Parsing Mildly Non-projective Dependency Structures[*]

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

David Weir and John Carroll
Department of Informatics
University of Sussex, United Kingdom
{davidw,johnca}@sussex.ac.uk

Abstract

We present parsing algorithms for various mildly non-projective dependency formalisms. In particular, algorithms are presented for: all well-nested structures of gap degree at most 1, with the same complexity as the best existing parsers for constituency formalisms of equivalent generative power; all well-nested structures with gap degree bounded by any constant k; and a new class of structures with gap degree up to k that includes some ill-nested structures. The third case includes all the gap degree k structures in a number of dependency treebanks.

1 Introduction

Dependency parsers analyse a sentence in terms of a set of directed links (dependencies) expressing the head-modifier and head-complement relationships which form the basis of predicate argument structure. We take dependency structures to be directed trees, where each node corresponds to a word and the root of the tree marks the syntactic head of the sentence. For reasons of efficiency, many practical implementations of dependency parsing are restricted to *projective* structures, in which the subtree rooted at each word covers a contiguous substring of the sentence. However, while free word order languages such as Czech do not satisfy this constraint, parsing without the projectivity constraint is computationally complex. Although it is possible to parse non-projective structures in quadratic time under a model in which each dependency decision is independent of all the others (McDonald et al., 2005), the problem is intractable in the absence of this assumption (McDonald and Satta, 2007).

Nivre and Nilsson (2005) observe that most non-projective dependency structures appearing in practice are "close" to being projective, since they contain only a small proportion of non-projective arcs. This has led to the study of classes of dependency structures that lie between projective and unrestricted non-projective structures (Kuhlmann and Nivre, 2006; Havelka, 2007). Kuhlmann (2007) investigates several such classes, based on well-nestedness and gap degree constraints (Bodirsky et al., 2005), relating them to lexicalised constituency grammar formalisms. Specifically, he shows that: linear context-free rewriting systems (LCFRS) with fan-out k (Vijay-Shanker et al., 1987; Satta, 1992) induce the set of dependency structures with gap degree at most $k - 1$; coupled context-free grammars in which the maximal rank of a nonterminal is k (Hotz and Pitsch, 1996) induce the set of well-nested dependency structures with gap degree at most $k - 1$; and LTAGs (Joshi and Schabes, 1997) induce the set of well-nested dependency structures with gap degree at most 1.

These results establish that there must be polynomial-time dependency parsing algorithms for well-nested structures with bounded gap degree, since such parsers exist for their corresponding lexicalised constituency-based formalisms. However, since most of the non-projective structures in treebanks are well-nested and have a small gap degree (Kuhlmann and Nivre, 2006), developing efficient dependency parsing strategies for these sets of structures has considerable practical interest, since we would be able to parse directly with dependencies in a data-driven manner, rather than indirectly by constructing intermediate constituency grammars and extracting dependencies from constituency parses.

We address this problem with the following contributions: (1) we define a parsing algorithm

[*]Partially supported by MEC and FEDER (HUM2007-66607-C04) and Xunta de Galicia (PGIDIT07SIN005206PR, INCITE08E1R104022ES, INCITE08ENA305025ES, INCITE08PXIB302179PR, Rede Galega de Proc. da Linguaxe e RI, Bolsas para Estadías INCITE – FSE cofinanced).

for well-nested dependency structures of gap degree 1, and prove its correctness. The parser runs in time $O(n^7)$, the same complexity as the best existing algorithms for LTAG (Eisner and Satta, 2000), and can be optimised to $O(n^6)$ in the non-lexicalised case; (2) we generalise the previous algorithm to any well-nested dependency structure with gap degree at most k in time $O(n^{5+2k})$; (3) we generalise the previous parsers to be able to analyse not only well-nested structures, but also ill-nested structures with gap degree at most k satisfying certain constraints[1], in time $O(n^{4+3k})$; and (4) we characterise the set of structures covered by this parser, which we call *mildly ill-nested* structures, and show that it includes all the trees present in a number of dependency treebanks.

2 Preliminaries

A *dependency graph* for a string $w_1 \dots w_n$ is a graph $G = (V, E)$, where $V = \{w_1, \dots, w_n\}$ and $E \subseteq V \times V$. We write the edge (w_i, w_j) as $w_i \rightarrow w_j$, meaning that the word w_i is a syntactic *dependent* (or a *child*) of w_j or, conversely, that w_j is the *governor* (*parent*) of w_i. We write $w_i \rightarrow^\star w_j$ to denote that there exists a (possibly empty) path from w_i to w_j. The *projection* of a node w_i, denoted $\lfloor w_i \rfloor$, is the set of reflexive-transitive dependents of w_i, that is: $\lfloor w_i \rfloor = \{w_j \in V \mid w_j \rightarrow^\star w_i\}$. An *interval* (with endpoints i and j) is a set of the form $[i, j] = \{w_k \mid i \leq k \leq j\}$.

A dependency graph is said to be a *tree* if it is: (1) acyclic: $w_j \in \lfloor w_i \rfloor$ implies $w_i \rightarrow w_j \notin E$; and (2) each node has exactly one parent, except for one node which we call the *root* or *head*. A graph verifying these conditions and having a vertex set $V \subseteq \{w_1, \dots, w_n\}$ is a *partial dependency tree*. Given a dependency tree $T = (V, E)$ and a node $u \in V$, the *subtree* induced by the node u is the graph $T_u = (\lfloor u \rfloor, E_u)$ where $E_u = \{w_i \rightarrow w_j \in E \mid w_j \in \lfloor u \rfloor\}$.

