Dynamic Programming for Higher Order Parsing of Gap-Minding Trees

Emily Pitler, Sampath Kannan, Mitchell Marcus
Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
epitler,kannan,mitch@seas.upenn.edu

Abstract

We introduce *gap inheritance*, a new structural property on trees, which provides a way to quantify the degree to which intervals of descendants can be nested. Based on this property, two new classes of trees are derived that provide a closer approximation to the set of plausible natural language dependency trees than some alternative classes of trees: unlike projective trees, a word can have descendants in more than one interval; unlike spanning trees, these intervals cannot be nested in arbitrary ways. The *1-Inherit* class of trees has *exactly* the same empirical coverage of natural language sentences as the class of mildly non-projective trees, yet the optimal scoring tree can be found in an order of magnitude less time. *Gap-minding trees* (the second class) have the property that all edges into an interval of descendants come from the same node, and thus an algorithm which uses only single intervals can produce trees in which a node has descendants in multiple intervals.

1 Introduction

Dependency parsers vary in what *space of possible tree structures* they search over when parsing a sentence. One commonly used space is the set of *projective* trees, in which every node's descendants form a contiguous interval in the input sentence. Finding the optimal tree in the set of projective trees can be done efficiently (Eisner, 2000), even when the score of a tree depends on higher order factors (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010). However, the projectivity assumption is too strict for all natural language dependency trees; for example, only 63.6% of Dutch sentences from the CoNLL-X training set are projective (Table 1).

At the other end of the spectrum, some parsers search over all *spanning trees*, a class of structures much larger than the set of plausible linguistic structures. The maximum scoring directed spanning tree can be found efficiently when the score of a tree depends only on edge-based factors (McDonald et al., 2005b). However, it is NP-hard to extend MST to include sibling or grandparent factors (McDonald and Pereira, 2006; McDonald and Satta, 2007). MST-based non-projective parsers that use higher order factors (Martins et al., 2009; Koo et al., 2010), utilize different techniques than the basic MST algorithm. In addition, learning is done over a relaxation of the problem, so the inference procedures at training and at test time are not identical.

We propose two new classes of trees between projective trees and the set of all spanning trees. These two classes provide a closer approximation to the set of plausible natural language dependency trees: unlike projective trees, a word can have descendants in more than one interval; unlike spanning trees, these intervals cannot be nested in arbitrary ways. We introduce *gap inheritance*, a new structural property on trees, which provides a way to quantify the degree to which these intervals can be nested. Different levels of gap inheritance define each of these two classes (Section 3).

The *1-Inherit* class of trees (Section 4) has *exactly* the same empirical coverage (Table 1) of natural language sentences as the class of *mildly non-projective trees* (Bodirsky et al., 2005), yet the optimal scoring tree can be found in an order of magnitude less time (Section 4.1).

Gap-minding trees (the second class) have the

Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 478–488, Jeju Island, Korea, 12–14 July 2012.

property that all edges into an interval of descendants come from the same node. Non-contiguous intervals are therefore *decoupled* given this single node, and thus an algorithm which uses only single intervals (as in projective parsing) can produce trees in which a node has descendants in multiple intervals (as in mildly non-projective parsing (Gómez-Rodríguez et al., 2011)). A procedure for finding the optimal scoring tree in this space is given in Section 5, which can be searched in yet another order of magnitude faster than the *1-Inherit* class.

Unlike the class of spanning trees, it is still tractable to find the optimal tree in these new spaces when higher order factors are included. An extension which finds the optimal scoring gap-minding tree with scores over pairs of adjacent edges (grandparent scoring) is given in Section 6. These gap-minding algorithms have been implemented in practice and empirical results are presented in Section 7.

2 Preliminaries

In this section, we review some relevant definitions from previous work that characterize degrees of non-projectivity. We also review how well these definitions cover empirical data from six languages: Arabic, Czech, Danish, Dutch, Portuguese, and Swedish. These are the six languages whose CoNLL-X shared task data are either available open source[1] or from the LDC[2].

A *dependency tree* is a rooted, directed spanning tree that represents a set of dependencies between words in a sentence.[3] The tree has one artificial root node and vertices that correspond to the words in an input sentence w_1, w_2,...,w_n. There is an edge from h to m if m depends on (or modifies) h.

Definition 1. *The* projection *of a node is the set of words in the subtree rooted at it (including itself).*

A tree is *projective* if, for every node in the tree, that node's projection forms a contiguous interval in the input sentence order.

A tree is *non-projective* if the above does not hold, i.e., there exists at least one word whose descendants do not form a contiguous interval.

Definition 2. *A* gap *of a node v is a non-empty, maximal interval that does not contain any words in the projection of v but lies between words that are in the projection of v. The* gap degree of a node *is the number of gaps it has. The* gap degree of a tree *is the maximum of the gap degrees of its vertices. (Bodirsky et al., 2005)*

Note that a projective tree will have gap degree 0.

Two subtrees *interleave* if there are vertices l_1, r_1 from one subtree and l_2, r_2 from the other such that $l_1 < l_2 < r_1 < r_2$.

Definition 3. *A tree is* well-nested *if no two* disjoint *subtrees interleave (Bodirsky et al., 2005).*

Definition 4. *A* mildly non-projective *tree has gap degree at most one and is well-nested.*

Mildly non-projective trees are of both theoretical and practical interest, as they correspond to derivations in Lexicalized Tree Adjoining Grammar (Bodirsky et al., 2005) and cover the overwhelming majority of sentences found in treebanks for Czech and Danish (Kuhlmann and Nivre, 2006).

