

# Minimal-length linearizations for mildly context-sensitive dependency trees

**Y. Albert Park**

Department of Computer Science and Engineering  
9500 Gilman Drive  
La Jolla, CA 92037-404, USA  
yapark@ucsd.edu

**Roger Levy**

Department of Linguistics  
9500 Gilman Drive  
La Jolla, CA 92037-108, USA  
rlevy@ling.ucsd.edu

## Abstract

The extent to which the organization of natural language grammars reflects a drive to minimize dependency length remains little explored. We present the first algorithm polynomial-time in sentence length for obtaining the minimal-length linearization of a dependency tree subject to constraints of mild context sensitivity. For the minimally context-sensitive case of gap-degree 1 dependency trees, we prove several properties of minimal-length linearizations which allow us to improve the efficiency of our algorithm to the point that it can be used on most naturally-occurring sentences. We use the algorithm to compare optimal, observed, and random sentence dependency length for both surface and deep dependencies in English and German. We find in both languages that analyses of surface and deep dependencies yield highly similar results, and that mild context-sensitivity affords very little reduction in minimal dependency length over fully projective linearizations; but that observed linearizations in German are much closer to random and farther from minimal-length linearizations than in English.

## 1 Introduction

This paper takes up the relationship between two hallmarks of natural language dependency structure. First, there seem to be qualitative constraints on the relationship between the dependency structure of the words in a sentence and their linear ordering. In particular, this relationship seems to be such that any

natural language sentence, together with its dependency structure, should be generable by a *mildly context-sensitivity* formalism (Joshi, 1985), in particular a linear context-free rewrite system in which the right-hand side of each rule has a distinguished head (Pollard, 1984; Vijay-Shanker et al., 1987; Kuhlmann, 2007). This condition places strong constraints on the linear contiguity of word-word dependency relations, such that only limited classes of crossing context-free dependency structures may be admitted.

The second constraint is a softer preference for words in a dependency relation to occur in close proximity to one another. This constraint is perhaps best documented in psycholinguistic work suggesting that large distances between governors and dependents induce processing difficulty in both comprehension and production (Hawkins, 1994, 2004; Gibson, 1998; Jaeger, 2006). Intuitively there is a relationship between these two constraints: consistently large dependency distances in a sentence would require many crossing dependencies. However, it is not the case that crossing dependencies always mean longer dependency distances. For example, (1) below has no crossing dependencies, but the distance between *arrived* and its dependent *Yesterday* is large. The overall dependency length of the sentence can be reduced by extraposing the relative clause *who was wearing a hat*, resulting in (2), in which the dependency *Yesterday*→*arrived* crosses the dependency *woman*←*who*.

- (1) Yesterday a woman who was wearing a hat arrived.
- (2) Yesterday a woman arrived who was wearing a hat.

There has been some recent work on dependency length minimization in natural language sentences (Gildea and Temperley, 2007), but the relationship between the precise constraints on available linearizations and dependency length minimization remains little explored. In this paper, we introduce the first efficient algorithm for obtaining linearizations of dependency trees that minimize overall dependency lengths subject to the constraint of mild context-sensitivity, and use it to investigate the relationship between this constraint and the distribution of dependency length actually observed in natural languages.

## 2 Projective and mildly non-projective dependency-tree linearizations

In the last few years there has been a resurgence of interest in computation on dependency-tree structures for natural language sentences, spurred by work such as McDonald et al. (2005a,b) showing that working with dependency-tree syntactic representations in which each word in the sentence corresponds to a node in the dependency tree (and vice versa) can lead to algorithmic benefits over constituency-structure representations. The *linearization* of a dependency tree is simply the linear order in which the nodes of the tree occur in a surface string. There is a broad division between two classes of linearizations: *projective* linearizations that do not lead to any crossing dependencies in the tree, and *non-projective* linearizations that involve at least one crossing dependency pair. Example (1), for example, is projective, whereas Example (2) is non-projective due to the crossing between the *Yesterday*→*arrived* and *woman*←*who* dependencies.