2.1 Properties of dependency trees

We now define the concepts of gap degree and well-nestedness (Kuhlmann and Nivre, 2006). Let T be a (possibly partial) dependency tree for $w_1 \dots w_n$: We say that T is **projective** if $\lfloor w_i \rfloor$ is an interval for every word w_i. Thus every node in the dependency structure must dominate a contiguous substring in the sentence. The **gap degree** of a particular node w_k in T is the minimum $g \in \mathbb{N}$ such that $\lfloor w_k \rfloor$ can be written as the union of $g+1$ intervals; that is, the number of discontinuities in $\lfloor w_k \rfloor$. The gap degree of the dependency tree T is the maximum among the gap degrees of its nodes. Note that T has gap degree 0 if and only if T is projective. The subtrees induced by nodes w_p and w_q are **interleaved** if $\lfloor w_p \rfloor \cap \lfloor w_q \rfloor = \emptyset$ and there are nodes $w_i, w_j \in \lfloor w_p \rfloor$ and $w_k, w_l \in \lfloor w_q \rfloor$ such that $i < k < j < l$. A dependency tree T is **well-nested** if it does not contain two interleaved subtrees. A tree that is not well-nested is said to be **ill-nested**. Note that projective trees are always well-nested, but well-nested trees are not always projective.

2.2 Dependency parsing schemata

The framework of parsing schemata (Sikkel, 1997) provides a uniform way to describe, analyse and compare parsing algorithms. Parsing schemata were initially defined for constituency-based grammatical formalisms, but Gómez-Rodríguez et al. (2008a) define a variant of the framework for dependency-based parsers. We use these *dependency parsing schemata* to define parsers and prove their correctness. Due to space constraints, we only provide brief outlines of the main concepts behind dependency parsing schemata.

The parsing schema approach considers parsing as deduction, generating intermediate results called *items*. An initial set of items is obtained from the input sentence, and the parsing process involves *deduction steps* which produce new items from existing ones. Each item contains information about the sentence's structure, and a successful parsing process produces at least one *final item* providing a full dependency analysis for the sentence or guaranteeing its existence. In a dependency parsing schema, items are defined as sets of partial dependency trees[2]. To define a parser by means of a schema, we must define an item set and provide a set of deduction steps that operate on it. Given an item set $\mathcal{I}$, the set of *final items* for strings of length n is the set of items in $\mathcal{I}$ that contain a full dependency tree for some arbitrary string of length n. A final item containing a dependency tree for a particular string $w_1 \dots w_n$ is said to be a *correct final item* for that string. These

[1]Parsing unrestricted ill-nested structures, even when the gap degree is bounded, is NP-complete: these structures are equivalent to LCFRS for which the recognition problem is NP-complete (Satta, 1992).

[2]The formalism allows items to contain forests, and the dependency structures inside items are defined in a notation with terminal and preterminal nodes, but these are not needed here.

concepts can be used to prove the correctness of a parser: for each input string, a parsing schema's deduction steps allow us to infer a set of items, called *valid items* for that string. A schema is said to be *sound* if all valid final items it produces for any arbitrary string are correct for that string. A schema is said to be *complete* if all correct final items are valid. A *correct* parsing schema is one which is both sound and complete.

In constituency-based parsing schemata, deduction steps usually have grammar rules as side conditions. In the case of dependency parsers it is also possible to use grammars (Eisner and Satta, 1999), but many algorithms use a data-driven approach instead, making individual decisions about which dependencies to create by using probabilistic models (Eisner, 1996) or classifiers (Yamada and Matsumoto, 2003). To represent these algorithms as deduction systems, we use the notion of *D-rules* (Covington, 1990). D-rules take the form $a \rightarrow b$, which says that word b can have a as a dependent. Deduction steps in non-grammar-based parsers can be tied to the D-rules associated with the links they create. In this way, we obtain a representation of the underlying logic of the parser while abstracting away from control structures (the particular model used to create the decisions associated with D-rules). Furthermore, the choice points in the parsing process and the information we can use to make decisions are made explicit in the steps linked to D-rules.

3 The *WG*$_1$ parser

3.1 Parsing schema for *WG*$_1$

We define *WG*$_1$, a parser for well-nested dependency structures of gap degree ≤ 1, as follows:

The item set is $\mathcal{I}_{WG1} = \mathcal{I}_1 \cup \mathcal{I}_2$, with

$$\mathcal{I}_1 = \{[i, j, h, \diamond, \diamond] \mid i, j, h \in \mathbb{N}, 1 \leq h \leq n, \\ 1 \leq i \leq j \leq n, h \neq j, h \neq i - 1\},$$

where each item of the form $[i, j, h, \diamond, \diamond]$ represents the set of all well-nested partial dependency trees[3] with gap degree at most 1, rooted at w_h, and such that $\lfloor w_h \rfloor = \{w_h\} \cup [i, j]$, and

$$\mathcal{I}_2 = \{[i, j, h, l, r] \mid i, j, h, l, r \in \mathbb{N}, 1 \leq h \leq n, \\ 1 \leq i < l \leq r < j \leq n, h \neq j, h \neq i - 1, \\ h \neq l - 1, h \neq r\}$$

where each item of the form $[i, j, h, l, r]$ represents the set of all well-nested partial dependency trees rooted at w_h such that $\lfloor w_h \rfloor = \{w_h\} \cup ([i, j] \setminus [l, r])$, and all the nodes (except possibly h) have gap degree at most 1. We call items of this form *gapped items*, and the interval $[l, r]$ the *gap* of the item. Note that the constraints $h \neq j, h \neq i + 1, h \neq l - 1, h \neq r$ are added to items to avoid redundancy in the item set. Since the result of the expression $\{w_h\} \cup ([i, j] \setminus [l, r])$ for a given head can be the same for different sets of values of i, j, l, r, we restrict these values so that we cannot get two different items representing the same dependency structures. Items ι violating these constraints always have an alternative representation that does not violate them, that we can express with a normalising function $nm(\iota)$ as follows:

$nm([i, j, j, l, r]) = [i, j - 1, j, l, r]$ (if $r \leq j - 1$ or $r = \diamond$),
or $[i, l - 1, j, \diamond, \diamond]$ (if $r = j - 1$).
$nm([i, j, l - 1, l, r]) = [i, j, l - 1, l - 1, r]$(if $l > i + 1$),
or $[r + 1, j, l - 1, \diamond, \diamond]$ (if $l = i + 1$).
$nm([i, j, i - 1, l, r]) = [i - 1, j, i - 1, l, r]$.
$nm([i, j, r, l, r]) = [i, j, r, l, r - 1]$ (if $l < r$),
or $[i, j, r, \diamond, \diamond]$ (if $l = r$).

$nm([i, j, h, l, r]) = [i, j, h, l, r]$ for all other items.

When defining the deduction steps for this and other parsers, we assume that they always produce normalised items. For clarity, we do not explicitly write this in the deduction steps, writing ι instead of $nm(\iota)$ as antecedents and consequents of steps.

The set of initial items is defined as the set

$$\mathcal{H} = \{[h, h, h, \diamond, \diamond] \mid h \in \mathbb{N}, 1 \leq h \leq n\},$$

where each item $[h, h, h, \diamond, \diamond]$ represents the set containing the trivial partial dependency tree consisting of a single node w_h and no links. This same set of hypotheses can be used for all the parsers, so we do not make it explicit for subsequent schemata. Note that initial items are separate from the item set $\mathcal{I}_{WG1}$ and not subject to its constraints, so they do not require normalisation.

The set of final items for strings of length n in *WG*$_1$ is defined as the set

$$\mathcal{F} = \{[1, n, h, \diamond, \diamond] \mid h \in \mathbb{N}, 1 \leq h \leq n\},$$

which is the set of items in $\mathcal{I}_{WG1}$ containing dependency trees for the complete input string (from position 1 to n), with their head at any word w_h.

The deduction steps of the parser can be seen in Figure 1A.

The *WG*$_1$ parser proceeds bottom-up, by building dependency subtrees and joining them to form larger subtrees, until it finds a complete dependency tree for the input sentence. The logic of

[3]In this and subsequent schemata, we use D-rules to express parsing decisions, so partial dependency trees are assumed to be taken from the set of trees licensed by a set of D-rules.

A. *WG*$_1$ parser:

Link Ungapped: $\dfrac{\begin{array}{c}[h1,h1,h1,\diamond,\diamond]\\ [i2,j2,h2,\diamond,\diamond]\end{array}}{[i2,j2,h1,\diamond,\diamond]}\ w_{h2}\rightarrow w_{h1}$
such that $w_{h2}\in[i2,j2]\wedge w_{h1}\notin[i2,j2]$,

Link Gapped: $\dfrac{\begin{array}{c}[h1,h1,h1,\diamond,\diamond]\\ [i2,j2,h2,l2,r2]\end{array}}{[i2,j2,h1,l2,r2]}\ w_{h2}\rightarrow w_{h1}$
such that $w_{h2}\in[i2,j2]\setminus[l2,r2]\wedge w_{h1}\notin[i2,j2]\setminus[l2,r2]$,

Combine Ungapped: $\dfrac{[i,j,h,\diamond,\diamond]\qquad[j+1,k,h,\diamond,\diamond]}{[i,k,h,\diamond,\diamond]}$

Combine Opening Gap: $\dfrac{[i,j,h,\diamond,\diamond]\qquad[k,l,h,\diamond,\diamond]}{[i,l,h,j+1,k-1]}$
such that $j<k-1$,

Combine Keeping Gap Left:
$\dfrac{[i,j,h,l,r]\qquad[j+1,k,h,\diamond,\diamond]}{[i,k,h,l,r]}$

Combine Keeping Gap Right:
$\dfrac{[i,j,h,\diamond,\diamond]\qquad[j+1,k,h,l,r]}{[i,k,h,l,r]}$

Combine Closing Gap:
$\dfrac{[i,j,h,l,r]\qquad[l,r,h,\diamond,\diamond]}{[i,j,h,\diamond,\diamond]}$

Combine Shrinking Gap Left:
$\dfrac{[i,j,h,l,r]\qquad[l,k,h,\diamond,\diamond]}{[i,j,h,k+1,r]}$

Combine Shrinking Gap Right:
$\dfrac{[i,j,h,l,r]\qquad[k,r,h,\diamond,\diamond]}{[i,j,h,l,k-1]}$

Combine Shrinking Gap Centre:
$\dfrac{[i,j,h,l,r]\qquad[l,r,h,l2,r2]}{[i,j,h,l2,r2]}$

B. *WG*$_K$ parser:

Link: $\dfrac{\begin{array}{c}[h1,h1,h1,[]]\\ [i2,j2,h2,[(l_1,r_1),\dots,(l_g,r_g)]]\end{array}}{[i2,j2,h1,[(l_1,r_1),\dots,(l_g,r_g)]]}\ w_{h2}\rightarrow w_{h1}$
such that $w_{h2}\in[i2,j2]\setminus\bigcup_{p=1}^{g}[l_p,r_p]$
$\wedge w_{h1}\notin[i2,j2]\setminus\bigcup_{p=1}^{g}[l_p,r_p]$.