Table 1 shows the proportion of mildly non-projective sentences for Arabic, Czech, Danish, Dutch, Portuguese, and Swedish, ranging from 95.4% of Portuguese sentences to 99.9% of Arabic sentences.[4] This definition covers a substantially larger set of sentences than projectivity does — an assumption of projectivity covers only 63.6% (Dutch) to 90.2% (Swedish) of examples (Table 1).

3 Gap Inheritance

Empirically, natural language sentences seem to be mostly mildly non-projective trees, but mildly non-projective trees are quite expensive to parse ($O(n^7)$ (Gómez-Rodríguez et al., 2011)). The parsing complexity comes from the fact that the definition allows two non-contiguous intervals of a projection to be tightly coupled, with an unbounded number of edges passing back and forth between the two intervals; however, this type of structure seems unusual

[1] `http://ilk.uvt.nl/conll/free_data.html`

[2] LDC catalogue numbers LDC2006E01 and LDC2006E02

[3] Trees are a reasonable assumption for most, but not all, linguistic structures. Parasitic gaps are an example in which a word perhaps should have multiple parents.

[4] While some of the treebank structures are ill-nested or have a larger gap degree because of annotation decisions, some linguistic constructions in German and Czech are ill-nested or require at least two gaps under any reasonable representation (Chen-Main and Joshi, 2010; Chen-Main and Joshi, 2012).

	Arabic		Czech		Danish		Dutch		Portuguese		Swedish		Parsing
Mildly non-proj	1458	(99.9)	72321	(99.5)	5175	(99.7)	12896	(96.6)	8650	(95.4)	10955	(99.2)	$O(n^7)$
Mild+1-Inherit	1458	(99.9)	72321	(99.5)	5175	(99.7)	12896	(96.6)	8650	(95.4)	10955	(99.2)	$O(n^6)$
Mild+0-Inherit	1394	(95.5)	70695	(97.2)	4985	(96.1)	12068	(90.4)	8481	(93.5)	10787	(97.7)	$O(n^5)$
Projective	1297	(88.8)	55872	(76.8)	4379	(84.4)	8484	(63.6)	7353	(81.1)	9963	(90.2)	$O(n^3)$
# Sentences	1460		72703		5190		13349		9071		11042		

Table 1: The number of sentences from the CoNLL-X training sets whose parse trees fall into each of the above classes. The two new classes of structures, *Mild+0-Inherit* and *Mild+1-Inherit*, have more coverage of empirical data than projective structures, yet can be parsed faster than mildly non-projective structures. Parsing times assume an edge-based factorization with no pruning of edges. The corresponding algorithms for Mild+1-Inherit and Mild+0-Inherit are in Sections 4 and 5.

for natural language. We therefore investigate if we can define further structural properties that are both appropriate for describing natural language trees and which admit more efficient parsing algorithms.

Let us first consider an example of a tree which both has gap degree at most one and satisfies well-nestedness, yet appears to be an unrealistic structure for a natural language syntactic tree. Consider a tree which is rooted at node x_{n+2}, which has one child, node x_{n+1}, whose projection is $[x_1, x_{n+1}] \cup [x_{n+3}, x_{2n+2}]$, with n children (x_1, ..., x_n), and each child x_i has a child at x_{2n-i+3}. This tree is well-nested, has gap degree 1, but all n of x_{n+1}'s children have edges into the other projection interval.

We introduce a further structural restriction in this section, and show that trees satisfying our new property can be parsed more efficiently with no drop in empirical coverage.

Definition 5. *A child is* gap inheriting *if its parent has gap degree 1 and it has descendants on both sides of its parent's gap. The* inheritance degree of a node *is the number of its children which inherit its gap. The* inheritance degree of a tree *is the maximum inheritance degree over all its nodes.*

Figure 1 gives examples of trees with varying degrees of gap inheritance. Each projection of a node with a gap is shown with two matching rectangles. If a child has a projection rectangle nested inside each of the parent's projection rectangles, then that child inherits the parent's gap. Figure 1(a) shows a mildly projective tree (with inheritance degree 2), with both node 2 and node 11 inheriting their parent (node 3)'s gap (note that both the dashed and dotted rectangles each show up inside both of the solid rectangles). Figure 1(b) shows a tree with inheritance degree 1: there is now only one pair of rectangles (the dotted ones) which show up in both of the solid ones. Figure 1(c) shows a tree with inheritance degree 0: while there are gaps, each set of matching rectangles is contained within a single rectangle (projection interval) of its parent, i.e., the two dashed rectangles of node 2's projection are contained within the left interval of node 3; the two dotted rectangles of node 12's projection are contained within the right interval of node 3, etc.

We now ask:

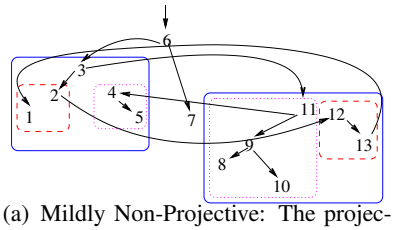
1. How often does gap inheritance occur in the parses of natural language sentences found in treebanks?

2. Furthermore, how often are there *multiple* gap inheriting children of the same node (inheritance degree at least two)?

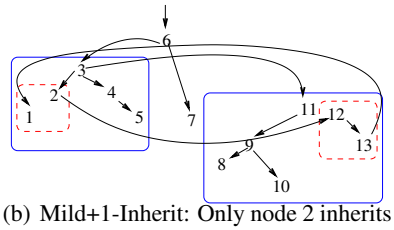
Table 1 shows what proportion of mildly non-projective trees have the added property of gap inheritance degree 0 (Mild+0-Inherit) or have gap inheritance degree 1 (Mild+1-Inherit). Over all six languages, there are *no* examples of multiple gap inheritance — Mild+1-Inherit has exactly the same empirical coverage as the unrestricted set of mildly non-projective trees.