Beyond this dichotomy, however, the homomorphism from headed tree structures to dependency structures (Miller, 2000) can be used together with work on the mildly context-sensitive formalism linear context-free rewrite systems (LCFRSs) (Vijay-Shanker et al., 1987) to characterize various classes of *mildly non-projective* dependency-tree linearizations (Kuhlmann and Nivre, 2006). The LCFRSs are an infinite sequence of classes of formalism for generating surface strings through derivation trees in a rule-based context-free rewriting system. The  $i$ -th LCFRS class (for  $i = 0, 1, 2, \dots$ ) imposes the con-

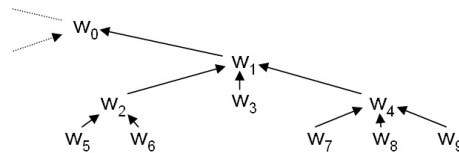


Figure 1: Sample dependency subtree for Figure 2

straint that every node in the derivation tree maps to a collection of at most  $i + 1$  contiguous substrings. The 0-th class of LCFRS, for example, corresponds to the context-free grammars, since each node in the derivation tree must map to a single contiguous substring; the 1st class of LCFRS corresponds to Tree-Adjoining Grammars (Joshi et al., 1975), in which each node in the derivation tree must map to at most a pair of contiguous substrings; and so forth. The dependency trees induced when each rewrite rule in an  $i$ -th order LCFRS distinguish a unique *head* can similarly be characterized by being of *gap-degree*  $i$ , so that  $i$  is the maximum number of gaps that may appear between contiguous substrings of any subtree in the dependency tree (Kuhlmann and Möhl, 2007). The dependency tree for Example (2), for example, is of gap-degree 1. Although there are numerous documented cases in which projectivity is violated in natural language, there are exceedingly few documented cases in which the documented gap degree exceeds 1 (though see, for example, Kobele, 2006).

## 3 Finding minimal dependency-length linearizations

Even under the strongest constraint of projectivity, the number of possible linearizations of a dependency tree is exponential in both sentence length and arity (the maximum number of dependencies for any word). As pointed out by Gildea and Temperley (2007), however, finding the unconstrained minimal-length linearization is a well-studied problem with an  $O(n^{1.6})$  solution (Chung, 1984). However, this approach does not take into account constraints of projectivity or mild context-sensitivity.

Gildea and Temperley themselves introduced a novel efficient algorithm for finding the minimized dependency length of a sentence subject to the constraint that the linearization is projective. Their algorithm can perhaps be most simply understood by making three observations. First, the total depen-

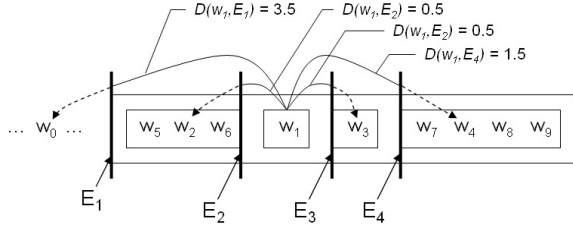


Figure 2: Dependency length factorization for efficient projective linearization, using the dependency subtree of Figure 1

dependency length of a projective linearization can be written as

$$\sum_{w_i} \left[ D(w_i, E_i) + \sum_{w_j \xrightarrow{\text{dep}} w_i} D(w_i, E_j) \right] \quad (1)$$

where  $E_i$  is the boundary of the contiguous substring corresponding to the dependency subtree rooted at  $w_i$  which stands between  $w_i$  and its governor, and  $D(w_i, E_j)$  is the distance from  $w_i$  to  $E_j$ , with the special case of  $D(w_{\text{root}}, E_{\text{root}}) = 0$  (Figures 1 and 2). Writing the total dependency length this way makes it clear that each term in the outer sum can be optimized independently, and thus one can use dynamic programming to recursively find optimal subtree orderings from the bottom up. Second, for each subtree, the optimal ordering can be obtained by placing dependent subtrees on alternating sides of  $w$  from inside out in order of increasing length. Third, the total dependency lengths between any words within an ordering stays the same when the ordering is reversed, letting us assume that  $D(w_i, E_i)$  will be the length to the closest edge. These three observations lead to an algorithm with worst-case complexity of  $O(n \log m)$  time, where  $n$  is sentence length and  $m$  is sentence arity. (The  $\log m$  term arises from the need to sort the daughters of each node into descending order of length.)

When limited subclasses of nonprojectivity are admitted, however, the problem becomes more difficult because total dependency length can no longer be written in such a simple form as in Equation (1). Intuitively, the size of the effect on dependency length of a decision to order a given subtree discontinuously, as in *a woman... who was wearing a hat* in Example (2), cannot be calculated without consulting the length of the string that the discontinuous

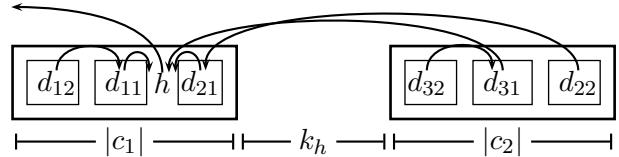


Figure 3: Factorizing dependency length at node  $w_i$  of a mildly context-sensitive dependency tree. This partial linearization of head with dependent components makes  $c_1$  the head component and leads to  $l = 2$  links crossing between  $c_1$  and  $c_2$ .

subtree would be wrapped around. Nevertheless, for any limited gap degree, it is possible to use a different factorization of dependency length that keeps computation polynomial in sentence length. We introduce this factorization in the next section.