Combine Opening Gap:
$\dfrac{\begin{array}{c}[i,l_q-1,h,[(l_1,r_1),\dots,(l_{q-1},r_{q-1})]]\\ [r_q+1,m,h,[(l_{q+1},r_{q+1}),\dots,(l_g,r_g)]]\end{array}}{[i,m,h,[(l_1,r_1),\dots,(l_g,r_g)]]}$
such that $g\le k$ and $l_q\le r_q$,

Combine Keeping Gaps:
$\dfrac{\begin{array}{c}[i,j,h,[(l_1,r_1),\dots,(l_q,r_q)]]\\ [j+1,m,h,[(l_{q+1},r_{q+1}),\dots,(l_g,r_g)]]\end{array}}{[i,m,h,[(l_1,r_1),\dots,(l_g,r_g)]]}$
such that $g\le k$,

Combine Shrinking Gap Right:
$\dfrac{\begin{array}{c}[i,j,h,[(l_1,r_1),\dots,(l_{q-1},r_{q-1}),(l_q,r'),(l_s,r_s),\dots,(l_g,r_g)]]\\ [r_q+1,r',h,[(l_{q+1},r_{q+1}),\dots,(l_{s-1},r_{s-1})]]\end{array}}{[i,j,h,[(l_1,r_1),\dots,(l_g,r_g)]]}$
such that $g\le k$

Combine Shrinking Gap Left:
$\dfrac{\begin{array}{c}[i,j,h,[(l_1,r_1),\dots,(l_q,r_q),(l',r_s),(l_{s+1},r_{s+1}),\dots,(l_g,r_g)]]\\ [l',l_s-1,h,[(l_{q+1},r_{q+1}),\dots,(l_{s-1},r_{s-1})]]\end{array}}{[i,j,h,[(l_1,r_1),\dots,(l_g,r_g)]]}$
such that $g\le k$

Combine Shrinking Gap Centre:
$\dfrac{\begin{array}{c}[i,j,h,[(l_1,r_1),\dots,(l_q,r_q),(l',r'),(l_s,r_s),\dots,(l_g,r_g)]]\\ [l',r',h,[(l_{q+1},r_{q+1}),\dots,(l_{s-1},r_{s-1})]]\end{array}}{[i,j,h,[(l_1,r_1),\dots,(l_g,r_g)]]}$
such that $g\le k$

C. Additional steps to turn *WG*$_1$ into *MG*$_1$:

Combine Interleaving: $\dfrac{[i,j,h,l,r]\qquad[l,k,h,r+1,j]}{[i,k,h,\diamond,\diamond]}$

Combine Interleaving Gap C: $\dfrac{[i,j,h,l,r]\qquad[l,k,h,m,j]}{[i,k,h,m,r]}$
such that $m<r+1$,

Combine Interleaving Gap L: $\dfrac{\begin{array}{c}[i,j,h,l,r]\\ [l,k,h,r+1,u]\end{array}}{[i,k,h,j+1,u]}$
such that $u>j$,

Combine Interleaving Gap R: $\dfrac{\begin{array}{c}[i,j,h,l,r]\\ [k,m,h,r+1,j]\end{array}}{[i,m,h,l,k-1]}$
such that $k>l$.

D. General form of the *MG*$_k$ Combine step:

$$\dfrac{\begin{array}{c}[i_{a_1},i_{a_p+1}-1,h,[(i_{a_1+1},i_{a_2}-1),\dots,(i_{a_{p-1}+1},i_{a_p}-1)]]\\ [i_{b_1},i_{b_q+1}-1,h,[(i_{b_1+1},i_{b_2}-1),\dots,(i_{b_{q-1}+1},i_{b_q}-1)]]\end{array}}{[i_{min(a_1,b_1)},i_{max(a_p+1,b_q+1)}-1,h,[(i_{g_1},i_{g_1+1}-1),\dots,(i_{g_r},i_{g_r+1}-1)]]}$$

for each string of length n with a's located at positions $a_1\dots a_p(1\le a_1<\dots<a_p\le n)$, b's at positions $b_1\dots b_q(1\le b_1<\dots<b_q\le n)$, and g's at positions $g_1\dots g_r(2\le g_1<\dots<g_r\le n-1)$, such that $1\le p\le k$, $1\le q\le k$, $0\le r\le k-1$, $p+q+r=n$, and the string does not contain more than one consecutive appearance of the same symbol.

Figure 1: Deduction steps for the parsers defined in the paper.

the parser can be understood by considering how it infers the item corresponding to the subtree induced by a particular node, given the items for the subtrees induced by the direct dependents of that node. Suppose that, in a complete dependency analysis for a sentence $w_1\dots w_n$, the word w_h has $w_{d_1}\dots w_{d_p}$ as direct dependents (i.e. we have dependency links $w_{d_1}\rightarrow w_h,\dots,w_{d_p}\rightarrow w_h$). Then, the item corresponding to the subtree induced by w_h is obtained from the ones corresponding to the subtrees induced by $w_{d_1}\dots w_{d_p}$ by: (1) applying the *Link Ungapped* or *Link Gapped* step to each of the items corresponding to the subtrees induced by the direct dependents, and to the hypothesis $[h,h,h,\diamond,\diamond]$. This allows us to infer p items representing the result of linking each of the dependent subtrees to the new head w_h; (2) applying the various *Combine* steps to join all of the items obtained in the previous step into a single item. The *Combine* steps perform a union operation between subtrees. Therefore, the result is a dependency tree containing all the dependent subtrees, and with all of them linked to h: this is the subtree induced by w_h. This process is applied repeatedly to build larger subtrees, until, if the parsing process is successful, a final item is found containing a dependency tree for the complete sentence.