4 Mild+1-Inherit Trees

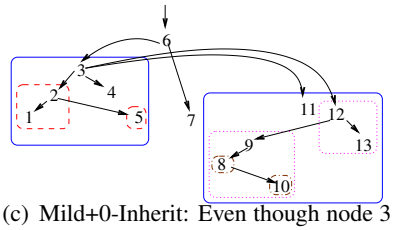
There are some reasons from syntactic theory why we might expect at most one child to inherit its parent's gap. Traditional Government and Binding theories of syntax (Chomsky, 1981) assume that there is an underlying projective (phrase structure) tree, and that gaps primarily arise through movement of



(a) Mildly Non-Projective: The projections (set of descendants) of both node 2 (the dashed red rectangles) and node 11 (dotted magenta) appear in both of node 3's intervals (the solid blue rectangles).



(b) Mild+1-Inherit: Only node 2 inherits node 3's gap: the dashed red rectangles appear in each of the two solid blue rectangles.



(c) Mild+0-Inherit: Even though node 3 has children with gaps (node 2 and node 12), neither of them inherit node 3's gap. There are several nodes with gaps, but every node with a gap is properly contained within just one of its parent's intervals.

Figure 1: Rectangles that match in color and style indicate the two projection intervals of a node, separated by a gap. In all three trees, node 3's two projection intervals are shown in the two solid blue rectangles. The number of children which inherit its gap vary, however; in 1(a), two children have descendants within both sides; in 1(b) only one child has descendants on both sides; in 1(c), none of its children do.

subtrees (constituents). One of the fundamental assumptions of syntactic theory is that movement is upward in the phrase structure tree.[5]

Consider one movement operation and its effect on the gap degree of all other nodes in the tree: (a) it should have no effect on the gap degree of the nodes in the subtree itself, (b) it can create a gap for an ancestor node if it moves *out* of its projection interval, and (c) it can create a gap for a non-ancestor node if it moves *in* to its projection interval. Now consider which cases can lead to gap inheritance: in case (b), there is a single path from the ancestor to the root of the subtree, so the parent of the subtree will have no gap inheritance and any higher ancestors will have a single child inherit the gap created by this movement. In case (c), it is possible for there to be multiple children that inherit this newly created gap if multiple children had descendents on both sides. However, the assumption of upward movement in the phrase structure tree should rule out movement into the projection interval of a non-ancestor. Therefore, under these syntactic assumptions, we would expect at most one child to inherit a parent's gap.

[5]The Proper Binding Condition (Fiengo, 1977) asserts that a moved element leaves behind a *trace* (unpronounced element), which must be *c-commanded* (Reinhart, 1976) by the corresponding pronounced material in its final location. Informally, c-commanded means that the first node is descended from the lowest ancestor of the other that has more than one child.

4.1 Parsing Mild+1-Inherit Trees

Finding the optimal Mild+1-Inherit tree can be done by bottom-up constructing the tree for each node and its descendants. We can maintain subtrees with two intervals (two endpoints each) and one root ($O(n^5)$ space). Consider the most complicated possible case: a parent that has a gap, a (single) child which inherits the gap, and additional children. An example of this is seen with the parent node 3 in Figure 1(b).

This subtree can be constructed by first starting with the child spanning the gap, updating its root index to be the parent, and then expanding the interval indices to the left and right to include the other children. In each case, only one index needs to be updated at a time, so the optimal tree can be found in $O(n^6)$ time. In the Figure 1(b) example, the subtree rooted at 3 would be built by starting with the intervals $[1, 2] \cup [12, 13]$ rooted at 2, first adding the edge from 2 to 3 (so the root is updated to 3), then adding an edge from 3 to 4 to extend the left interval to $[1, 5]$, and then adding an edge from 3 to 11 to extend the right interval to $[8, 13]$. The subtree corresponds to the completed item $[1, 5] \cup [8, 13]$ rooted at 3.

This procedure corresponds to Gómez-Rodríguez et al. (2011)'s $O(n^7)$ algorithm for parsing mildly non-projective structures if the most expensive step (*Combine Shrinking Gap Centre*) is dropped; this step would only ever be needed if a parent node has

more than one child inheriting its gap.

This is also similar in spirit to the algorithm described in Satta and Schuler (1998) for parsing a restricted version of TAG, in which there are some limitations on adjunction operations into the spines of trees.[6] That algorithm has similar steps and items, with the root portion of the item replaced with a node in a phrase structure tree (which may be a non-terminal).

5 Gap-minding Trees

The algorithm in the previous section used $O(n^5)$ space and $O(n^6)$ time. While more efficient than parsing in the space of mildly projective trees, this is still probably not practically implementable. Part of the difficulty lies in the fact that gap inheritance causes the two non-contiguous projection intervals to be coupled.

Definition 6. *A tree is called* gap-minding[7] *if it has gap degree at most one, is well-nested, and has gap inheritance degree 0.*

Gap-minding trees still have good empirical coverage (between 90.4% for Dutch and 97.7% for Swedish). We now turn to the parsing of gap-minding trees and show how a few consequences of its definition allow us to use items ranging over only *one* interval.

In Figure 1(c), notice how each rectangle has edges incoming from *exactly one* node. This is not unique to this example; all projection intervals in a gap-minding tree have incoming edges from exactly one node outside the interval.