#### 4 Minimization with limited gap degree

We begin by defining some terms. We use the word *component* to refer to a full linearization of a subtree in the case where it is realized as a single contiguous string, or to refer to any of the contiguous substrings produced when a subtree is realized discontinuously. We illustrate the factorization for gap-degree 1, so that any subtree has at most two components. We refer to the component containing the head of the subtree as the *head component*, the remaining component as the *dependent component*, and for any given (head component, dependent component) pair, we use *pair component* to refer to the other component in the pair. We refer to the two components of dependent  $d_j$  as  $d_{j1}$  and  $d_{j2}$  respectively, and assume that  $d_{j1}$  is the head component. When dependencies can cross, total dependency length cannot be factorized as simply as in Equation (1) for the projective case. However, we can still make use of a more complex factorization of the total dependency length as follows:

$$\sum_{w_i} \left[ D(w_i, E_i) + \sum_{w_j \xrightarrow{\text{dep}} w_i} [D(w_i, E_j) + l_j k_j] \right] \quad (2)$$

where  $l_j$  is the number of links crossing between the two components of  $d_j$ , and  $k_j$  is the distance added between these two components by the partial linearization at  $w_i$ . Figure 3 illustrates an example of

such a partial linearization, where  $k_2$  is  $|d_{31}| + |d_{32}|$  due to the fact that the links between  $d_{21}$  and  $d_{22}$  have to cross both components of  $d_3$ . The factorization in Equation (2) allows us to use dynamic programming to find minimal-length linearizations, so that worst-case complexity is polynomial rather than exponential in sentence length. However, the additional term in the factorization means that we need to track the number of links  $l$  crossing between the two components of the subtree  $S_i$  headed by  $w_i$  and the component lengths  $|c_1|$  and  $|c_2|$ . Additionally, the presence of crossing dependencies means that Gildea and Temperley’s proof that ordering dependent components from the inside out in order of increasing length no longer goes through. This means that at each node  $w_i$  we need to hold on to the minimal-length partial linearization for each combination of the following quantities:

- $|c_2|$  (which also determines  $|c_1|$ );
- the number of links  $l$  between  $c_1$  and  $c_2$ ;
- and the direction of the link between  $w_i$  and its governor.

We shall refer to a combination of these factors as a *status set*. The remainder of this section describes a dynamic-programming algorithm for finding optimal linearizations based on the factorization in Equation (2), and continues with several further findings leading to optimizations that make the algorithm tractable for naturally occurring sentences.

#### 4.1 Algorithm 1

Our first algorithm takes a tree and recursively finds the optimal orderings for each possible status set of each of its child subtrees, which it then uses to calculate the optimal ordering of the tree. To calculate the optimal orderings for each possible status set of a subtree  $S$ , we use the brute-force method of choosing all combinations of one status set from each child subtree, and for each combination, we try all possible orderings of the components of the child subtrees, calculate all possible status sets for  $S$ , and store the minimal dependency value for each appearing status set of  $S$ . The number of possible length pairings  $|c_1|, |c_2|$  and number of crossing links  $l$  are each bounded above by the sentence length  $n$ ,

so that the maximum number of status sets at each node is bounded above by  $n^2$ . Since the sum of the status sets of all child subtrees is also bounded by  $n^2$ , the maximum number of status set combinations is bounded by  $(\frac{n^2}{m})^m$  (obtainable from the inequality of arithmetic and geometric means). There are  $(2m+1)!m$  possible arrangements of head word and dependent components into two components. Since there are  $n$  nodes in the tree and each possible combination of status sets from each dependent sub tree must be tried, this algorithm has worst-case complexity of  $O((2m+1)!mn(\frac{n^2}{m})^m)$ . This algorithm could be generalized for mildly context-sensitive linearizations polynomial in sentence length for any gap degree desired, by introducing additional  $l$  terms denoting the number of links between pairs of components. However, even for gap degree 1 this bound is incredibly large, and as we show in Figure 7, algorithm 1 is not computationally feasible for batch processing sentences of arity greater than 5.