3.2 Proving correctness

The parsing schemata formalism can be used to prove the correctness of a parsing schema. To prove that *WG*$_1$ is correct, we need to prove its soundness and completeness.[4] Soundness is proven by checking that valid items always contain well-nested trees. Completeness is proven by induction, taking initial items as the base case and showing that an item containing a correct subtree for a string can always be obtained from items corresponding to smaller subtrees. In order to prove this induction step, we use the concept of **order annotations** (Kuhlmann, 2007; Kuhlmann and Möhl, 2007), which are strings that lexicalise the precedence relation between the nodes of a dependency tree. Given a correct subtree, we divide the proof into cases according to the order annotation of its head and we find that, for every possible form of this order annotation, we can find a sequence of *Combine* steps to infer the relevant item from smaller correct items.

3.3 Computational complexity

The time complexity of *WG*$_1$ is $O(n^7)$, as the step *Combine Shrinking Gap Centre* works with 7 free string positions. This complexity with respect to the length of the input is as expected for this set of structures, since Kuhlmann (2007) shows that they are equivalent to LTAG, and the best existing parsers for this formalism also perform in $O(n^7)$ (Eisner and Satta, 2000). Note that the *Combine* step which is the bottleneck only uses the 7 indexes, and not any other entities like D-rules, so its $O(n^7)$ complexity does not have any additional factors due to grammar size or other variables. The space complexity of *WG*$_1$ is $O(n^5)$ for recognition, due to the 5 indexes in items, and $O(n^7)$ for full parsing.

[4]Due to space constraints, correctness proofs for the parsers are not given here. Full proofs are provided in the extended version of this paper, see (Gómez-Rodríguez et al., 2008b).

It is possible to build a variant of this parser with time complexity $O(n^6)$, as with parsers for unlexicalised TAG, if we work with unlexicalised D-rules specifying the possibility of dependencies between pairs of categories instead of pairs of words. In order to do this, we expand the item set with unlexicalised items of the form $[i, j, C, l, r]$, where C is a category, apart from the existing items $[i, j, h, l, r]$. Steps in the parser are duplicated, to work both with lexicalised and unlexicalised items, except for the *Link* steps, which always work with a lexicalised item and an unlexicalised hypothesis to produce an unlexicalised item, and the *Combine Shrinking Gap* steps, which can work only with unlexicalised items. Steps are added to obtain lexicalised items from their unlexicalised equivalents by binding the head to particular string positions. Finally, we need certain variants of the *Combine Shrinking Gap* steps that take 2 unlexicalised antecedents and produce a lexicalised consequent; an example is the following:

$$\textit{Combine Shrinking Gap Centre L: } \frac{[i,j,C,l,r] \quad [l+1,r,C,l2,r2]}{[i,j,l,l2,r2]}$$

such that $cat(w_l){=}C$

Although this version of the algorithm reduces time complexity with respect to the length of the input to $O(n^6)$, it also adds a factor related to the number of categories, as well as constant factors due to using more kinds of items and steps than the original *WG*$_1$ algorithm. This, together with the advantages of lexicalised dependency parsing, may mean that the original *WG*$_1$ algorithm is more practical than this version.

4 The *WG*$_k$ parser

The *WG*$_1$ parsing schema can be generalised to obtain a parser for all well-nested dependency structures with gap degree bounded by a constant $k (k \geq 1)$, which we call *WG*$_k$ parser. In order to do this, we extend the item set so that it can contain items with up to k gaps, and modify the deduction steps to work with these multi-gapped items.

4.1 Parsing schema for *WG*$_k$

The item set $\mathcal{I}_{WGk}$ is the set of all $[i, j, h, [(l_1, r_1), \ldots, (l_g, r_g)]]$ where $i, j, h, g \in \mathbb{N}$, $0 \leq g \leq k$, $1 \leq h \leq n$, $1 \leq i \leq j \leq n$, $h \neq j$, $h \neq i - 1$; and for each $p \in \{1, 2, \ldots, g\}$: $l_p, r_p \in \mathbb{N}$, $i < l_p \leq r_p < j$, $r_p < l_{p+1} - 1$, $h \neq l_p - 1$, $h \neq r_p$.

An item $[i, j, h, [(l_1, r_1), \ldots, (l_g, r_g)]]$ represents the set of all well-nested partial dependency trees rooted at w_h such that $\lfloor w_h \rfloor = \{w_h\} \cup ([i,j] \setminus \bigcup_{p=1}^{g} [l_p, r_p])$, where each interval $[l_p, r_p]$ is called a gap. The constraints $h \neq j, h \neq i+1, h \neq l_p - 1, h \neq r_p$ are added to avoid redundancy, and normalisation is defined as in *WG*$_1$. The set of final items is defined as the set $\mathcal{F} = \{[1, n, h, []] \mid h \in \mathbb{N}, 1 \leq h \leq n\}$. Note that this set is the same as in *WG*$_1$, as these are the items that we denoted $[1, n, h, \diamond, \diamond]$ in the previous parser.

The deduction steps can be seen in Figure 1B. As expected, the *WG*$_1$ parser corresponds to *WG*$_k$ when we make $k = 1$. *WG*$_k$ works in the same way as *WG*$_1$, except for the fact that *Combine* steps can create items with more than one gap[5]. The correctness proof is also analogous to that of *WG*$_1$, but we must take into account that the set of possible order annotations is larger when $k > 1$, so more cases arise in the completeness proof.

