dependency graphs.

Definition 14

A dependency graph $G = (V, A)$ is ***k*-planar** if there exist planar dependency graphs $G_1 = (V, A_1), \ldots, G_k = (V, A_k)$ (called **planes**) such that $A = A_1 \cup \cdots \cup A_k$.

Intuitively, we can associate planes with colors and say that a dependency graph G is k-planar if it is possible to assign one of k colors to each of its arcs in such a way that arcs with the same color do not cross. Note that there may be multiple ways of dividing a k-planar graph into planes, as shown in the example of Figure 4. Therefore,

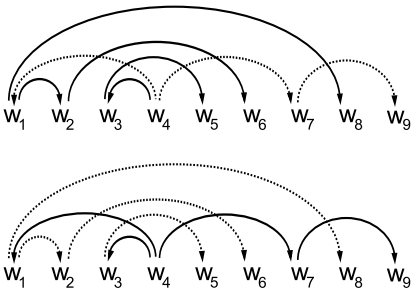

Figure 4
A 2-planar dependency structure with two different ways of distributing its arcs into two planes (represented by solid and dotted lines).

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

1-planarity is equivalent to planarity, and increasing values of k yield increasingly rich sets of dependency graphs.

The notion of k-planarity has so far played a marginal role in the dependency parsing literature, because little was known about the properties of these structures. No algorithm was known to determine whether a given graph was k-planar, and no efficient parsing algorithm existed for k-planar dependency structures. In this article, we overcome these problems. In the remainder of this section, we present a procedure to determine the minimum value of k for which a given structure is k-planar, and we use it to show that the overwhelming majority of sentences in a number of dependency treebanks have a tree that is at most 2-planar. In Section 5, we then show how the 1-planar dependency parser described in Section 3.3 can be generalized to handle k-planar dependency graphs by introducing additional stacks. In particular, we present a linear-time transition-based parser that is provably correct for 2-planar dependency trees.[9]

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

4.1 Test for Multiplanarity

In order for a constraint on non-projective dependency structures to be useful for practical parsing, it must provide 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, Kuhlmann, and Möhl 2005), leading to an interest in parsers for these kinds of structures (Gómez-Rodríguez, Weir, and Carroll 2009; Kuhlmann and Satta 2009). No similar analysis has been performed for k-planar structures, however. Yli-Jyrä (2003) does provide evidence that all except two structures in the Danish Dependency Treebank (Kromann 2003) are at most 3-planar, but his analysis is based on constraints that restrict the possible ways of assigning planes to dependency arcs, and he is not guaranteed to find the minimal number k for which a given structure is k-planar.

Here we provide a procedure for finding the minimal natural number k such that a dependency graph is k-planar and use it to show that the vast majority of sentences in a number of dependency treebanks are 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 $G = (V, A)$ is k-planar, for a given value of k, to a standard graph coloring problem. To do this, we first consider the following undirected graph:

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

Note that we can formally say that two arcs (i, j) and (k, l) in a dependency graph G such that $i < k$ are crossing arcs if and only if $\min(i, j) < \min(k, l) < \max(i, j) < \max(k, l)$. These are the pairs of arcs that were forbidden in the planarity constraint introduced in Definition 2. The graph $U(G)$, which we call the **crossings graph** of G, has one node corresponding to each arc in the dependency graph G, with an undirected edge between

9 The test for multiplanarity and the 2-planar parser have previously been described in Gómez-Rodríguez and Nivre (2010).

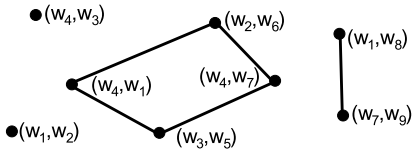

Figure 5
The crossings graph corresponding to the dependency structure of Figure 4.

two nodes if they correspond to crossing arcs in G. Figure 5 shows the crossings graph of the 2-planar structure in Figure 4.

As noted earlier, a dependency graph G is k-planar if each of its arcs can be assigned one of k colors in such a way that two arcs that cross each other are not assigned the same color. In terms of the crossings graph, because each arc in G corresponds to a node in $U(G)$ and each pair of crossing arcs in G corresponds to an edge in $U(G)$, this is equivalent to saying that G is k-planar if each of the *nodes* of $U(G)$ can be assigned one of k colors such that no two neighbors have the same color. This amounts to solving the well-known k-coloring problem for $U(G)$.

For $k = 1$ the problem is trivial: A graph is 1-colorable only if it has no edges. This corresponds to a dependency graph being planar only if it does not have crossing arcs. For $k = 2$, the problem is equivalent to determining whether the graph is bipartite, and it can be solved in time linear in the size of the graph by simple breadth-first search. Given any undirected graph $U = (V, E)$, we pick an arbitrary node v and give it one of two colors. This forces us to give the other color to all its neighbors, the first color to the neighbors' neighbors, 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 colors to the same node, the graph is not 2-colorable. Otherwise, we have obtained a 2-coloring 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-coloring of the whole graph if it exists. Because 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 arcs to determine if they cross, and cannot contain more than n^2 edges, meaning 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-coloring problem is known to be NP-complete (Karp 1972).[10] We have found this not to be a problem in practice when using it to measure multiplanarity in natural language treebanks, because 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. If we have a valid coloring for all the cycles in the graph, the rest of the nodes can be safely colored by breadth-first search as in the $k = 2$ case. Given that non-projective sentences in natural language tend to have a small proportion of non-projective arcs, the connected components of their crossings graphs tend to be very small and with few cycles, and k-colorings for them can quickly be found by brute-force search.

10 Note that this does not necessarily imply that the problem of determining whether a graph is k-planar is also NP-complete, because there might be polynomial algorithms that solve it without involving a reduction to the k-coloring problem.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

4.2 Treebank Coverage

To find out the prevalence of k-planar trees in natural language treebanks for various values of k, we applied the technique described in the previous section to all the trees in the training set for eight languages in the CoNLL-X shared task on dependency parsing (Buchholz and Marsi 2006): 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, Hall, and Nivre 2005), and Turkish (Atalay, Oflazer, and Say 2003; Oflazer et al. 2003). The results are 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. In comparison to well-nestedness, it is worth noting that no efficient parser has been proposed that is able to handle *all* well-nested dependency trees, only well-nested trees with bounded gap degree, which reduces coverage (Kuhlmann and Möhl 2007; Gómez-Rodríguez, Carroll, and Weir 2011). As will be seen in the next section, the class of 2-planar dependency trees not only has good coverage of linguistic structures in existing treebanks but is also parsable with a linear-time transition-based parser, making it a theoretically as well as practically interesting subclass of non-projective dependency trees.

5. Multiplanar Dependency Parsing

The divisible transition system framework introduced in Section 3 can be generalized to support k-planar dependency graphs by using k stacks instead of only one and applying the SHIFT and UNSHIFT elementary transitions to all of them at the same time, whereas REDUCE, LEFT-ARC, and RIGHT-ARC only affect one stack at a time. The stack on which these latter transitions are applied is decided by an extra elementary transition, called SWITCH, which cycles through the k stacks selecting one of them as the *active* stack.