#### 4.2 Algorithm 2

We now show how to speed up our algorithm by proving by contradiction that for any optimal ordering which minimizes the total dependency length with the two-cluster constraint, for any given subtree  $S$  and its child subtree  $C$ , the pair components  $c_1$  and  $c_2$  of a child subtree  $C$  must be placed on opposite sides of the head  $h$  of subtree  $S$ .

Let us assume that for some dependency tree structure, there exists an optimal ordering where  $c_1$  and  $c_2$  are on the same side of  $h$ . Let us refer to the ordered set of words between  $c_1$  and  $c_2$  as  $v$ . None of the words in  $v$  will have dependency links to any of the words in  $c_1$  and  $c_2$ , since the dependencies of the words in  $c_1$  and  $c_2$  are either between themselves or the one link to  $h$ , which is not between the two components by our assumption. There will be  $j_1 \geq 0$  links from  $v$  going over  $c_1$ ,  $j_2 \geq 0$  dependency links from  $v$  going over  $c_2$ , and  $l \geq 1$  links between  $c_1$  and  $c_2$ . Without loss of generality, let us assume that  $h$  is on the right side of  $c_2$ . Let us consider the effect on total dependency length of swapping  $c_1$  with  $v$ , so that the linear ordering is  $v c_1 c_2 \prec h$ . The total dependency length of the new word ordering changes by  $-j_1|c_1| - l|v| + j_2|c_1|$  if  $c_2$  is the head component, and decreases by another  $|v|$  if  $c_1$  is the head component. Thus the total change in dependency length

is less than or equal to

$$(j_2 - j_1)|c_1| - l \times |v| < (j_2 - j_1)|c_1| \quad (3)$$

If instead we swap places of  $v$  with  $c_2$  instead of  $c_1$  so that we have  $c_1 c_2 v \prec h$ , we find that the total change in dependency length is less than or equal to

$$(j_1 - j_2)|c_2| - (l - 1)|v| \leq (j_1 - j_2)|c_2| \quad (4)$$

It is impossible for the right-hand sides of (3) and (4) to be positive at the same time, so swapping  $v$  with either  $c_1$  or  $c_2$  must lead to a linearization with lower overall dependency length. But this is a contradiction to our original assumption, so we see that for any optimal ordering, all split child subtree components  $c_1$  and  $c_2$  of the child subtree of  $S$  must be placed on opposite sides of the head  $h$ .

This constraint allows us to simplify our algorithm for finding the minimal-length linearization. Instead of going through all logically possible orderings of components of the child subtrees, we can now decide on which side the head component will be on, and go through all possible orderings for each side. This changes the factorial part of our algorithm run time from  $(2m + 1)!m$  to  $2^m(m!)^2m$ , giving us  $O(2^m(m!)^2mn(\frac{n^2}{m})^m)$ , greatly reducing actual processing time.

### 4.3 Algorithm 3

We now present two more findings for further increasing the efficiency of the algorithm. First, we look at the status sets which need to be stored for the dynamic programming algorithm. In the straightforward approach we first presented, we stored the optimal dependency lengths for all cases of possible status sets. We now know that we only need to consider cases where the pair components are on opposite sides. This means the direction of the link from the head to the parent will always be toward the inside direction of the pair components, so we can re-define the status set as  $(p, l)$  where  $p$  is again the length of the dependent component, and  $l$  is the number of links between the two pair components. If the  $p$  values for sets  $s_1$  and  $s_2$  are equal,  $s_1$  has a smaller number of links than  $s_2$  ( $l_{s_1} \leq l_{s_2}$ ) and  $s_1$  has a smaller or equal total dependency length to  $s_2$ , then replacing the components of  $s_2$  with  $s_1$  will always give us the same or more optimal total

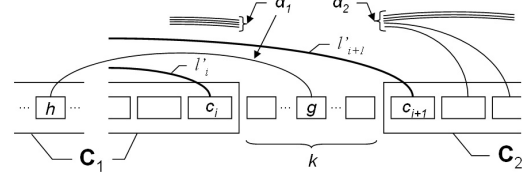


Figure 4: Initial setup for latter part of optimization proof in section 4.4. To the far left is the head  $h$  of subtree  $S$ . The component pair  $C_1$  and  $C_2$  makes up  $S$ , and  $g$  is the governor of  $h$ . The length of the substring  $v$  between  $C_1$  and  $C_2$  is  $k$ .  $c_i$  and  $c_{i+1}$  are child subtree components.

dependency length. Thus, we do not have to store instances of these cases for our algorithm.