4.2 Computational complexity

The *WG*$_k$ parser runs in time $O(n^{5+2k})$: as in the case of *WG*$_1$, the deduction step with most free variables is *Combine Shrinking Gap Centre*, and in this case it has $5 + 2k$ free indexes. Again, this complexity result is in line with what could be expected from previous research in constituency parsing: Kuhlmann (2007) shows that the set of well-nested dependency structures with gap degree at most k is closely related to coupled context-free grammars in which the maximal rank of a nonterminal is $k + 1$; and the constituency parser defined by Hotz and Pitsch (1996) for these grammars also adds an n^2 factor for each unit increment of k. Note that a small value of k should be enough to cover the vast majority of the non-projective sentences found in natural language treebanks. For example, the Prague Dependency Treebank contains no structures with gap degree greater than 4. Therefore, a *WG*$_4$ parser would be able to analyse all the well-nested structures in this treebank, which represent 99.89% of the total. Increasing k beyond 4 would not produce further improvements in coverage.

5 Parsing ill-nested structures

The *WG*$_k$ parser analyses dependency structures with bounded gap degree as long as they are well-nested. This covers the vast majority of the structures that occur in natural-language treebanks (Kuhlmann and Nivre, 2006), but there is still a significant minority of sentences that contain ill-nested structures. Unfortunately, the general problem of parsing ill-nested structures is NP-complete, even when the gap degree is bounded: this set of structures is closely related to LCFRS with bounded fan-out and unbounded production length, and parsing in this formalism has been proven to be NP-complete (Satta, 1992). The reason for this high complexity is the problem of *unrestricted crossing configurations*, appearing when dependency subtrees are allowed to interleave in every possible way. However, just as it has been noted that most non-projective structures appearing in practice are only "slightly" non-projective (Nivre and Nilsson, 2005), we characterise a sense in which the structures appearing in treebanks can be viewed as being only "slightly" ill-nested. In this section, we generalise the algorithms *WG*$_1$ and *WG*$_k$ to parse a proper superset of the set of well-nested structures in polynomial time; and give a characterisation of this new set of structures, which includes all the structures in several dependency treebanks.

5.1 The *MG*$_1$ and *MG*$_k$ parsers

The *WG*$_k$ parser presented previously is based on a bottom-up process, where *Link* steps are used to link completed subtrees to a head, and *Combine* steps are used to join subtrees governed by a common head to obtain a larger structure. As *WG*$_k$ is a parser for well-nested structures of gap degree up to k, its *Combine* steps correspond to all the ways in which we can join two sets of sibling subtrees meeting these constraints, and having a common head, into another. Thus, this parser does not use *Combine* steps that produce interleaved subtrees, since these would generate items corresponding to ill-nested structures.

We obtain a polynomial parser for a wider set of structures of gap degree at most k, including some ill-nested ones, by having *Combine* steps representing every way in which two sets of sibling subtrees of gap degree at most k with a common head can be joined into another, including those producing interleaved subtrees, like the steps for gap degree 1 shown in Figure 1C. Note that this does not mean that we can build every possible ill-nested structure: some structures with complex crossed configurations have gap degree k, but cannot be built by combining two structures of that gap degree. More specifically, our algorithm will be able

[5]In all the parsers in this paper, *Combine* steps may be applied in different orders to produce the same result, causing spurious ambiguity. In *WG*$_1$ and *WG*$_k$, this can be avoided when implementing the schemata, by adding flags to items so as to impose a particular order.

to parse a dependency structure (well-nested or not) if there exists a *binarisation* of that structure that has gap degree at most k. The parser implicitly works by finding such a binarisation, since *Combine* steps are always applied to two items and no intermediate item generated by them can exceed gap degree k (not counting the position of the head in the projection).

More formally, let T be a dependency structure for the string $w_1 \dots w_n$. A **binarisation** of T is a dependency tree T' over a set of nodes, each of which may be unlabelled or labelled with a word in $\{w_1 \dots w_n\}$, such that the following conditions hold: (1) each node has at most two children, and (2) $w_i \rightarrow w_j$ in T if and only if $w_i \rightarrow^\star w_j$ in T'. A dependency structure is **mildly ill-nested** for gap degree k if it has at least one binarisation of gap degree $\leq k$. Otherwise, we say that it is **strongly ill-nested** for gap degree k. It is easy to prove that the set of mildly ill-nested structures for gap degree k includes all well-nested structures with gap degree up to k.

We define *MG*$_1$, a parser for mildly ill-nested structures for gap degree 1, as follows: (1) the item set is the same as that of *WG*$_1$, except that items can now contain any mildly ill-nested structures for gap degree 1, instead of being restricted to well-nested structures; and (2) deduction steps are the same as in *WG*$_1$, plus the additional steps shown in Figure 1C. These extra *Combine* steps allow the parser to combine interleaved subtrees with simple crossing configurations. The *MG*$_1$ parser still runs in $O(n^7)$, as these new steps do not use more than 7 string positions.

The proof of correctness for this parser is similar to that of *WG*$_1$. Again, we use the concept of order annotations. The set of mildly ill-nested structures for gap degree k can be defined as those that only contain annotations meeting certain constraints. The soundness proof involves showing that *Combine* steps always generate items containing trees with such annotations. Completeness is proven by induction, by showing that if a subtree is mildly ill-nested for gap degree k, an item for it can be obtained from items for smaller subtrees by applying *Combine* and *Link* steps. In the cases where *Combine* steps have to be applied, the order in which they may be used to produce a subtree can be obtained from its head's order annotation.