Claim 1. *Within a gap-minding tree, consider any node n with a gap (i.e., n's projection forms two non-contiguous intervals $[x_i, x_j] \cup [x_k, x_l]$). Let p be the parent of n.*

1. *For each of the intervals of n's projection:*
   - *(a) If the interval contains n, the only edge incoming to that interval is from p to n.*
   - *(b) If the interval does not contain n, all edges incoming to that interval come from n.*
2. *For the gap interval ($[x_{j+1}, x_{k-1}]$):*
   - *(a) If the interval contains p, then the only edge incoming is from p's parent to p*
   - *(b) If the interval does not contain p, then all edges incoming to that interval come from p.*

As a consequence of the above, $[x_i, x_j] \cup \{n\}$ forms a gap-minding tree rooted at n, $[x_k, x_l] \cup \{n\}$ also forms a gap-minding tree rooted at n, and $[x_{j+1}, x_{k-1}] \cup \{p\}$ forms a gap-minding tree rooted at p.

Proof. (Part 1): Assume there was a directed edge (x, y) such that y is inside a projection interval of n and x is not inside the same interval, and $x \neq y \neq n$. y is a descendant of n since it is contained in n's projection. Since there is a directed edge from x to y, x is y's parent, and thus x must also be a descendant of n and therefore in another of n's projection intervals. Since x and y are in different intervals, then whichever child of n that x and y are descended from would have inherited n's gap, leading to a contradiction.

(Part 2): First, suppose there existed a set of nodes in n's gap which were not descended from p. Then p has a gap over these nodes. (p clearly has descendants on each side of the gap, because all descendants of n are also descendants of p). n, p's child, would then have descendants on both sides of p's gap, which would violate the property of no gap inheritance. It is also not possible for there to be edges incoming from other descendants of p outside the gap, as that would imply another child of p being ill-nested with respect to n. □

From the above, we can build gap-minding trees using only single intervals, potentially with a single node outside of the interval. Our objective is to find the maximum scoring gap-minding tree, in which the score of a tree is the sum of the scores of its edges. Let $\mathbf{Score}(\mathbf{p}, \mathbf{x})$ indicate the score of the directed edge from p to x.

Therefore, the main type of sub-problems we will use are:

[6] That algorithm has a running time of $O(Gn^5)$, where as written G would likely add a factor of n^2 with bilexical selectional preferences; this can be lowered to n using the same technique as in Eisner and Satta (2000) for non-restricted TAG.

[7] The terminology is a nod to the London Underground but imagines parents admonishing children to mind the gap.

1. $\mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{p}]$: The maximum score of any gap-minding tree, rooted at p, with vertices $[i, j] \cup \{p\}$ (p may or may not be within $[i, j]$).

This improves our space requirement, but not necessarily the time requirement. For example, if we built up the subtree in Figure 1(c) by concatenating the three intervals $[1, 5]$ rooted at 3, $[6, 7]$ rooted at 6, and $[8, 13]$ rooted at 3, and add the edge $6 \rightarrow 3$, we would still need 6 indices to describe this operation (the four interval endpoints and the two roots), and so we have not yet improved the running time over the *Inherit-1* case.

By part 2, we can concatenate one interval of a child with its gap, knowing that the gap is entirely descended from the child's parent, and forget the concatenation split point between the parent's other descendants and this side of the child. This allows us to substitute all operations involving 6 indices with two operations involving just 5 indices. For example, in Figure 1(c), we could first merge $[6, 7]$ rooted at 6 with $[8, 13]$ rooted at 3 to create an interval $[6, 13]$ and say that it is descended from 6, with the rightmost side descended from its child 3. That step required 5 indices. The following step would merge this concatenated interval ($[6, 13]$ rooted at 6 and 3) with $[1, 5]$ rooted at 3. This step also requires only 5 indices.

Our helper subtype we make use of is then:

2. $\mathbf{D}[\mathbf{i}, \mathbf{j}, \mathbf{p}, \mathbf{x}, \mathbf{b}]$: The maximum score of any set of two gap-minding trees, one rooted at p, one rooted at x, with vertices $[i, j] \cup \{p, x\}$ ($x \notin [i, j]$, p may or may not be in $[i, j]$), such that for some k, vertices $[i, k]$ are in the tree rooted at p if $b =$ *true* (and at x if $b =$ *false*), and vertices $[k+1, j]$ are in the tree rooted at x (p).

Consider an optimum scoring gap-minding tree T rooted at p with vertices $V = [i, j] \cup \{p\}$ and edges E, where $E \neq \emptyset$. The form of the dynamic program may depend on whether:

- p is within (i, j) **(I)** or external to $[i, j]$ **(E)**[8]

[8]In the discussion we will assume that $p \neq i$ and $p \neq j$, since any optimum solution with $V = [i, j] \cup \{i\}$ and a root at i will be equivalent to $V = [i + 1, j] \cup \{i\}$ rooted at i (and similarly for $p = j$).

We can exhaustively enumerate all possibilities for T by considering all valid combinations of the following binary cases:

- p has a single child **(S)** or multiple children **(M)**
- i and j are descended from the same child of p **(C)** or different children of p **(D)**

Note that case **(S/D)** is not possible: i and j cannot be descended from different children of p if p has only a single child. We therefore need to find the maximum scoring tree over the three cases of **S/C**, **M/C**, and **M/D**.