Next, we prove by contradiction that for any two status sets  $s_1$  and  $s_2$ , if  $p_{s_1} > p_{s_2} > 0$ ,  $l_{s_1} = l_{s_2}$ , and the TOTAL INTERNAL DEPENDENCY LENGTH  $t_1$  of  $s_1$ —defined as the sum in Equation (2) over only those words inside the subtree headed by  $h$ —is less than or equal to  $t_2$  of  $s_2$ , then using  $s_1$  will be at least as good as  $s_2$ , so we can ignore  $s_2$ . Let us suppose that the optimal linearization can use  $s_2$  but not  $s_1$ . Then in the optimal linearization, the two pair components  $c_{s_2,1}$  and  $c_{s_2,2}$  of  $s_2$  are on opposite sides of the parent head  $h$ . WLOG, let us assume that components  $c_{s_1,1}$  and  $c_{s_2,1}$  are the dependent components. Let us denote the total number of links going over  $c_{s_2,1}$  as  $j_1$  and the words between  $c_{s_2,1}$  and  $c_{s_2,2}$  as  $v$  (note that  $v$  must contain  $h$ ). If we swap  $c_{s_2,1}$  with  $v$ , so that  $c_{s_2,1}$  lies adjacent to  $c_{s_2,2}$ , then there would be  $j_2 + 1$  links going over  $c_{s_2,1}$ . By moving  $c_{s_2,1}$  from opposite sides of the head to be right next to  $c_{s_2,2}$ , the total dependency length of the sentence changes by  $-j_1|c_{s_2,1}| - l_{s_2}|v| + (j_2 + 1)|c_{s_2,1}|$ . Since the ordering was optimal, we know that

$$(j_2 - j_1 + 1)|c_{s_2,1}| - l_{s_2}|v| \geq 0$$

Since  $l > 0$ , we can see that  $j_1 - j_2 \leq 0$ . Now, instead of swapping  $v$  with  $c_{s_2,1}$ , let us try substituting the components from  $s_1$  instead of  $s_2$ . The change of the total dependency length of the sentence will be:

$$\begin{aligned} & j_1 \times (|c_{s_1,1}| - |c_{s_2,1}|) + j_2 \times (|c_{s_1,2}| \\ & \quad - |c_{s_2,2}|) + t_1 - t_2 \\ & = (j_1 - j_2) \times (p_{s_1} - p_{s_2}) + (t_1 - t_2) \end{aligned}$$

Since  $j_1 - j_2 \leq 0$  and  $p_{s_1} > p_{s_2}$ , the first term is less than or equal to 0 and since  $t_1 - t_2 \leq 0$ , the total dependency length will have been equal or

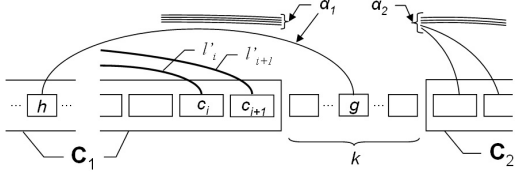


Figure 5: Moving  $c_{i+1}$  to  $C_1$

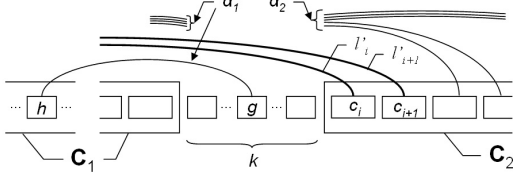


Figure 6: Moving  $c_i$  to  $C_2$

have decreased. But this contradicts our assumption that only  $s_2$  can be part of an optimal ordering.

This finding greatly reduces the number of status sets we need to store and check higher up in the algorithm. The worst-case complexity remains  $O(2^m m!^2 mn (\frac{n^2}{m})^m)$ , but the actual runtime is reduced by several orders of magnitude.

#### 4.4 Algorithm 4

Our last optimization is on the ordering among the child subtree components on each side of the subtree head  $h$ . The initially proposed algorithm went through all combinations of possible orderings to find the optimal dependency length for each status set. By the first optimization in section 4.2 we have shown that we only need to consider the orderings in which the components are on opposite sides of the head. We now look into the ordering of the components on each side of the head. We first define the *rank value*  $r$  for each component  $c$  as follows:

$$\frac{|c|}{\# \text{ links between } c \text{ and its pair component} + I(c)}$$

where  $I(c)$  is the indicator function having value 1 if  $c$  is a head component and 0 otherwise. Using this definition, we prove by contradiction that the ordering of the components from the head outward must be in order of increasing rank value.