This generalization has the property that the set of arcs created in the context of each individual stack will be planar, but pairs of arcs created in different stacks are allowed to cross. In this way, a k-stack parser will be able to build a k-planar dependency forest by using each of the stacks to construct one of its k planes.

Although the general case of k-planar dependency parsing is interesting as a theoretical construction, we will limit ourselves in this article to the 2-planar case and show

Table 1
Proportion of dependency trees classified by projectivity, planarity, k-planarity, and ill-nestedness in a sample of treebanks.

Language	Trees	Non-Projective	Not Planar	Not 2-Pl.	Not 3-Pl.	Not 4-Pl.	Ill-nested
Arabic	2,995	205 (6.84%)	158 (5.28%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	1 (0.03%)
Czech	87,889	20,353 (23.16%)	16,660 (18.96%)	82 (0.09%)	0 (0.00%)	0 (0.00%)	96 (0.11%)
Danish	5,512	853 (15.48%)	827 (15.00%)	1 (0.02%)	1 (0.02%)	0 (0.00%)	6 (0.11%)
Dutch	13,349	4,865 (36.44%)	4,115 (30.83%)	162 (1.21%)	1 (0.01%)	0 (0.00%)	15 (0.11%)
German	39,573	10,927 (27.61%)	10,908 (27.56%)	671 (1.70%)	0 (0.00%)	0 (0.00%)	419 (1.06%)
Portuguese	9,071	1,718 (18.94%)	1,713 (18.88%)	8 (0.09%)	0 (0.00%)	0 (0.00%)	7 (0.08%)
Swedish	6,159	293 (4.76%)	280 (4.55%)	5 (0.08%)	0 (0.00%)	0 (0.00%)	14 (0.23%)
Turkish	5,510	657 (11.92%)	657 (11.92%)	10 (0.18%)	0 (0.00%)	0 (0.00%)	20 (0.36%)

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

how a system built by generalizing the planar parser defined in Section 3.3 to use two stacks instead of one can yield an efficient parser for 2-planar dependency graphs, in particular 2-planar trees. As we saw in Section 4.2, this class of structures gives almost perfect coverage in existing treebanks, and we will therefore leave the exploration of k-planar dependency parsing for k higher than 2 as future work.

Note that, because we are only interested in defining a single transition system using the multi-stack generalization of the divisible transition system framework, we will introduce the system directly as a generalization of the planar transition system, rather than showing the step-by-step details of how the general framework is first extended to multiple stacks (as outlined earlier) and then defining the new system on top of the extended framework.

5.1 2-Planar Dependency Parsing

The 2-planar transition system S_{2P} has configurations of the form $(\sigma^1, \sigma^2, B, A)$, where we call σ^1 the **active stack** and σ^2 the **inactive stack**. Because the system uses two stacks rather than one, it does not conform to the standard definition of a stack-based transition system given in Section 2.2, but it behaves analogously. In this case, the initialization function is $c_s(w_1, \ldots, w_n) = ([\,], [\,], [1, \ldots, n], \varnothing)$ and the set of terminal configurations is $C_t = \{c \mid c = (\sigma^1, \sigma^2, [\,], A) \text{ for any } \sigma^1, \sigma^2, A\}$. The transitions of this system are the following:

$$
\begin{aligned}
\textsc{Shift}_{2P} &= (\sigma^1, \sigma^2, i|B, A) \Rightarrow (\sigma^1|i, \sigma^2|i, B, A) \\
\textsc{Reduce}_{2P} &= (\sigma^1|i, \sigma^2, B, A) \Rightarrow (\sigma^1, \sigma^2, B, A) \\
\textsc{Left-Arc}_{2P} &= (\sigma^1|i, \sigma^2, j|B, A) \Rightarrow (\sigma^1|i, \sigma^2, j|B, A \cup \{(j, i)\}) \\
\textsc{Right-Arc}_{2P} &= (\sigma^1|i, \sigma^2, j|B, A) \Rightarrow (\sigma^1|i, \sigma^2, j|B, A \cup \{(i, j)\}) \\
\textsc{Switch}_{2P} &= (\sigma^1, \sigma^2, B, A) \Rightarrow (\sigma^2, \sigma^1, B, A)
\end{aligned}
$$

The $\textsc{Shift}_{2P}$ transition pops the first (leftmost) word in the buffer, and pushes it to *both* stacks. The $\textsc{Left-Arc}_{2P}$ transition adds an arc from the first word in the buffer to the top of the *active* stack. The $\textsc{Right-Arc}_{2P}$ transition adds an arc from the top of the *active* stack to the first word in the buffer. The $\textsc{Reduce}_{2P}$ transition pops the top word from the *active* stack, implying that we have added all arcs to or from it on the plane tied to that stack. The $\textsc{Switch}_{2P}$ transition, finally, makes the active stack inactive and vice versa, changing the plane the parser is working with. In order to exemplify how this system can parse non-planar dependency graphs, Figure 6 shows a transition sequence for the tree in Figure 1.

5.1.1 Correctness of 2-Planar Dependency Parsing. To show that this transition system is correct for the set of 2-planar dependency graphs, we need to prove that it is sound (every graph produced by the system is 2-planar) and complete (all 2-planar graphs can be derived from the system). We do this by proving two corresponding lemmas, the second of which is a stronger claim than mere completeness.

Lemma 6

The system S_{2P} is sound for the set of 2-planar dependency graphs.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