Claim 2. *Let T be the optimum scoring gap-minding tree rooted at p with vertices $V = [i, j] \cup \{p\}$. Then T and its score are derived from one of the following:*

S/C *If p has a single child x in T, then if $p \in (i, j)$* **(I)**, *T's score is $\mathbf{Score}(\mathbf{p}, \mathbf{x}) + \mathbf{C}[\mathbf{i}, \mathbf{p}-\mathbf{1}, \mathbf{x}] + \mathbf{C}[\mathbf{p}+\mathbf{1}, \mathbf{j}, \mathbf{x}]$; if $p \notin [i, j]$* **(E)**, *T's score is $\mathbf{Score}(\mathbf{p}, \mathbf{x}) + \mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{x}]$.*

M/C *If p has multiple children in T and i and j are descended from the same child x in T, then there is a split point k such that T's score is: $\mathbf{Score}(\mathbf{p}, \mathbf{x}) + \mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{x}] + \mathbf{D}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{p}, \mathbf{x}, \mathbf{T}]$ if x is on the left side of its own gap, and T's score is: $\mathbf{Score}(\mathbf{p}, \mathbf{x}) + \mathbf{C}[\mathbf{k}, \mathbf{j}, \mathbf{x}] + \mathbf{D}[\mathbf{i}, \mathbf{k}-\mathbf{1}, \mathbf{p}, \mathbf{x}, \mathbf{F}]$ if x is on the right side.*

M/D *If p has multiple children in T and i and j are descended from different children in T, then there is a split point k such that T's score is $\mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{p}] + \mathbf{C}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{p}]$.*

T has the maximum score over each of the above cases, for all valid choices of x and k.

Proof. **Case S/C**: If p has exactly one child x, then the tree can be decomposed into the edge from p to x and the subtree rooted at x. If p is outside the interval, then the maximum scoring such tree is clearly $\mathbf{Score}(\mathbf{p}, \mathbf{x}) + \mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{x}]$. If p is inside, then x has a gap across p, and so using Claim 1, the maximum scoring tree rooted at p with a single child x has score of $\mathbf{Score}(\mathbf{p}, \mathbf{x}) + \mathbf{C}[\mathbf{i}, \mathbf{p}-\mathbf{1}, \mathbf{x}] + \mathbf{C}[\mathbf{p}+\mathbf{1}, \mathbf{j}, \mathbf{x}]$.

Case M/C: If there are multiple children and the endpoints are descended from the same child x, then

the child x has to have gap degree 1. x itself is on either the left or right side of its gap. For the moment, assume x is in the left interval. By Claim 1, we can split up the score of the tree as the score of the edge from p to x ($\mathbf{Score}(\mathbf{p}, \mathbf{x})$), the score of the subtree corresponding to the projection of x to the left of its gap ($\mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{x}]$), and the score of the subtrees rooted at p with its remaining children and the subtree rooted at x corresponding to the right side of x's projection ($\mathbf{D}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{p}, \mathbf{x}, \mathbf{T}]$). The case in which x is on the right side of its gap is symmetric.

Case M/D: If there are multiple children and the endpoints are descended from different children of p, then there must exist a split point k that partitions the children of p into two non-empty sets, such that each child's projection is either entirely on the left or entirely on the right of the split point. We show one such split point to demonstrate that there always exists at least one. Let x be the child of p that i is descended from, and let x_l and x_r be x's leftmost and right descendants, respectively.[9] Consider all the children of p (whose projections taken together partition $[i, j] - \{p\}$). No child can have descendants both to the left of x_r and to the right of x_r, because otherwise that child and x would be *ill-nested*. Therefore we can split up the interval at x_r to have two gap-minding trees, both rooted at p. The score of T is then the sum of the scores of the best subtree rooted at p over $[i, k]$ ($\mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{p}]$) and the score of the best subtree rooted at p over $[k+1, j]$ ($\mathbf{C}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{p}]$).

The above cases cover all non-empty gap-minding trees, so the maximum will be found. □

Using Claim 2 to Devise an Algorithm The above claim showed that any problem of type $\mathbf{C}$ can be decomposed into subproblems of types $\mathbf{C}$ and $\mathbf{D}$. From the definition of $\mathbf{D}$, any problem of type $\mathbf{D}$ can clearly be decomposed into two problems of type $\mathbf{C}$ — simply split the interval at the split point known to exist and assign p or x as the roots for each side of the interval, as prescribed by the boolean b:

$$\mathbf{D}(\mathbf{i}, \mathbf{j}, \mathbf{p}, \mathbf{x}, \mathbf{T}) = \mathbf{max_k} \mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{p}] + \mathbf{C}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{x}]$$
$$\mathbf{D}(\mathbf{i}, \mathbf{j}, \mathbf{p}, \mathbf{x}, \mathbf{F}) = \mathbf{max_k} \mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{x}] + \mathbf{C}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{p}]$$

[9]Note that $x_l = i$ by construction, and $x_r \neq j$ (because the endpoints are descended from different children).

Algorithm 1 makes direct use of the above claims. Note that in every gap-minding tree referred to in the cases above, all vertices that were not the root formed a single interval. Algorithm 1 builds up trees in increasing sizes of $[i, j] \cup \{p\}$. The tree in $\mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{p}]$ corresponds to the maximum of four subroutines: SingleChild (**S/C**), EndpointsDiff (**M/D**), EndsFromLeftChild (**M/C**), and EndsFromRightChild (**M/C**). The $\mathbf{D}$ subproblems are filled in with the subroutine Max2Subtrees, which uses the above discussion. The maximum score of any gap-minding tree is then found in $\mathbf{C}[\mathbf{1}, \mathbf{n}, \mathbf{0}]$, and the tree itself can be found using backpointers.

5.1 Runtime analysis

If the input is assumed to be the complete graph (any word can have any other word as its parent), then the above algorithm takes $O(n^5)$ time. The most expensive steps are **M/C**, which take $O(n^2)$ time to fill in each of the $O(n^3)$ $\mathbf{C}$ cells. and solving a $\mathbf{D}$ subproblem, which takes $O(n)$ time on each of the $O(n^4)$ possible such problems.

Pruning: In practice, the set of edges considered (m) is not necessarily $O(n^2)$. Many edges can be ruled out beforehand, either based on the distance in the sentence between the two words (Eisner and Smith, 2010), the predictions of a local ranker (Martins et al., 2009), or the marginals computed from a simpler parsing model (Carreras et al., 2008).