Let us suppose that at some subtree  $S$  headed by  $h$  and with head component  $C_1$  and dependent component  $C_2$ , there is an optimal linearization in which there exist two components  $c_i$  and  $c_{i+1}$  of immediate subtrees of  $S$  such that  $c_i$  is closer to  $h$ , the com-

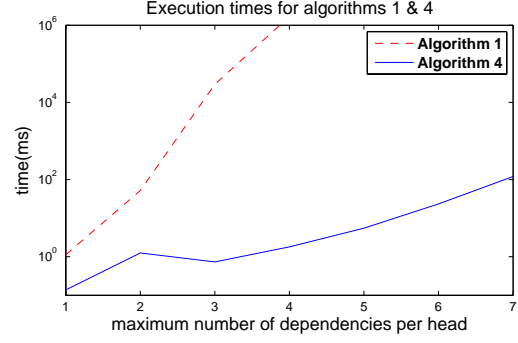


Figure 7: Timing comparison of first and fully optimized algorithms

ponents have rank values  $r_i$  and  $r_{i+1}$  respectively,  $r_i > r_{i+1}$ , and no other component of the immediate subtrees of  $S$  intervenes between  $c_i$  and  $c_{i+1}$ . We shall denote the number of links between each component and its pair component as  $l_i, l_{i+1}$ . Let  $l'_i = l_i + I(c_i)$  and  $l'_{i+1} = l_{i+1} + I(c_{i+1})$ . There are two cases to consider: either (1)  $c_i$  and  $c_{i+1}$  are within the same component of  $S$ , or (2)  $c_i$  is at the edge of  $C_1$  nearest  $C_2$  and  $c_{i+1}$  is at the edge of  $C_2$  nearest  $C_1$ .

Consider case 1, and let us swap  $c_i$  with  $c_{i+1}$ ; this affects only the lengths of links involving connections to  $c_i$  or  $c_{i+1}$ . The total dependency length of the new linearization will change by

$$-l'_{i+1}|c_i| + l'_i|c_{i+1}| = -l'_i l'_{i+1}(r_i - r_{i+1}) < 0$$

This is a contradiction to the assumption that we had an optimal ordering.

Now consider case 2, which is illustrated in Figure 4. We denote the number of links going over  $c_i$  and  $c_{i+1}$ , excluding links to  $c_i, c_{i+1}$  as  $\alpha_1$  and  $\alpha_2$  respectively, and the length of words between the edges of  $C_1$  and  $C_2$  as  $k$ . Let us move  $c_{i+1}$  to the outermost position of  $C_1$ , as shown in Figure 5. Since the original linearization was optimal, we have:

$$\begin{aligned} -\alpha_2|c_{i+1}| + \alpha_1|c_{i+1}| - l'_{i+1}k &\geq 0 \\ (\alpha_1 - \alpha_2)|c_{i+1}| &\geq l'_{i+1}k \\ (\alpha_1 - \alpha_2)r_{i+1} &\geq k \end{aligned}$$

Let us also consider the opposite case of moving  $c_i$  to the inner edge of  $C_2$ , as shown in Figure 6. Once again due to optimality of the original linearization, we have

DLA	English		German	
	Surface	Deep	Surface	Deep
Optimal with one crossing dependency	32.7	33.0	24.5	23.3
Optimal with projectivity constraint	34.1	34.4	25.5	24.2
Observed	46.6	48.0	43.6	43.1
Random with projectivity constraint	82.4	82.8	50.6	49.2
Random with two-cluster constraint	84.0	84.3	50.7	49.5
Random ordering with no constraint	183.2	184.2	106.9	101.1

Table 1: Average sentence dependency lengths(with max arity of 10)

$$\begin{aligned}
-\alpha_1|c_i| + \alpha_2|c_i| + l'_i k &\geq 0 \\
(\alpha_2 - \alpha_1)|c_i| &\geq -l'_i k \\
(\alpha_1 - \alpha_2)r_i &\leq k
\end{aligned}$$

But this is a contradiction, since  $r_i > r_{i+1}$ . Combining the two cases, we can see that regardless of where the components may be split, in an optimal ordering the components going outwards from the head must have an increasing rank value.

This result allows us to simplify our algorithm greatly, because we no longer need to go through all combinations of orderings. Once it has been decided which components will come on each side of the head, we can sort the components by rank value and place them from the head out. This reduces the factorial component of the algorithm’s complexity to  $m \log m$ , and the overall worst-case complexity to  $O(nm^2 \log m (\frac{2n^2}{m})^m)$ . Although this is still exponential in the arity of the tree, nearly all sentences encountered in treebanks have an arity low enough to make the algorithm tractable and even very efficient, as we show in the following section.