		(	[ ],	[ ],	$[1,\ldots,9]$,	$\varnothing$)
SHIFT	$\Rightarrow$	(	[1],	[1],	$[2,\ldots,9]$,	$\varnothing$)
SHIFT	$\Rightarrow$	(	$[1,2]$,	$[1,2]$,	$[3,\ldots,9]$,	$\varnothing$)
RIGHT-ARC	$\Rightarrow$	(	$[1,2]$,	$[1,2]$,	$[3,\ldots,9]$,	$A_1 = \{(2,3)\}$)
SHIFT	$\Rightarrow$	(	$[1,2,3]$,	$[1,2,3]$,	$[4,\ldots,9]$,	A_1)
REDUCE	$\Rightarrow$	(	$[1,2]$,	$[1,2,3]$,	$[4,\ldots,9]$,	A_1)
REDUCE	$\Rightarrow$	(	[1],	$[1,2,3]$,	$[4,\ldots,9]$,	A_1)
RIGHT-ARC	$\Rightarrow$	(	[1],	$[1,2,3]$,	$[4,\ldots,9]$,	$A_2 = A_1 \cup \{(1,4)\}$)
SHIFT	$\Rightarrow$	(	$[1,4]$,	$[1,\ldots,4]$,	$[5,\ldots,9]$,	A_2)
SHIFT	$\Rightarrow$	(	$[1,4,5]$,	$[1,\ldots,5]$,	$[6,\ldots,9]$,	A_2)
LEFT-ARC	$\Rightarrow$	(	$[1,4,5]$,	$[1,\ldots,5]$,	$[6,\ldots,9]$,	$A_3 = A_2 \cup \{(6,5)\}$)
REDUCE	$\Rightarrow$	(	$[1,4]$,	$[1,\ldots,5]$,	$[6,\ldots,9]$,	A_3)
RIGHT-ARC	$\Rightarrow$	(	$[1,4]$,	$[1,\ldots,5]$,	$[6,\ldots,9]$,	$A_4 = A_3 \cup \{(4,6)\}$)
SWITCH	$\Rightarrow$	(	$[1,\ldots,5]$,	$[1,4]$,	$[6,\ldots,9]$,	A_4)
REDUCE	$\Rightarrow$	(	$[1,\ldots,4]$,	$[1,4]$,	$[6,\ldots,9]$,	A_4)
REDUCE	$\Rightarrow$	(	$[1,2,3]$,	$[1,4]$,	$[6,\ldots,9]$,	A_4)
REDUCE	$\Rightarrow$	(	$[1,2]$,	$[1,4]$,	$[6,\ldots,9]$,	A_4)
LEFT-ARC	$\Rightarrow$	(	$[1,2]$,	$[1,4]$,	$[6,\ldots,9]$,	$A_5 = A_4 \cup \{(6,2)\}$)
SHIFT	$\Rightarrow$	(	$[1,2,6]$,	$[1,4,6]$,	$[7,8,9]$,	A_5)
SWITCH	$\Rightarrow$	(	$[1,4,6]$,	$[1,2,6]$,	$[7,8,9]$,	A_5)
REDUCE	$\Rightarrow$	(	$[1,4]$,	$[1,2,6]$,	$[7,8,9]$,	A_5)
RIGHT-ARC	$\Rightarrow$	(	$[1,4]$,	$[1,2,6]$,	$[7,8,9]$,	$A_6 = A_5 \cup \{(4,7)\}$)
SHIFT	$\Rightarrow$	(	$[1,4,7]$,	$[1,\ldots,7]$,	$[8,9]$,	A_6)
RIGHT-ARC	$\Rightarrow$	(	$[1,4,7]$,	$[1,\ldots,7]$,	$[8,9]$,	$A_7 = A_6 \cup \{(7,8)\}$)
SHIFT	$\Rightarrow$	(	$[1,4,7,8]$,	$[1,\ldots,8]$,	[9],	A_7)
REDUCE	$\Rightarrow$	(	$[1,4,7]$,	$[1,\ldots,8]$,	[9],	A_7)
REDUCE	$\Rightarrow$	(	$[1,4]$,	$[1,\ldots,8]$,	[9],	A_7)
REDUCE	$\Rightarrow$	(	[1],	$[1,\ldots,8]$,	[9],	A_7)
RIGHT-ARC	$\Rightarrow$	(	[1],	$[1,\ldots,8]$,	[9],	$A_8 = A_7 \cup \{(1,9)\}$)
SHIFT	$\Rightarrow$	(	$[1,9]$,	$[1,\ldots,9]$,	[ ],	A_8)

Figure 6
2-planar transition sequence for the (unlabeled) dependency graph in Figure 1.

Proof

This lemma is proven by showing that the algorithm cannot create a pair of crossing arcs on the same stack. This is done by applying the proof of Proposition 4 separately to each of the two stacks of the 2-planar system (or, alternatively, by observing that the transition system resulting from ignoring one of the stacks in the 2-planar system is divisible). This implies that, given each of the two stacks, the subgraph formed by the arcs created by a transition sequence in configurations where that stack was active is planar, which trivially implies that the graph generated by the sequence is 2-planar. ■

Lemma 7

Let $G = (V, A)$ be a 2-planar dependency graph for a sentence $w_1 \ldots w_n$, with planes P_1 and P_2. Then there is a transition sequence in S_{2P} ending in a terminal configuration of the form $(\sigma^1, \sigma^2, [\,], A)$ such that all the nodes that are not covered by any dependency arc in P_1 are in σ^1, and all the nodes that are not covered by any dependency arc in P_2 are in σ^2.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Proof

The proof is analogous to that of the planar parser, but we have to handle two stacks and two planes. As in the planar case, we proceed by induction on the length n of the sentence. In the case where $n = 1$, the only possible 2-planar dependency graph is the graph $G_0 = (\{1\}, \varnothing)$ with a single node and no arcs, and the transition sequence that applies a single SHIFT transition meets the conditions of the lemma, because it ends in a terminal configuration ([1], [1], [ ], $\varnothing$).

For the induction step, we assume that the lemma holds for sentences of length n and prove that it also holds for sentences of length $n+1$, for any $n \geq 0$. Let $G_{n+1} = (V_{n+1}, A_{n+1})$ be a 2-planar dependency graph for a sentence $w_1 \dots w_{n+1}$, with planes $P^1_{n+1} = (V_{n+1}, A^1_{n+1})$ and $P^2_{n+1} = (V_{n+1}, A^2_{n+1})$. We denote by L_{n+1} the set of arcs

$$L_{n+1} = \{(n+1, i) \in A_{n+1}\} \cup \{(j, n+1) \in A_{n+1}\}$$

that is, the set of incoming and outgoing arcs from the node $n+1$ in G_{n+1}, and we denote by G_n the graph

$$G_n = (V_n = V_{n+1} \setminus \{n+1\}, A_n = A_{n+1} \setminus L_{n+1})$$

that is, the graph obtained by removing the node $n+1$ and all its incoming and outgoing arcs from G_{n+1}. It is easy to show that the graphs $P^1_n = (V_n, A^1_{n+1} \setminus L_{n+1})$ and $P^2_n = (V_n, A^2_{n+1} \setminus L_{n+1})$ are planes of G_n. They are planar graphs (being subgraphs of P^1_{n+1} and P^2_{n+1}, which are planar) and the union of their arc set is $A_{n+1} \setminus L_{n+1} = A_n$ (because $A^1_{n+1} \cup A^2_{n+1} = A_{n+1}$, as P^1_{n+1} and P^2_{n+1} are planes of G_{n+1}).

By the induction hypothesis, there exists a transition sequence C_n whose final configuration is of the form $(\sigma^1_n, \sigma^2_n, [\,], A_n)$, such that σ^b_n contains all the nodes that are not covered by any dependency arc in P^b_n, for $b = 1, 2$. From this transition sequence C_n, we will obtain a transition sequence C_{n+1} meeting the conditions asserted by the lemma for the graph G_{n+1}.