If we choose a pruning strategy such that each word has at most k potential parents (incoming edges), then the running time drops to $O(kn^4)$. The five indices in an **M/C** step were: i, j, k, p, and x. As there must be an edge from p to x, and x only has k possible parents, there are now only $O(kn^4)$ valid such combinations. Similarly, each $\mathbf{D}$ subproblem (which ranges over i, j, k, p, x) may only come into existence because of an edge from p to x, so again the runtime of these such steps drops to $O(kn^4)$.

6 Extension to Grandparent Factorizations

The ability to define slightly non-local features has been shown to improve parsing performance. In this section, we assume a *grandparent-factored* model, where the score of a tree is now the sum over scores of (g, p, c) triples, where (g, p) and (p, c) are both

directed edges in the tree. Let $\mathbf{Score}(\mathbf{g}, \mathbf{p}, \mathbf{c})$ indicate the score of this grandparent-parent-child triple. We now show how to extend the above algorithm to find the maximum scoring gap-minding tree with grandparent scoring.

Our two subproblems are now $\mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{p}, \mathbf{g}]$ and $\mathbf{D}[\mathbf{i}, \mathbf{j}, \mathbf{p}, \mathbf{x}, \mathbf{b}, \mathbf{g}]$; each subproblem has been augmented with an additional grandparent index g, which has the meaning that g is p's parent. Note that g must be outside of the interval $[i, j]$ (if it were not, a cycle would be introduced). Edge scores are now computed over (g, p, x) triples. In particular, claim 2 is modified:

Claim 3. *Let T be the optimum scoring gap-minding tree rooted at p with vertices $V = [i, j] \cup \{p\}$, where $p \in (i, j)$* **(I)***, with a grandparent index g ($g \notin V$). Then T and its score are derived from one of the following:*

S/C *If p has a single child x in T, then if $p \in (i, j)$* **(I)***, T's score is* $\mathbf{Score}(\mathbf{g}, \mathbf{p}, \mathbf{x}) + \mathbf{C}[\mathbf{i}, \mathbf{p}-\mathbf{1}, \mathbf{x}, \mathbf{p}] + \mathbf{C}[\mathbf{p}+\mathbf{1}, \mathbf{j}, \mathbf{x}, \mathbf{p}]$*; if $p \notin [i, j]$* **(E)***, T's score is* $\mathbf{Score}(\mathbf{g}, \mathbf{p}, \mathbf{x}) + \mathbf{C}[\mathbf{i}, \mathbf{j}, \mathbf{x}, \mathbf{p}]$.

M/C *If p has multiple children in T and i and j are descended from the same child x in T, then there is a split point k such that T's score is:* $\mathbf{Score}(\mathbf{g}, \mathbf{p}, \mathbf{x}) + \mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{x}, \mathbf{p}] + \mathbf{D}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{p}, \mathbf{x}, \mathbf{T}, \mathbf{g}]$ *if x is on the left side of its own gap, and T's score is:* $\mathbf{Score}(\mathbf{g}, \mathbf{p}, \mathbf{x}) + \mathbf{C}[\mathbf{k}, \mathbf{j}, \mathbf{x}, \mathbf{p}] + \mathbf{D}[\mathbf{i}, \mathbf{k}-\mathbf{1}, \mathbf{p}, \mathbf{x}, \mathbf{F}, \mathbf{g}]$ *if x is on the right side.*

M/D *If p has multiple children in T and i and j are descended from different children in T, then there is a split point k such that T's score is* $\mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{p}, \mathbf{g}] + \mathbf{C}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{p}, \mathbf{g}]$.

T has the maximum score over each of the above cases, for all valid choices of x and k.

Note that for subproblems rooted at p, g is the grandparent index, while for subproblems rooted at x, p is the updated grandparent index. The $\mathbf{D}$ subproblems with the grandparent index are shown below:

$$\mathbf{D}(\mathbf{i}, \mathbf{j}, \mathbf{p}, \mathbf{x}, \mathbf{T}, \mathbf{g}) = \mathbf{max_k} \mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{p}, \mathbf{g}] + \mathbf{C}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{x}, \mathbf{p}]$$
$$\mathbf{D}(\mathbf{i}, \mathbf{j}, \mathbf{p}, \mathbf{x}, \mathbf{F}, \mathbf{g}) = \mathbf{max_k} \mathbf{C}[\mathbf{i}, \mathbf{k}, \mathbf{x}, \mathbf{p}] + \mathbf{C}[\mathbf{k}+\mathbf{1}, \mathbf{j}, \mathbf{p}, \mathbf{g}]$$

We have added another index which ranges over n, so without pruning, we have now increased the running time to $O(n^6)$. However, every step now includes both a g and a p (and often an x), so there is at least one implied edge in every step. If pruning is done in such a way that each word has at most k parents, then each word's set of grandparent and parent possibilities is at most k^2. To run all of the **S/C** steps, we therefore need $O(k^2 n^3)$ time; for all of the **M/C** steps, $O(k^2 n^4)$ time; for all of the **M/D** steps, $O(k n^4)$; for all of the **D** subproblems, $O(k^2 n^4)$. The overall running time is therefore $O(k^2 n^4)$, and we have shown that when edges are sufficiently pruned, grandparent factors add only an extra factor of k, and not a full extra factor of n.

7 Experiments

The space of projective trees is strictly contained within the space of gap-minding trees which is strictly contained within spanning trees. Which space is most appropriate for natural language parsing may depend on the particular language and the type and frequencies of non-projective structures found in it. In this section we compare the parsing accuracy across languages for a parser which uses either the Eisner algorithm (projective), MST (spanning trees), or MaxGapMindingTree (gap-minding trees) as its decoder for both training and inference.