## 5 Empirical results

Using the above algorithm, we calculated minimal dependency lengths for English sentences from the WSJ portion of the Penn Treebank, and for German sentences from the NEGRA corpus. The English-German comparison is of interest because word order is freer, and crossing dependencies more common, in German than in English (Kruijff and Vasisht, 2003). We extracted dependency trees from these corpora using the head rules of Collins (1999) for English, and the head rules of Levy and Manning (2004) for German. Two dependency trees were extracted from each sentence, the *surface tree* extracted by using the head rules on the context-

free tree representation (i.e. no crossing dependencies), and the *deep tree* extracted by first returning discontinuous dependents (marked by \*T\* and \*ICH\* in WSJ, and by \*T\* in the Penn-format version of NEGRA) before applying head rules. Figure 7 shows the average time it takes to calculate the minimal dependency length with crossing dependencies for WSJ sentences using the unoptimized algorithm of Section 4.1 and the fully optimized algorithm of Section 4.4. Timing tests were implemented and performed using Java 1.6.0.10 on a system running Linux 2.6.18-6-amd64 with a 2.0 GHz Intel Xeon processor and 16 gigs of memory, run on a single core. We can see from Figure 7 that the straight-forward dynamic programming algorithm takes many more magnitudes of time than our optimized algorithm, making it infeasible to calculate the minimal dependency length for larger sentences. The results we present below were obtained with the fully optimized algorithm from the sentences with a maximum arity of 10, using 49,176 of the 49,208 WSJ sentences and 20,563 of the 20,602 NEGRA sentences.

Summary results over all sentences from each corpus are shown in Table 1. We can see that for both corpora, the observed dependency length is smaller than the dependency length of random orderings, even when the random ordering is subject to the projectivity constraint. Relaxing the projectivity constraint by allowing crossing dependencies introduces a slightly lower optimal dependency length. The average sentence dependency lengths for the three random orderings are significantly higher than the observed values. It is interesting to note that the random orderings given the projectivity constraint and the two-cluster constraint have very similar dependency lengths, where as a total random ordering

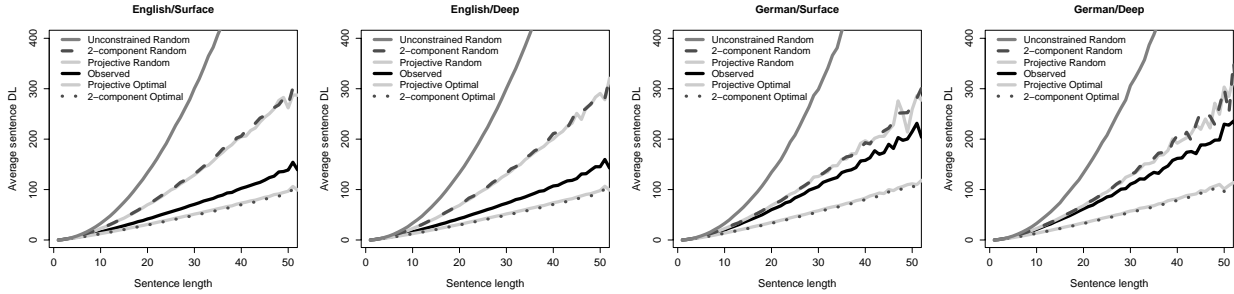


Figure 8: Average sentence DL as a function of sentence length. Legend is ordered top curve to bottom curve.

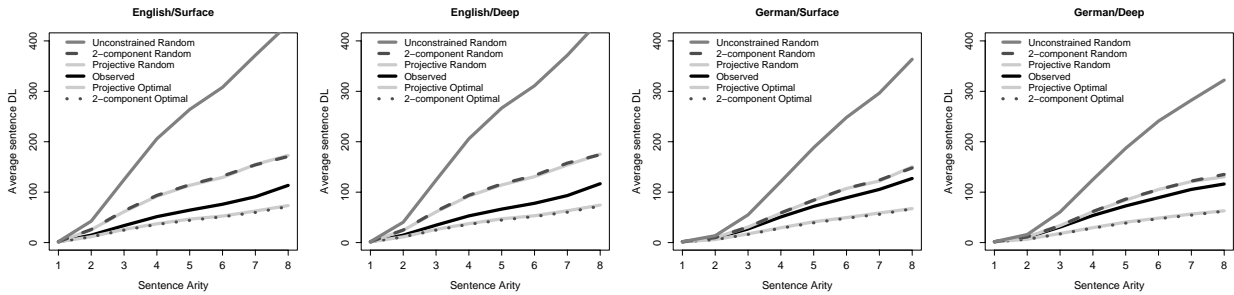


Figure 9: Average sentence DL as a function of sentence arity. Legend is ordered top curve to bottom curve.

increases the dependency length significantly.