To generalise this algorithm to mildly ill-nested structures for gap degree k, we need to add a *Combine* step for every possible way of joining two structures of gap degree at most k into another. This can be done systematically by considering a set of strings over an alphabet of three symbols: a and b to represent intervals of words in the projection of each of the structures, and g to represent intervals that are not in the projection of either structure, and will correspond to gaps in the joined structure. The legal combinations of structures for gap degree k will correspond to strings where symbols a and b each appear at most $k+1$ times, g appears at most k times and is not the first or last symbol, and there is no more than one consecutive appearance of any symbol. Given a string of this form, the corresponding *Combine* step is given by the expression in Figure 1D. As a particular example, the *Combine Interleaving Gap C* step in Figure 1C is obtained from the string $abgab$.

Thus, we define the parsing schema for *MG*$_k$, a parser for mildly ill-nested structures for gap degree k, as the schema where (1) the item set is like that of *WG*$_k$, except that items can now contain any mildly ill-nested structures for gap degree k, instead of being restricted to well-nested structures; and (2) the set of deduction steps consists of a *Link* step as the one in *WG*$_k$, plus a set of *Combine* steps obtained as expressed in Figure 1D.

As the string used to generate a *Combine* step can have length at most $3k+2$, and the resulting step contains an index for each symbol of the string plus two extra indexes, the *MG*$_k$ parser has complexity $O(n^{3k+4})$. Note that the item and deduction step sets of an *MG*$_k$ parser are always supersets of those of *WG*$_k$. In particular, the steps for *WG*$_k$ are those obtained from strings that do not contain $abab$ or $baba$ as a scattered substring.

5.2 Mildly ill-nested dependency structures

The *MG*$_k$ algorithm defined in the previous section can parse any mildly ill-nested structure for a given gap degree k in polynomial time. We have characterised the set of mildly ill-nested structures for gap degree k as those having a binarisation of gap degree $\leq k$. Since a binarisation of a dependency structure cannot have lower gap degree than the original structure, this set only contains structures with gap degree at most k. Furthermore, by the relation between *MG*$_k$ and *WG*$_k$, we know that it contains all the well-nested structures with gap degree up to k.

Figure 2 shows an example of a structure that has gap degree 1, but is strongly ill-nested for gap degree 1. This is one of the smallest possible such structures: by generating all the possible trees up to 10 nodes (without counting a dummy root node

Language	Structures								
	Total	Nonprojective							
		Total	By gap degree				By nestedness		
			Gap degree 1	Gap degree 2	Gap degree 3	Gap deg. > 3	Well-Nested	Mildly Ill-Nested	Strongly Ill-Nested
Arabic	2995	205	189	13	2	1	204	1	0
Czech	87889	20353	19989	359	4	1	20257	96	0
Danish	5430	864	854	10	0	0	856	8	0
Dutch	13349	4865	4425	427	13	0	4850	15	0
Latin	3473	1743	1543	188	10	2	1552	191	0
Portuguese	9071	1718	1302	351	51	14	1711	7	0
Slovene	1998	555	443	81	21	10	550	5	0
Swedish	11042	1079	1048	19	7	5	1008	71	0
Turkish	5583	685	656	29	0	0	665	20	0

Table 1: Counts of dependency trees classified by gap degree, and mild and strong ill-nestedness (for their gap degree); appearing in treebanks for Arabic (Hajič et al., 2004), Czech (Hajič et al., 2006), Danish (Kromann, 2003), Dutch (van der Beek et al., 2002), Latin (Bamman and Crane, 2006), Portuguese (Afonso et al., 2002), Slovene (Džeroski et al., 2006), Swedish (Nilsson et al., 2005) and Turkish (Oflazer et al., 2003; Atalay et al., 2003).

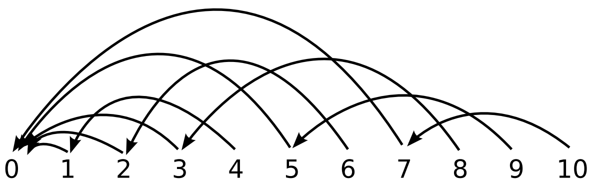

Figure 2: One of the smallest strongly ill-nested structures. This dependency structure has gap degree 1, but is only mildly ill-nested for gap degree ≥ 2.

located at position 0), it can be shown that all the structures of any gap degree k with length smaller than 10 are well-nested or only mildly ill-nested for that gap degree k.

Even if a structure T is strongly ill-nested for a given gap degree, there is always some $m \in \mathbb{N}$ such that T is mildly ill-nested for m (since every dependency structure can be binarised, and binarisations have finite gap degree). For example, the structure in Figure 2 is mildly ill-nested for gap degree 2. Therefore, *MG*$_k$ parsers have the property of being able to parse any possible dependency structure as long as we make k large enough.

In practice, structures like the one in Figure 2 do not seem to appear in dependency treebanks. We have analysed treebanks for nine different languages, obtaining the data presented in Table 1. None of these treebanks contain structures that are strongly ill-nested for their gap degree. Therefore, in any of these treebanks, the *MG*$_k$ parser can parse every sentence with gap degree at most k.

6 Conclusions and future work

We have defined a parsing algorithm for well-nested dependency structures with bounded gap degree. In terms of computational complexity, this algorithm is comparable to the best parsers for related constituency-based formalisms: when the gap degree is at most 1, it runs in $O(n^7)$, like the fastest known parsers for LTAG, and can be made $O(n^6)$ if we use unlexicalised dependencies. When the gap degree is greater than 1, the time complexity goes up by a factor of n^2 for each extra unit of gap degree, as in parsers for coupled context-free grammars. Most of the non-projective sentences appearing in treebanks are well-nested and have a small gap degree, so this algorithm directly parses the vast majority of the non-projective constructions present in natural languages, without requiring the construction of a constituency grammar as an intermediate step.