We implemented both the basic gap-minding algorithm and the gap-minding algorithm with grandparent scoring as extensions to MSTParser[10]. MSTParser (McDonald et al., 2005b; McDonald et al., 2005a) uses the Margin Infused Relaxed Algorithm (Crammer and Singer, 2003) for discriminative training. Training requires a decoder which produces the highest scoring tree (in the space of valid trees) under the current model weights. This same decoder is then used to produce parses at test time. MSTParser comes packaged with the Eisner algorithm (for projective trees) and MST (for spanning trees). MSTParser also includes two second order models: one of which is a projective decoder that also scores siblings (Proj+Sib) and the other of which produces non-projective trees by rearranging edges after producing a projective tree (Proj+Sib+Rearr). We add a further decoder with

[10] http://sourceforge.net/projects/mstparser/

the algorithm presented here for gap minding trees, and plan to make the extension publicly available. The gap-minding decoder has both an edge-factored implementation and a version which scores grandparents as well.[11]

The gap-minding algorithm is much more efficient when edges have been pruned so that each word has at most k potential parents. We use the weights from the trained MST models combined with the Matrix Tree Theorem (Smith and Smith, 2007; Koo et al., 2007; McDonald and Satta, 2007) to produce marginal probabilities of each edge. We wanted to be able to both achieve the running time bound and yet take advantage of the fact that the size of the set of reasonable parent choices is variable. We therefore use a hybrid pruning strategy: each word's set of potential parents is the *smaller* of a) the top k parents (we chose $k = 10$) or b) the set of parents whose probabilities are above a threshold (we chose $th = .001$). The running time for the gap-minding algorithm is then $O(kn^4)$; with the grandparent features the gap-minding running time is $O(k^2n^4)$.

The training and test sets for the six languages come from the CoNLL-X shared task.[12] We train the gap-minding algorithm on sentences of length at most 100[13] (the vast majority of sentences). The projective and MST models are trained on all sentences and are run without any pruning. The Czech training set is much larger than the others and so for Czech only the first 10,000 training sentences were used. Testing is on the full test set, with no length restrictions.

The results are shown in Table 2. The first three lines show the first order gap-minding decoder compared with the first order projective and MST decoders. The gap-minding decoder does better than the projective decoder on Czech, Danish, and Dutch, the three languages with the most non-projectivity, even though it was at a competitive disadvantage in terms of both pruning and (on languages with very long sentences) training data. The gap-minding decoder with grandparent features is better than the projective decoder with sibling features on all six of the languages. On some languages, the local search decoder with siblings has the absolute highest accuracy in Table 2; on other languages (Czech and Swedish) the gap-minding+grandparents has the highest accuracy. While not directly comparable because of the difference in features, the promising performance of the gap-minding+grandparents decoder shows that the space of gap-minding trees is larger than the space of projective trees, yet unlike spanning trees, it is tractable to find the best tree with higher order features. It would be interesting to extend the gap-minding algorithm to include siblings as well.

	Ar	Cz	Da	Du	Pt	Sw
Proj.	78.0	80.0	88.2	79.8	87.4	86.9
MST	78.0	80.4	88.1	84.6	86.7	86.2
Gap-Mind	77.6	80.8	88.6	83.9	86.8	86.0
Proj+Sib	78.2	80.0	88.9	81.1	87.5	88.1
+Rearr	78.5	81.3	89.3	85.4	88.2	87.7
GM+Grand	78.3	82.1	89.1	84.6	87.7	88.5

Table 2: Unlabeled Attachment Scores on the CoNLL-X shared task test set.

8 Conclusion

Gap inheritance, a structural property on trees, has implications both for natural language syntax and for natural language parsing. We have shown that the mildly non-projective trees present in natural language treebanks *all* have zero or one children inherit each parent's gap. We also showed that the assumption of 1 gap inheritance removes a factor of n from parsing time, and the further assumption of 0 gap inheritance removes yet another factor of n. The space of gap-minding trees provides a closer fit to naturally occurring linguistic structures than the space of projective trees, and unlike spanning trees, the inclusion of higher order factors does not substantially increase the difficulty of finding the maximum scoring tree in that space.

[11] The grandparent features used were identical to the features provided within MSTParser for the second-order sibling parsers, with one exception — many features are conjoined with a direction indicator, which in the projective case has only two possibilities. We replaced this two-way distinction with a six-way distinction of the six possible orders of the grandparent, parent, and child.

[12] MSTParser produces labeled dependencies on CoNLL formatted input. We replace all labels in the training set with a single dummy label to produce unlabeled dependency trees.

[13] Because of long training times, the gap-minding with grandparent models for Portuguese and Swedish were trained on only sentences up to 50 words.

Acknowledgments

We would like to thank Aravind Joshi for comments on an earlier draft. This material is based upon work supported under a National Science Foundation Graduate Research Fellowship.

Algorithm 1: MaxGapMindingTree

Init: $\forall_{i\in[1,n]} C[i,i,i] = 0$
for $size = 0$ *to* $n-1$ **do**
  for $i = 1$ *to* $n - size$ **do**
    $j = i + size$
    /* Endpoint parents */
    if $size > 0$ **then**
      $C[i,j,i] = C[i+1,j,i]$
      $C[i,j,j] = C[i,j-1,j]$
    /* Interior parents */
    for $p = i+1$ *to* $j-1$ **do**
      $C[i,j,p] = \max$ (SingleChild(i,j,p),
      EndpointsDiff(i,j,p),
      EndsFromLeftChild(i,j,p),
      EndsFromRightChild(i,j,p))
    /* Exterior parents */
    forall the $p \in [0, i-1] \cup [j+1, n]$ **do**
      $C[i,j,p] = \max$ (SingleChild(i,j,p),
      EndpointsDiff(i,j,p),
      EndsFromLeftChild(i,j,p),
      EndsFromRightChild(i,j,p))
    /* Helper subproblems */
    for $p \in [0, n]$ **do**
      forall the $x \in PosChild[p] \wedge x \notin [i,j]$ **do**
        if $p \neq j$ **then**
          $D[i,j,p,x,T] = Max2Subtrees(i,j,p,x,T)$
        if $p \neq i$ **then**
          $D[i,j,p,x,F] = Max2Subtrees(i,j,p,x,F)$
Final answer: $C[1,n,0]$