To do so, we first observe that for $b = 1, 2$, the planarity of the graph P^b_{n+1} implies that the left endpoints of the arcs in A^b_{n+1} cannot be covered by any arc in P^b_n, because this would mean that the arc in A^b_{n+1} and the covering arc would cross. Therefore, by the induction hypothesis, we know that all the left endpoints of the arcs in A^b_{n+1} are in σ^b_n. Thus, if the left endpoints of the arcs in A^1_{n+1} are $i_1, i_2, \dots, i_e$ and those of the arcs in A^2_{n+1} are $j_1, j_2, \dots, j_f$; then the stack σ^1_n (which is ordered, because the same reasoning as in Lemma 1 can be applied to the 2-planar transition system) is of the form

$$\sigma^1_n = [s_1, \dots, s_{z_1} = i_1, \dots, s_{z_2} = i_2, \dots, s_{z_e} = i_e, \dots, s_m]$$

and the stack σ^2_n is of the form

$$\sigma^2_n = [t_1, \dots, t_{y_1} = j_1, \dots, t_{y_2} = j_2, \dots, t_{y_f} = j_f, \dots, t_q]$$

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

With this in mind, we can obtain the transition sequence C_{n+1} from C_n by adding the following extra transitions at the end of its associated transition chain:

$$\textsc{Reduce}^{m-z_e}; arcs(i_e); \textsc{Reduce}^{z_e-z_{e-1}}; arcs(i_{e-1}); \ldots; \textsc{Reduce}^{z_2-z_1}; arcs(i_1); \textsc{Switch};$$
$$\textsc{Reduce}^{q-y_f}; arcs(j_f); \textsc{Reduce}^{y_f-y_{f-1}}; arcs(j_{f-1}); \ldots; \textsc{Reduce}^{y_2-y_1}; arcs(j_1); \textsc{Shift};$$
$$\textsc{Switch}$$

where we use the notation $arcs(i)$ as shorthand for:

- Left-Arc, if $(n+1, i) \in L_{n+1} \wedge (i, n+1) \notin L_{n+1}$,
- Right-Arc, if $(n+1, i) \notin L_{n+1} \wedge (i, n+1) \in L_{n+1}$,
- Left-Arc; Right-Arc, if $(n+1, i) \in L_{n+1} \wedge (i, n+1) \in L_{n+1}$.

The final configuration of the transition sequence obtained by applying these transitions at the end of C_n is of the form $(\sigma^1, \sigma^2, \beta, A)$, where:

- $\beta = [\,]$, because the nodes $1, \ldots, n$ are removed from the buffer by C_n, and $n+1$ is removed by the extra Shift transition;
- $A = A_{n+1}$, because the arcs in A_n are added to the set by C_n, and all the arcs in L_{n+1} are added by $arcs(i_e), \ldots, arcs(i_1), arcs(j_f), \ldots, arcs(j_1)$;
- all the nodes that are not covered by arcs in P^b_{n+1} are in σ^b, for $b = 1, 2$, because they were in σ^b_n (a node not covered by arcs in P^b_{n+1} is trivially not covered by arcs in P^b_n) and the Reduce transitions applied after C_n only remove nodes to the right of i_1 from the first stack, which are covered by the arc $(n+1, i_1)$ or $(i_1, n+1)$, and from the right of j_1 from the second stack, which are covered by the arc $(n+1, j_1)$ or $(j_1, n+1)$.[11]

This proves the induction step for Lemma 7. ■

Proposition 10

The system S_{2P} is correct for the set of 2-planar dependency graphs.

Proof

The proposition follows from Lemma 6 and Lemma 7. ■

5.1.2 Constraints on 2-Planar Dependency Parsing. The 2-planar parser can be restricted to graphs satisfying the Single-Head and Acyclicity constraints in exactly the same way as the planar parser, and the proofs follow the same line of reasoning. Therefore, a version of the 2-planar parser that is sound and complete for the set of 2-planar

11 This is assuming that arcs are created to or from node $n+1$ in both planes (i.e., that $e > 0$ and $f > 0$), but the cases where $e = 0$ or $f = 0$ are trivial, because in those cases no new Reduce transitions are applied to the respective stacks after C_n.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

dependency forests (that is, 2-planar dependency graphs without cycles and with each node having at most one head) can be defined as follows:

$$\begin{array}{lcl}
\textsc{Shift}_{2P} & = & (\sigma^1, \sigma^2, i|B, A) \Rightarrow (\sigma^1|i, \sigma^2|i, B, A) \\
\textsc{Reduce}_{2P} & = & (\sigma^1|i, \sigma^2, B, A) \Rightarrow (\sigma^1, \sigma^2, B, A) \\
\textsc{Left-Arc}_{2P-ShAc} & = & \textsc{Left-Arc}|_{\overline{H_\sigma(C)} \cap \{(\sigma^1|i,\sigma^2,j|\beta,A) \in C | \neg(i \leftrightarrow^* j \in A)\}} \\
& = & (\sigma^1|i, \sigma^2, j|B, A) \Rightarrow (\sigma^1|i, \sigma^2, j|B, A \cup \{(j, i)\}) \\
& & \text{only if } \neg\exists k \mid (k, i) \in A \text{ (single-head) and } \neg i \leftrightarrow^* j \in A \text{ (acyclicity)} \\
\textsc{Right-Arc}_{2P-ShAc} & = & \textsc{Right-Arc}|_{\overline{H_\beta(C)} \cap \{(\sigma^1|i,\sigma^2,j|\beta,A) \in C | \neg(i \leftrightarrow^* j \in A)\}} \\
& = & (\sigma^1|i, \sigma^2, j|B, A) \Rightarrow (\sigma^1|i, \sigma^2, j|B, A \cup \{(i, j)\}) \\
& & \text{only if } \neg\exists k|(k, j) \in A \text{ (single-head) and } \neg i \leftrightarrow^* j \in A \text{ (acyclicity)} \\
\textsc{Switch}_{2P} & = & (\sigma^1, \sigma^2, B, A) \Rightarrow (\sigma^2, \sigma^1, B, A)
\end{array}$$

Because structures in dependency treebanks are typically restricted to forests, this is the version of the 2-planar parser that we use in the experimental evaluation in Section 5.2.

The NO-COVERED-ROOTS constraint is not so straightforward to implement in the 2-planar parser, because in the 2-planar case a node without a head may need to be reduced from one stack and get a head later from the other stack, so restricting the REDUCE transitions in the 2-planar parser to nodes with a head would also forbid some structures without covered roots. In any case, the NO-COVERED-ROOTS constraint does not seem practically meaningful when we go beyond planar structures.

5.1.3 Complexity of 2-Planar Dependency Parsing. To reason about the complexity of the 2-planar parser, we first note that a naive implementation of the transition system as given here does not guarantee termination. The reason is that the system allows an infinite sequence of SWITCH transitions, switching the active and inactive stacks repeatedly and cycling between the same two configurations without making any advance. This can easily be avoided in practice by forbidding SWITCH transitions from being executed if the last transition in the sequence was also a SWITCH. Note that we could also have incorporated this restriction into the formal system (for example, by adding a flag to configurations to indicate whether the previous transition was a SWITCH or not), but this would have unnecessarily complicated the notation. Assuming that our implementation of the 2-planar parser has this restriction on SWITCH transitions, we can show that the length of a transition sequence for a sentence of length n is $O(n)$ in the same way as for efficient divisible systems (see Section 3.2.2).