Additionally, we have defined a set of structures for any gap degree k which we call mildly ill-nested. This set includes ill-nested structures verifying certain conditions, and can be parsed in $O(n^{3k+4})$ with a variant of the parser for well-nested structures. The practical interest of mildly ill-nested structures can be seen in the data obtained from several dependency treebanks, showing that all of the ill-nested structures in them are mildly ill-nested for their corresponding gap degree. Therefore, our $O(n^{3k+4})$ parser can analyse all the gap degree k structures in these treebanks.

The set of mildly ill-nested structures for gap degree k is defined as the set of structures that have a binarisation of gap degree at most k. This definition is directly related to the way the *MG*$_k$ parser works, since it implicitly finds such a binarisation. An interesting line of future work would be to find an equivalent characterisation of mildly ill-nested structures which is more grammar-oriented and would provide a more linguistic insight into these structures. Another research direction, which we are currently working on, is exploring how variants of the *MG*$_k$ parser's strategy can be applied to the problem of binarising LCFRS (Gómez-Rodríguez et al., 2009).

References

Susana Afonso, Eckhard Bick, Renato Haber, and Diana Santos. 2002. “Floresta sintá(c)tica”: a treebank for Portuguese. In *Proc. of LREC 2002*, pages 1968–1703, Las Palmas, Spain.

Nart B. Atalay, Kemal Oflazer, and Bilge Say. 2002. The annotation process in the Turkish treebank. In *Proc. of EACL Workshop on Linguistically Interpreted Corpora - LINC*, Budapest, Hungary.

David Bamman and Gregory Crane. 2006. The design and use of a Latin dependency treebank. In *Proc. of 5th Workshop on Treebanks and Linguistic Theories (TLT2006)*, pages 67–78.

Manuel Bodirsky, Marco Kuhlmann, and Mathias Möhl. 2005. Well-nested drawings as models of syntactic structure. Technical Report, Saarland University. Electronic version available at: `http://www.ps.uni-sb.de/Papers/`.

Michael A. Covington. 1990. A dependency parser for variable-word-order languages. Technical Report AI-1990-01, Athens, GA.

Sašo Džeroski, Tomaž Erjavec, Nina Ledinek, Petr Pajas, Zdeněk Žabokrtský, and Andreja Žele. 2006. Towards a Slovene dependency treebank. In *Proc. of LREC 2006*, pages 1388–1391, Genoa, Italy.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proc. of ACL-99*, pages 457–464, Morristown, NJ. ACL.

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

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of COLING-96*, pages 340–345, Copenhagen.

Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2008a. A deductive approach to dependency parsing. In *Proc. of ACL'08:HLT*, pages 968–976, Columbus, Ohio. ACL.

Carlos Gómez-Rodríguez, David Weir, and John Carroll. 2008b. Parsing mildly non-projective dependency structures. Technical Report CSRP 600, Department of Informatics, University of Sussex.

Carlos Gómez-Rodríguez, Marco Kuhlmann, Giorgio Satta, and David Weir. 2009. Optimal reduction of rule length in linear context-free rewriting systems. In *Proc. of NAACL'09:HLT* (to appear).

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

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

Jiří Havelka. 2007. Beyond projectivity: Multilingual evaluation of constraints and measures on non-projective structures. In *Proc. of ACL 2007*, Prague, Czech Republic. ACL.

Günter Hotz and Gisela Pitsch. 1996. On parsing coupled-context-free languages. *Theor. Comput. Sci.*, 161(1-2):205–233. Elsevier, Essex, UK.

Aravind K. Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In *Handbook of formal languages*, pages 69–124. Springer-Verlag, Berlin/Heidelberg/NY.

Matthias T. Kromann. 2003. The Danish dependency treebank and the underlying linguistic theory. In *Proc. of the 2nd Workshop on Treebanks and Linguistic Theories (TLT2003)*.

Marco Kuhlmann and Mathias Möhl. 2007. Mildly context-sensitive dependency languages. In *Proc. of ACL 2007*, Prague, Czech Republic. ACL.

Marco Kuhlmann and Joakim Nivre. 2006. Mildly non-projective dependency structures. In *Proc. of COLING/ACL main conference poster sessions*, pages 507–514, Morristown, NJ, USA. ACL.

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

Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *IWPT 2007: Proc. of the 10th Conference on Parsing Technologies*. ACL.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of HLT/EMNLP 2005*, pages 523–530, Morristown, NJ, USA. ACL.

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

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proc. of ACL'05*, pages 99–106, Morristown, NJ, USA. ACL.

Kemal Oflazer, Bilge Say, Dilek Zeynep Hakkani-Tür and Gökhan Tür. 2003. Building a Turkish treebank. In A. Abeille, ed., *Building and Exploiting Syntactically-annotated Corpora*. Kluwer, Dordrecht.

Giorgio Satta. 1992. Recognition of linear context-free rewriting systems. In *Proc. of ACL-92*, pages 89–95, Morristown, NJ. ACL.

Klaas Sikkel. 1997. *Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms*. Springer-Verlag, Berlin/Heidelberg/NY.

L. van der Beek, G. Bouma, R. Malouf, and G. van Noord. 2002. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*, Twente University.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proc. of ACL-87*, pages 104–111, Morristown, NJ. ACL.

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