Function SingleChild(i,j,p)

$X = PosChild[p] \cap [i,j]$
/* Interior p */
if $p > i \wedge p < j$ **then**
  return $\max_{x\in X} C[i,p-1,x]$
    $+C[p+1,j,x] + Score(p,x)$
/* Exterior p */
else
  return $\max_{x\in X} C[i,j,x] + Score(p,x)$

Function EndpointsDiff(i,j,p)

return $\max_{k\in[i,j-1]} C[i,k,p] + C[k+1,j,p]$

Function EndsFromLeftChild(i,j,p)

/* Interior p */
if $p > i \wedge p < j$ **then**
  $X = PosChild[p] \cap [i,p-1]$
  forall the $x \in X \wedge x < p$ **do**
    $K[x] = [x, p-1]$
/* Exterior p */
else
  $X = PosChild[p] \cap [i,j]$
  forall the $x \in X$ **do**
    $K = [x, j-2]$
return $\max_{x\in X, k\in K[x]} C[i,k,x]$
    $+Score(p,x) + D[k+1,j,p,x,T]$

Function EndsFromRightChild(i,j,p)

/* Interior p */
if $p > i \wedge p < j$ **then**
  $X = PosChild[p] \cap [p+1,j]$
  forall the $x \in X \wedge x > p$ **do**
    $K[x] = [p+1, x]$
/* Exterior p */
else
  $X = PosChild[p] \cap [i,j]$
  forall the $x \in X$ **do**
    $K[x] = [i+2, x]$
return $\max_{x\in X, k\in K[x]} C[k,j,x]$
    $+Score(p,x) + D[i,k-1,p,x,F]$

Function Max2Subtrees(i,j,p,x,pOnLeft)

/* Interior p */
if $p \geq i \wedge p \leq j$ **then**
  if *pOnLeft* **then**
    $K = [p, j-1]$
    return $\max_{k\in K} C[i,k,p] + C[k+1,j,x]$
  else
    $K = [i, p-1]$
    return $\max_{k\in K} C[i,k,x] + C[k+1,j,p]$
/* Exterior p */
else
  $K = [i, j-1]\}$
  if *pOnLeft* **then**
    return $\max_{k\in K} C[i,k,p] + C[k+1,j,x]$
  else
    return $\max_{k\in K} C[i,k,x] + C[k+1,j,p]$

References

M. Bodirsky, M. Kuhlmann, and M. Möhl. 2005. Well-nested drawings as models of syntactic structure. In *In Tenth Conference on Formal Grammar and Ninth Meeting on Mathematics of Language*, pages 88–1. University Press.

X. Carreras, M. Collins, and T. Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of CoNLL*, pages 9–16. Association for Computational Linguistics.

X. Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, volume 7, pages 957–961.

J. Chen-Main and A. Joshi. 2010. Unavoidable ill-nestedness in natural language and the adequacy of tree local-mctag induced dependency structures. In *Proceedings of the Tenth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+ 10)*.

J. Chen-Main and A.K. Joshi. 2012. A dependency perspective on the adequacy of tree local multi-component tree adjoining grammar. In *Journal of Logic and Computation*. (to appear).

N. Chomsky. 1981. *Lectures on Government and Binding*. Dordrecht: Foris.

K. Crammer and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, March.

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

J. Eisner and N.A. Smith. 2010. Favor short dependencies: Parsing with soft and hard constraints on dependency length. In Harry Bunt, Paola Merlo, and Joakim Nivre, editors, *Trends in Parsing Technology: Dependency Parsing, Domain Adaptation, and Deep Parsing*, chapter 8, pages 121–150. Springer.

J. Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer Academic Publishers, October.

R. Fiengo. 1977. On trace theory. *Linguistic Inquiry*, 8(1):35–61.

C. Gómez-Rodríguez, J. Carroll, and D. Weir. 2011. Dependency parsing schemata and mildly non-projective dependency parsing. *Computational Linguistics*, 37(3):541–586.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of ACL*, pages 1–11.

T. Koo, A. Globerson, X. Carreras, and M. Collins. 2007. Structured prediction models via the matrix-tree theorem. In *Proceedings of EMNLP-CoNLL*.

T. Koo, A.M. Rush, M. Collins, T. Jaakkola, and D. Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of EMNLP*, pages 1288–1298.

M. Kuhlmann and J. Nivre. 2006. Mildly non-projective dependency structures. In *Proceedings of COLING/ACL*, pages 507–514.

A.F.T. Martins, N.A. Smith, and E.P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of ACL*, pages 342–350.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, pages 81–88.

R. McDonald and G. Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, pages 121–132.

R. McDonald, K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT-EMNLP*, pages 523–530.

T. Reinhart. 1976. *The Syntactic Domain of Anaphora*. Ph.D. thesis, Massachusetts Institute of Technology.

G. Satta and W. Schuler. 1998. Restrictions on tree adjoining languages. In *Proceedings of COLING-ACL*, pages 1176–1182.

D.A. Smith and N.A. Smith. 2007. Probabilistic models of nonprojective dependency trees. In *Proceedings of EMNLP-CoNLL*.