Proposition 11

Let S_{2P} be the 2-planar system restricted so that two consecutive SWITCH transitions are not permitted. Then the length of every transition sequence for a sentence x of length n in S_{2P} is $O(n)$.

Proof

The proof follows the same lines as for efficient divisible transition systems. For every transition chain $T_{1,m} = t_1, \ldots, t_m$ for $x = w_1, \ldots, w_n$, the following must hold:

- The number of SHIFT transitions in $T_{1,m}$ is at most n, because each node in $\{1, \ldots, n\}$ can only be shifted once.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

- The number of REDUCE transitions in $T_{1,m}$ is at most $2n$, because each node in $\{1, \ldots, n\}$ can only be reduced twice (once per stack).
- The number of LEFT-ARC and RIGHT-ARC transitions in $T_{1,m}$ is bounded by the maximum number of arcs in a 2-planar dependency graph with n nodes, which is $8n - 12$.[12]
- Given the ban on consecutive SWITCH transitions, the maximum number of SWITCH transitions in $T_{1,m}$ is 1 plus the number of other transitions.

It follows that $m \leq 2(n + 2n + 8n - 12) + 1$ and hence that m is $O(n)$. ■

Applying the same reasoning as for the planar parser regarding constant-time execution of transitions and fixed-size beam search, we conclude that the complexity of the 2-planar parser is still $O(n)$, both for the unrestricted version and for the variant with the SINGLE-HEAD and ACYCLICITY constraints.

Throughout this article, we have presented complexity results for transition-based parsers under the assumption that these parsers use deterministic search or fixed-size beam search because this is the most straightforward method to make parsing practically feasible with the rich history-based feature models that are the key component of accurate transition-based parsers. The relevance of this assumption is further supported by recent results on tabularization and dynamic programming for transition-based parsing, which show that such techniques either lead to a significant increase in parsing complexity or require drastic simplifications in the feature models used. In the former case, practical parsing still has to rely on approximate inference, as in Huang and Sagae (2010). In the latter case, dynamic programming provides an exact inference method only for a very simple approximation of the original transition-based model, as in Kuhlmann, Gómez-Rodríguez, and Satta (2011). In general, this exemplifies the tradeoff between approximate inference with richer models (beam search) and exact inference with simpler models (dynamic programming). Thus, although the feature model used by Zhang and Nivre (2011) to achieve state-of-the-art accuracy for English makes dynamic programming very difficult due to the combinatorial effect on parsing complexity of complex valency and label set features, the feature representation of a single configuration can still be computed in constant time, which is all that is required to achieve linear-time parsing with beam search. The same is true for all the transition systems and feature models explored in this article. Nevertheless, it is an interesting theoretical question whether the novel 2-planar system allows for tabularization and what the resulting complexity would be. At present, we do not know the exact answer to this question, but a reasonable conjecture is that complexity would be exponential for the class of feature models that are relevant for transition-based parsing.

5.2 Experimental Evaluation

In this section, we present an experimental evaluation of the novel 1-planar and 2-planar transition systems in comparison to the widely used arc-eager projective system of Nivre (2003) (analyzed earlier in Example (4)). Besides being the default parsing

12 This follows from Lemma 4, because a 2-planar graph can be broken up into two planes, each of which is a planar graph with n nodes. Moreover, if the SINGLE-HEAD and ACYCLICITY constraints are used, the maximum number of arcs is $n - 1$, because every node can have at most one incoming arc and there must be at least one root.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

algorithm in MaltParser (Nivre, Hall, and Nilsson 2006), this system is also the basis of the ISBN Dependency Parser (Titov and Henderson 2007) and ZPar (Zhang and Clark 2008; Zhang and Nivre 2011). In addition to a strictly projective arc-eager parser, we also include a version that uses **pseudo-projective parsing** (Nivre and Nilsson 2005) to recover non-projective arcs. This is the most widely used method for non-projective transition-based parsing and as such a competitive baseline for the 2-planar parser.

In order to make the comparison as exact as possible, we have chosen to implement all four systems in the MaltParser framework and use the same type of classifiers and feature models. For the arc-eager baselines, we copy the set-up from the CoNLL-X shared task on dependency parsing, which includes the use of support vector machines with a polynomial kernel, history-based feature models tuned separately for each language, and pseudo-projective parsing with the Head encoding (Nivre et al. 2006). For the 1-planar and 2-planar parsers, we use the same type of classifier but modify the feature model to take into account the following systematic differences between the transition systems:

- In both the 1-planar and 2-planar parser, we need to add features over the arc connecting the top node of the stack and the first node of the buffer (if any). No such arc can exist in the arc-eager system used by the projective and pseudo-projective baseline systems.
- In the 2-planar parser, we need to add features over the top nodes of the inactive stack. No such nodes exist in the 1-planar and arc-eager systems.

We did not perform extensive feature optimization experiments for the new systems, so it is likely that there is room for further improvement. For replicability, the complete experimental settings are available at `http://stp.lingfil.uu.se/~nivre/exp`.

Table 2 shows parsing results for the same eight data sets from the CoNLL-X shared task that were investigated with respect to *k*-planarity in Section 4.2: Arabic, Czech, Danish, Dutch, German, Portuguese, Swedish, and Turkish. The overall accuracy metric is labeled attachment score (LAS), the percentage of tokens that are assigned both the

Table 2
Parsing accuracy for projective, 1-planar, pseudo-projective, and 2-planar transition systems in MaltParser. LAS = labeled attachment score; LP-NP = labeled precision on non-projective arcs; LR-NP = labeled recall on non-projective arcs. Statistical significance of LAS differences reported at the 0.05 level using McNemar's test: $\star$ = significantly better than one other system; $\ddagger$ = significantly better than two other systems; $\overset{\ddagger}{\star}$ = significantly better than three other systems.