NEGRA generally has shorter sentences than WSJ, so we need a more detailed picture of dependency length as a function of sentence length; this is shown in Figure 8. As in Table 1, we see that English, which has less crossing dependency structures than German, has observed DL closer to optimal DL and farther from random DL. We also see that the random and observed DLs behave very similarly across different sentence lengths in English and German, but observed DL grows faster in German. Perhaps surprisingly, optimal projective DL and gap-degree 1 DL tend to be very similar even for longer sentences. The picture as a function of sentence arity is largely the same (Figure 9).

## 6 Conclusion

In this paper, we have presented an efficient dynamic programming algorithm which finds minimum-length dependency-tree linearizations subject to constraints of mild context-sensitivity. For the gap-degree 1 case, we have proven several properties of these linearizations, and have used these properties to optimize our algorithm. This made it possible to find minimal dependency lengths for sentences from

the English Penn Treebank WSJ and German NEGRA corpora. The results show that for both languages, using surface dependencies and deep dependencies lead to generally similar conclusions, but that minimal lengths for deep dependencies are consistently slightly higher for English and slightly lower for German. This may be because German has many more crossing dependencies than English. Another finding is that the difference between average sentence DL does not change much between optimizing for the projectivity constraint and the two-cluster constraint: projectivity seems to give natural language almost all the flexibility it needs to minimize DL. For both languages, the observed linearization is much closer in DL to optimal linearizations than to random linearizations; but crucially, we see that English is closer to the optimal linearization and farther from random linearization than German. This finding is resonant with the fact that German has richer morphology and overall greater variability in observed word order, and with psycholinguistic results suggesting that dependencies of greater linear distance do not always pose the same increased processing load in German sentence comprehension as they do in English (Konieczny, 2000).

## References

- Chung, F. R. K. (1984). On optimal linear arrangements of trees. *Computers and Mathematics with Applications*, 10:43–60.
- Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania.
- Gibson, E. (1998). Linguistic complexity: Locality of syntactic dependencies. *Cognition*, 68:1–76.
- Gildea, D. and Temperley, D. (2007). Optimizing grammars for minimum dependency length. In *Proceedings of ACL*.
- Hawkins, J. A. (1994). *A Performance Theory of Order and Constituency*. Cambridge.
- Hawkins, J. A. (2004). *Efficiency and Complexity in Grammars*. Oxford University Press.
- Jaeger, T. F. (2006). *Redundancy and Syntactic Reduction in Spontaneous Speech*. PhD thesis, Stanford University, Stanford, CA.
- Joshi, A. K. (1985). How much context-sensitivity is necessary for characterizing structural descriptions – Tree Adjoining Grammars. In Dowty, D., Karttunen, L., and Zwicky, A., editors, *Natural Language Processing – Theoretical, Computational, and Psychological Perspectives*. Cambridge.
- Joshi, A. K., Levy, L. S., and Takahashi, M. (1975). Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1).
- Kobele, G. M. (2006). *Generating Copies: An investigation into Structural Identity in Language and Grammar*. PhD thesis, UCLA.
- Konieczny, L. (2000). Locality and parsing complexity. *Journal of Psycholinguistic Research*, 29(6):627–645.
- Kruijff, G.-J. M. and Vasishth, S. (2003). Quantifying word order freedom in natural language: Implications for sentence processing. Proceedings of the Architectures and Mechanisms for Language Processing conference.
- Kuhlmann, M. (2007). *Dependency Structures and Lexicalized Grammars*. PhD thesis, Saarland University.
- Kuhlmann, M. and Möhl, M. (2007). Mildly context-sensitive dependency languages. In *Proceedings of ACL*.
- Kuhlmann, M. and Nivre, J. (2006). Mildly non-projective dependency structures. In *Proceedings of COLING/ACL*.
- Levy, R. and Manning, C. (2004). Deep dependencies from context-free statistical parsers: correcting the surface dependency approximation. In *Proceedings of ACL*.
- McDonald, R., Crammer, K., and Pereira, F. (2005a). Online large-margin training of dependency parsers. In *Proceedings of ACL*.
- McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005b). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of ACL*.
- Miller, P. (2000). *Strong Generative Capacity: The Semantics of Linguistic Formalism*. Cambridge.
- Pollard, C. (1984). *Generalized Phrase Structure Grammars, Head Grammars, and Natural Languages*. PhD thesis, Stanford.
- Vijay-Shanker, K., Weir, D. J., and Joshi, A. K. (1987). Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of ACL*.