	LAS				LP-NP				LR-NP			
Language	**Pr**	**1Pl**	**PPr**	**2Pl**	**Pr**	**1Pl**	**PPr**	**2Pl**	**Pr**	**1Pl**	**PPr**	**2Pl**
Arabic	66.2	66.7	66.1	‡67.1	–	25.0	–	10.0	0.0	9.1	18.2	27.3
Czech	77.6	78.3	‡79.7	‡80.1	–	52.0	77.0	71.4	7.5	15.9	58.9	58.9
Danish	85.1	84.6	84.9	84.8	–	56.5	42.9	55.6	0.0	12.5	22.5	17.5
Dutch	75.2	74.9	$\overset{\ddagger}{\star}$78.1	‡77.2	–	59.9	62.1	66.0	1.9	11.7	47.0	46.2
German	85.6	85.7	⋆86.2	‡86.9	–	54.7	61.4	71.2	16.9	35.6	42.4	45.8
Portuguese	85.6	⋆86.3	‡87.1	‡87.6	–	43.8	82.4	77.8	1.5	3.1	44.6	35.4
Swedish	⋆84.0	83.0	⋆84.0	83.5	–	24.4	16.7	6.9	12.5	12.5	12.5	4.2
Turkish	63.7	63.8	64.2	63.5	–	25.9	12.5	7.0	6.6	13.2	10.5	10.5

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

correct head and the correct label. In addition, we report labeled precision (LP-NP) and recall (LR-NP) specifically on non-projective dependency arcs, where an arc (i, j) is taken to be non-projective if and only if there is some node k such that $\min(i, j) < k < \max(i, j)$ and not $i \rightarrow^* k$. Precision is the percentage of non-projective arcs output by the system that are correct, and recall is the percentage of non-projective arcs in the gold standard that are output by the system. Note that, although precision is undefined for the projective parser because it does not output any non-projective arcs, recall may nevertheless be greater than zero because arcs that are non-projective in the gold standard can be projective in the output of the parser.[13]

Looking first at the overall LAS results, we see that the 2-planar parser outperforms both the 1-planar and the projective parser for languages with a high proportion of non-projective trees ($\geq$ 19%): Czech, Dutch, German, and Portuguese. This is in line with our expectations, given the substantially higher coverage of the 2-planar parser for non-projective structures, and the difference is statistically significant at the 0.05 level for all languages in this group (McNemar's test). For three of these languages, the 2-planar parser also outperforms the pseudo-projective parser, although the differences are not statistically significant, and only in the case of Dutch is the pseudo-projective parser significantly better. Given the relatively small difference in coverage between the projective and 1-planar parser, one would expect these systems to have very similar performance, and this is also what we find except for Portuguese where the 1-planar parser is significantly better than the projective arc-eager parser.

For languages with a lower proportion of non-projective trees (Arabic, Danish, Swedish, Turkish), there are generally smaller differences between the parsers, and for Danish and Turkish there are in fact no statistically significant differences at all, which indicates that the increased expressivity is not beneficial (nor harmful) when non-projective structures are rare. Interestingly, it seems that the planar parsers have an advantage over the arc-eager parsers for Arabic, where the 2-planar parser is significantly better than both the projective and pseudo-projective parsers. By contrast, the arc-eager parsers seem to have an advantage for Swedish, where the projective and pseudo-projective parsers are both significantly better than the 1-planar parser. At present, we have no explanation for this language-specific variation.

Turning next to labeled precision (LP-NP) and recall (LR-NP) on non-projective dependency arcs, we again find that the 2-planar parser does quite well on the four languages with 19% or more non-projective trees, with precision consistently over 50% and recall in the 35–60% range. Again, the results are very similar to those achieved with the pseudo-projective parser, with the 2-planar parser giving higher precision for Dutch and German and higher recall for German. For the remaining four languages, both precision and recall remains low, which probably points to a sparse data problem when learning how to switch between the two planes during parsing, but the same holds true for the pseudo-projective parser. As expected, the 1-planar parser has only marginally higher recall than the projective parser (which, as pointed out earlier, may recover non-projective dependencies by accident), but it is interesting to note that the 1-planar parser has relatively high precision on the few non-projective arcs that it predicts, in some cases comparable to that of the 2-planar parser.

In conclusion, the experimental evaluation shows that the 2-planar parser has the potential to improve parsing accuracy over a strictly projective (or 1-planar) parser

13 When this happens, there is by necessity an error elsewhere in the parser output, because the projectivity of the arc implies that at least one gold standard arc must be missing.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

for languages with a sufficient proportion of non-projective trees, and that it generally performs at about the same level as the widely used arc-eager pseudo-projective parser. We believe that it is possible to improve results even further by careful optimization of features and other parameters, but this will have to be left for future research. It would also be interesting to explore the use of global optimization and beam search, which has been shown to improve accuracy over local learning and greedy search (Titov and Henderson 2007; Zhang and Clark 2008; Zhang and Nivre 2011).

6. Related Work

The literature on dependency parsing has grown enormously in recent years and we will not attempt a comprehensive review here but focus on previous research related to the three main themes of the article: a formal framework for analyzing and constructing transition systems for dependency parsing (Section 3), a procedure for classifying mildly non-projective dependency structures in terms of multiplanarity (Section 4), and a novel transition-based parser for (a subclass of) non-projective dependency structures (Section 5).

6.1 Frameworks for Dependency Parsing

Due to the growing popularity of dependency parsing, several proposals have been made that group and study different dependency parsers under common (more or less formal) frameworks. Thus, Buchholz and Marsi (2006) observed that almost all of the systems participating in the CoNLL-X shared task could be classified as belonging to one of two approaches, which they called the "all pairs" and the "stepwise" approaches. This was taken up by McDonald and Nivre (2007), who called the first approach global exhaustive graph-based parsing and the second approach local greedy transition-based parsing. The terms *graph-based* and *transition-based* have become well established, even though there now exist graph-based models that do not perform exhaustive search (McDonald and Pereira 2006; Koo et al. 2010) as well as transition-based models that are neither local nor greedy (Titov and Henderson 2007; Zhang and Clark 2008).

Nivre (2008), building on earlier work in Nivre (2006b), formalizes transition-based parsing by means of *transition systems* and *oracles*. Two distinct types of transition systems are described, differing in the data structures they use to store partially processed tokens: stack-based and list-based systems. The formalization of stack-based systems provided there has been one point of departure for the present article (see Section 2.2) but, whereas general stack-based systems allow transitions to be arbitrary partial functions from configurations to configurations, we have focused on a class of systems where transitions are obtained by composing a small set of elementary transitions, allowing us to derive specific formal properties.

Gómez-Rodríguez, Carroll, and Weir (2011) propose a common deductive framework that can be used to describe a wide range of dependency parsers, including both graph-based and transition-based algorithms. Although the high abstraction level of this framework makes it able to describe and relate very different parsing strategies, it also means that it is not suitable to describe lower-level properties of transition-based parsers such as their computational complexity when implemented with beam search. Kuhlmann, Gómez-Rodríguez, and Satta (2011) introduce a technique to obtain polynomial-time deductive parsers that simulate all the transition sequences allowed by a transition system.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

6.2 Mildly Non-Projective Dependency Structures

Most natural language treebanks contain non-projective dependency analyses (Havelka 2007), but the general problem of parsing arbitrary non-projective dependency graphs has been shown to be computationally intractable except under strong independence assumptions (McDonald and Satta 2007). This has motivated researchers to look for sets of dependency structures that have more coverage of linguistic phenomena than projective structures, while being more efficiently parsable than unrestricted non-projective graphs.

Several sets have been defined by applying different restrictions to dependency graphs, such as arc degree (Nivre 2006a, 2007), gap degree and well-nestedness (Bodirsky, Kuhlmann, and Möhl 2005; Kuhlmann and Nivre 2006; Kuhlmann and Möhl 2007), and k-ill-nestedness (Maier and Lichte 2009). Among these sets, only well-nested dependency structures with bounded gap degree have been shown to have exact polynomial-time algorithms (Kuhlmann 2010; Gómez-Rodríguez, Carroll, and Weir 2011). For dependency structures with bounded arc degree, a greedy transition-based parser based on the algorithm of Covington (2001) is described in Nivre (2007).

Other sets have been defined operationally as the set of dependency structures that are parsable by a given algorithm. These include the graphs parsable by the transition system of Attardi (2006) or the more restrictive dynamic programming variant of Cohen, Gómez-Rodríguez, and Satta (2011), the set of structures that yield binarizable productions with the algorithm of Kuhlmann and Satta (2009), or the set of mildly ill-nested structures (Gómez-Rodríguez, Weir, and Carroll 2009; Gómez-Rodríguez, Carroll, and Weir 2011).

As mentioned earlier, the notion of multiplanarity was originally introduced by Yli-Jyrä (2003), who also presents additional constraints on k-planar graphs. No algorithms were previously known to determine whether a given graph was k-planar or to efficiently parse k-planar dependency structures, however.

6.3 Non-Projective Transition-Based Parsing

Whereas early transition-based dependency parsers were restricted to projective dependency graphs (Yamada and Matsumoto 2003; Nivre 2003), several techniques have been proposed to accomodate non-projectivity within the transition-based framework. Pseudo-projective parsing, proposed by Nivre and Nilsson (2005), is a general technique applicable to any data-driven parser. Before training the parser, dependency structures are projectivized using lifting operations (Kahane, Nasr, and Rambow 1998), and partial information about the lifting paths is encoded in augmented arc labels. After parsing, dependency structures are deprojectivized using a heuristic search procedure guided by the augmented arc labels.

A more integrated approach is to deal with with non-projectivity by adding extra transitions to projective transition systems. Attardi (2006) parses a restricted set of non-projective trees by adding transitions that create arcs using nodes deeper than the top of the stack. Nivre (2009) instead uses a transition that changes the order of input words, obtaining full coverage of non-projective structures in quadratic worst-case time (but achieving linear practical performance). A similar technique is used by Tratz and Hovy (2011) to develop an $O(n^2 \log n)$ non-projective version of the easy-first parser of Goldberg and Elhadad (2010).

Finally, the parsing algorithm described by Covington (2001) can be implemented as a list-based transition system that in its unrestricted form is complete for

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

all non-projective trees (Nivre 2008). The worst-case complexity for this system is $O(n^2)$, but efficiency can be improved in practice by bounding the arc degree (Nivre 2006a, 2007).

7. Conclusion

Although data-driven dependency parsing has seen tremendous progress during the last decade in terms of empirically observed accuracy for a wide range of languages, it is probably fair to say that our theoretical understanding of the methods used is still less developed than for the more familiar paradigm of context-free grammar parsing. In this article, we have tried to contribute to the theoretical foundations of dependency parsing in essentially two ways.

Our first contribution is the framework of divisible transition systems, where transition systems for dependency parsing can be defined by composition and restriction of the five elementary transitions SHIFT, UNSHIFT, REDUCE, LEFT-ARC, and RIGHT-ARC. On the one hand, this can be used as an analytical tool to characterize existing systems for dependency parsing and prove formal properties related to expressivity and complexity. Thus, we have shown that all divisible systems, including a number of well-known systems from the literature, are sound for planar dependency graphs and can be restricted to satisfy a number of other formal constraints, and we have characterized the subclass of efficient divisible transition systems that give linear parsing complexity when combined with greedy inference or beam search as is customary in transition-based parsing. Even though most of these results have been established previously for particular systems, the general framework allows us to show how the results follow from more general principles. On the other hand, the framework can be used to develop new systems with required formal properties. To illustrate this, we have presented a system that is both sound and complete for planar dependency graphs (with or without additional formal constraints) and that fills a gap in the dependency parsing literature.

Our second contribution consists in extending the available techniques for dependency parsing to multiplanar dependency graphs, an interesting hierarchy of mildly non-projective dependency structures that have remained unexplored due to the lack of suitable formal tools. First of all, we have shown that the problem of finding the smallest k such that a dependency graph is k-planar can be reduced to the familiar k-coloring problem for undirected graphs and can thereby be solved efficiently for $k \leq 2$ but in practice also for higher k due to the sparseness of non-projective dependencies in natural language. Using this procedure, we have shown that the set of 2-planar dependency trees have a coverage in existing treebanks that is at least as good as alternative characterizations of mildly non-projective dependency structures. In addition, we have shown how the planar dependency parser defined in the first part of the article can be generalized to the k-planar dependency graphs and in particular to the 2-planar case. Preliminary experiments using standard methods for transition-based parsing show that this system can give significant improvements over a strictly projective system for languages with a non-negligible proportion of non-projective dependencies.

There are a number of directions for future research that suggest themselves. First of all, there are many instances of divisible transition systems that have not yet been explored, either theoretically or for practical parsing applications. For example, as remarked in Section 3.3.2, there is a way of restricting the 1-planar parser to projective forests, which is different from previously explored systems for projective dependency parsing. Secondly, it may be interesting to study different ways of extending divisible

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

transition systems for greater expressivity, besides introducing additional stacks. This may involve the addition of new transition types, as proposed by Attardi (2006) and Nivre (2009), or new data structures, as in the list-based systems of Nivre (2008). Finally, it would be interesting to see what level of accuracy can be reached for 2-planar dependency parsing with proper feature selection in combination with the latest techniques for global optimization and non-greedy search (Titov and Henderson 2007; Zhang and Clark 2008; Huang and Sagae 2010; Zhang and Nivre 2011).

Acknowledgments

The authors would like to thank Johan Hall for support with the MaltParser system and three anonymous reviewers for useful comments on previous versions of the manuscript. The first author has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER (project TIN2010-18552-C03-02) and Xunta de Galicia (Rede Galega de Recursos Lingüísticos para unha Sociedade do Coñecemento). Part of the reported experiments were conducted with the help of computing resources provided by the Supercomputing Center of Galicia (CESGA).

References

Afonso, Susana, 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 1,968–1,703, Paris.

Atalay, Nart B., 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.

Attardi, Giuseppe. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170, New York, NY.

Bikel, Daniel M. and Vittorio Castelli. 2008. Event matching using the transitive closure of dependency relations. In *Proceedings of ACL-08: HLT, Short Papers*, pages 145–148, Columbus, OH.

Bodirsky, Manuel, Marco Kuhlmann, and Mattias Möhl. 2005. Well-nested drawings as models of syntactic structure. In *Proceedings of FG-MoL 2005: The 10th Conference on Formal Grammar*, pages 195–203, Edinburgh.

Böhmová, Alena, Jan Hajič, Eva Hajičová, and Barbora Hladká. 2003. The Prague Dependency Treebank: A three-level annotation scenario. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*. Kluwer, pages 103–127.

Brants, Sabine, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. TIGER treebank. In *Proceedings of the 1st Workshop on Treebanks and Linguistic Theories (TLT)*, pages 24–42, Sozopol.

Buchholz, Sabine 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, New York, NY.

Buyko, Ekaterina and Udo Hahn. 2010. Evaluating the impact of alternative dependency graph encodings on solving event extraction tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 982–992, Cambridge, MA.

Carreras, Xavier. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 957–961, Prague.

Cohen, Shay B., Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Exact inference for generative probabilistic non-projective dependency parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP 2011)*, pages 1,234–1,245, Edinburgh.

Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. 1990. *Introduction to Algorithms*. MIT Press.

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

Culotta, Aron and Jeffery Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual*

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Meeting of the Association for Computational Linguistics (ACL), pages 423–429, Barcelona.
Eisner, Jason M. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 340–345, Copenhagen.
Goldberg, Yoav and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 742–750, Los Angeles, CA.
Gómez-Rodríguez, Carlos, John Carroll, and David Weir. 2011. Dependency parsing schemata and mildly non-projective dependency parsing. *Computational Linguistics*, 37(3):541–586.
Gómez-Rodríguez, Carlos 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)*, pages 1,492–1,501, Uppsala.
Gómez-Rodríguez, Carlos, 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, Athens.
Hajič, Jan, 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.
Hajič, Jan, 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, Cairo.
Hajič, Jan, Barbora Vidova Hladka, Jarmila Panevová, Eva Hajičová, Petr Sgall, and Petr Pajas. 2001. Prague Dependency Treebank 1.0. LDC, 2001T10.
Havelka, Jiri. 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, Prague.
Huang, Liang, Wenbin Jiang, and Qun Liu. 2009. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1,222–1,231, Suntec.
Huang, Liang 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)*, pages 1,077–1,086, Uppsala.
Hudson, Richard A. 1990. *English Word Grammar*. Blackwell.
Johansson, Richard and Pierre Nugues. 2006. Investigating multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 206–210, New York, NY.
Kahane, Sylvain, Alexis Nasr, and Owen Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics (ACL) and the 17th International Conference on Computational Linguistics (COLING)*, pages 646–652, Montréal.
Karp, Richard M. 1972. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*. Plenum Press, pages 85–103.
Koo, Terry and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–11, Uppsala.
Koo, Terry, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1,288–1,298, Cambridge, MA.
Kromann, Matthias Trautner. 2003. The Danish Dependency Treebank and the DTAG treebank tool. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 217–220, Växjö.
Kuhlmann, Marco. 2010. *Dependency Structures and Lexicalized Grammars: An Algebraic Approach*, volume 6270 of *Lecture Notes in Computer Science*. Springer.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Kuhlmann, Marco, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 673–682, Portland, OR.

Kuhlmann, Marco 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, Prague.

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

Kuhlmann, Marco 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, Athens.

Maier, Wolfgang and Timm Lichte. 2009. Characterizing discontinuity in constituent treebanks. In *Proceedings of the 14th Conference on Formal Grammar*, pages 164–179, Opole.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330.

Marcus, Mitchell P., Beatrice Santorini, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating predicate-argument structure. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 114–119, Plainsboro, NJ.

Martins, Andre, 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, Suntec.

Martins, Andre, Noah Smith, Eric Xing, Pedro Aguiar, and Mario Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 34–44, Cambridge, MA.

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

McDonald, Ryan 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, Prague.

McDonald, Ryan and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 81–88, Trento.

McDonald, Ryan 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, Prague.

Mel'čuk, Igor. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

Nakagawa, Tetsuji. 2007. Multilingual dependency parsing using global features. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 952–956, Prague.

Nilsson, Jens, Johan Hall, and Joakim Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from Antiquity. In *Proceedings of the NODALIDA Special Session on Treebanks*, pages 121–132, Joensuu.

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

Nivre, Joakim. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pages 50–57, Barcelona.

Nivre, Joakim. 2006a. 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, Trento.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Nivre, Joakim. 2006b. *Inductive Dependency Parsing*. Springer.
Nivre, Joakim. 2007. 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)*, pages 396–403, Rochester, NY.
Nivre, Joakim. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.
Nivre, Joakim. 2009. Non-projective dependency parsing in expected linear time. 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 351–359, Suntec.
Nivre, Joakim, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pages 49–56, Boston, MA.
Nivre, Joakim, 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 2,216–2,219, Genoa.
Nivre, Joakim, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225, New York, NY.
Nivre, Joakim 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, Ann Arbor, MI.
Oflazer, Kemal, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür. 2003. Building a Turkish treebank. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*. Kluwer, pages 261–277.
Quirk, Chris, Arul Menezes, and Colin Cherry. 2005. Dependency treelet translation: Syntactically informed phrasal SMT. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 271–279, Ann Arbor, MI.
Riedel, Sebastian and James Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 129–137, Sydney.
Sagae, Kenji and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING)*, pages 753–760, Manchester.
Sgall, Petr, Eva Hajičová, and Jarmila Panevová. 1986. *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel.
Shen, Dan and Dietrich Klakow. 2006. Exploring correlation of dependency relation paths for answer extraction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 889–896, Sydney.
Shieber, Stuart M., Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36.
Smith, David and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 145–156, Waikiki, HI.
Stevenson, Mark and Mark A. Greenwood. 2006. Comparing information extraction pattern models. In *Proceedings of the Workshop on Information Extraction Beyond The Document*, pages 12–19, Sydney.
Tesnière, Lucien. 1959. *Éléments de syntaxe structurale*. Editions Klincksieck.
Titov, Ivan 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, Prague.
Tratz, Stephen and Eduard Hovy. 2011. A fast, accurate, non-projective, semantically-enriched parser. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1,257–1,268, Edinburgh.
Van der Beek, Leonoor, Gosse Bouma, Robert Malouf, and Gertjan Van Noord.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

2002. The Alpino dependency treebank. In Mariet Theune, Anton Nijholt, and Hendri Hondorp, editors, *Language and Computers, Computational Linguistics in the Netherlands 2001. Selected Papers from the Twelfth CLIN Meeting*, pages 8–22, Rodopi.

Xu, Peng, Jaeho Kang, Michael Ringgaard, and Franz Och. 2009. Using a dependency parser to improve SMT for subject-object-verb languages. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 245–253, Boulder, CO.

Yamada, Hiroyasu 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, Nancy.

Yli-Jyrä, Anssi. 2003. Multiplanarity—a model for dependency structures in treebanks. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories (TLT)*, pages 189–200, Växjö.

Zhang, Yue and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 562–571, Waikiki, HI.

Zhang, Yue and Joakim Nivre. 2011. Transition-based parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 188–193, Portland, OR.

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024

Downloaded from http://direct.mit.edu/coli/article-pdf/39/4/799/1802388/coli_a_00150.pdf by guest on 23 April 2024