Bounded-Depth High-Coverage Search Space for Noncrossing Parses

Anssi Yli-Jyrä
University of Helsinki, Finland
anssi.yli-jyra@helsinki.fi

Abstract

A recently proposed encoding for noncrossing digraphs can be used to implement generic inference over families of these digraphs and to carry out first-order factored dependency parsing. It is now shown that the recent proposal can be substantially streamlined without information loss. The improved encoding is less dependent on hierarchical processing and it gives rise to a *high-coverage bounded-depth approximation* of the space of noncrossing digraphs. This subset is presented elegantly by a *finite-state machine* that recognizes an infinite set of encoded graphs. The set includes more than 99.99% of the 0.6 million noncrossing graphs obtained from the UDv2 treebanks through planarisation. Rather than taking the low probability of the residual as a flat rate, it can be modelled with a joint probability distribution that is factorised into two underlying stochastic processes – the *sentence length distribution* and the related *conditional distribution for deep nesting*. This model points out that deep nesting in the streamlined code requires extreme sentence lengths. High depth is categorically out in common sentence lengths but emerges slowly at infrequent lengths that prompt further inquiry.

Syntactic and semantic dependency structures – rooted trees and more general digraphs – have tremendous importance in multilingual language analysis as demonstrated by the Universal Dependencies (UD) initiative[1] and many applications of dependency annotations. The main approaches to produce dependency structures include *graph-based parsers* (Eisner and Satta, 1999; McDonald et al., 2005) that build the structures in the bottom up fashion and *transition-based parsers* that produce structures while reading the input buffer (Nivre, 2008; Bohnet et al., 2016).

Recently, Yli-Jyrä and Gómez-Rodríguez (2017) have explored a perspective that combines graph-based parsing with coding theory: instead of rewriting digraphs directly, they propose a linear encoding for noncrossing digraphs and then manipulate the code strings using string automata. This method brings the two parsing approaches closer to each other as the graphical parsing reduces to a combination of state-driven processing of the underlying regular component and graphical processing of context-free component of the encoding. Their main result is that some 50 natural families of dependency structures reduce to unambiguous context-free languages. Generic parsing to noncrossing digraphs can thus be viewed as weighted context-free parsing.

The parsing objective in the framework of Yli-Jyrä and Gómez-Rodríguez (2017) is to maximize the total arc weight of the parse using an exact cubic-time inference procedure over the language associated with the input sentence. Polynomial time is often too expensive as such alternative methods as transition-based parsing with beam search may produce similar accuracy in linear time. Since higher efficiency is welcome in many real-time data applications, one may ask whether the encoded search space could be optimized to allow efficient, linear-time inference over *the most plausible* candidate graphs.

In this paper, we present an improved representation for the search space. First, the linear encoding of noncrossing graphs is streamlined by a technique called *weak edge bracketing*. Second, the context-free language of the streamlined en-

[1] http://universaldependencies.org/

Proceedings of the 13th International Conference on Finite State Methods and Natural Language Processing, pages 30–40,
Umeå, Sweden, 4–6 September 2017.
https://doi.org/10.18653/v1/W17-4004

coding is further *approximated* with a regular subset that contains the most probable dependency analyses. This gives us a finite-state representation of the search space. We also construct a *factored joint probability model* for the event that the correct parse is outside of the search space. Under the approximation and the current experiments, the probability for a failure is less than 0.3% for all languages and 0.006% on average.

The low error rate means that if the proposed approximation was used to restrict the search space of the state-of-the-art parsers, the obtained finite-state approximation would potentially improve the efficiency significantly while leaving the parsing accuracy nearly intact. When used in this way as a search space restriction, the currently proposed regular approximations for the families of digraphs – as proposed by Yli-Jyrä and Gómez-Rodríguez (2017) – become available to higher-order graphical parsers, neural transition-based parsers and generative neural models of syntax.

The current paper focuses on the structure and the motivation for the finite-state search space for noncrossing digraphs. Section 2 presents the problem of finding a good finite-state approximation of the search space. The streamlined context-free encoding for the noncrossing digraphs is introduced in Section 3. Sections 4 and 5 discuss its minimum-DFA state complexity and coverage under a bounded nesting depth. Before conclusion, the results are related to the prior work and discussed critically in Section 6.

1 Definitions

In this section, I follow Yli-Jyrä and Gómez-Rodríguez (2017) to define noncrossing dependency graphs and describe how they can be encoded as linear strings.

Graphs A *graph* is a pair (V,E) where V is a finite set of vertices and $E \subseteq \{\{u,v\} \subseteq V\}$ is a set of edges. It is common to assume that the edges do not contain self-loops of the form $\{v,v\}$. For convenience, the vertices in graphs are ordered $V = [1,...,n]$. Two edges $\{i,j\}$, $\{k,l\}$ in an ordered graph are said to be *crossing* if $\min\{i,j\} < \min\{k,l\} < \max\{i,j\} < \max\{k,l\}$. A graph is *noncrossing* if it has no crossing edges.

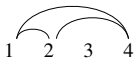
Yli-Jyrä and Gómez-Rodríguez (2017) have proposed a scheme according to which any noncrossing ordered graph $([1,...,n],E)$ is encoded as a string of brackets using the algorithm `enc` in

```
func enc(n,E):
   for i in [1,...,n]:
      for j in [i-1,...,2,1]:
         if {j,i} in E:
            print "]"
      for j in [n,n-1,...,i+1]:
         if {i,j} in E:
            print "["
      if i<n:
         print "{}"
```

```
func dec(stdin):
   n = 1; E = {}; s = []
   while c in stdin:
      if c == "[":
         s.push(n)
      if c == "]":
         i = s.pop()
         E.insert((i,n))
      if c == "{":
         n = n + 1
   return (n,E)
```

Figure 1: The encoding and decoding algorithms

Fig. 1. For example, the output for the ordered graph



$$n = 4, \quad E = \left\{ \begin{matrix} \{1,2\}, \\ \{2,4\}, \end{matrix} \quad \{1,4\} \right\}$$

is the string `[[{}][{}{}]]`. Intuitively, pairs of brackets of the form {} can be interpreted as spaces between vertices, and then each set of matching brackets `[...]` encodes an arc that covers the spaces represented inside the brackets.

The noncrossing ordered graphs are encoded with strings that constitute a context-free language. These *encoded graphs* are generated exactly by the context-free grammar $S \to [\,S'\,]\ S \mid \{\}\ S \mid \varepsilon$, $S' \to [\,S'\,]\ T \mid \{\}\ S$, $T \to [\,S'\,]\ S \mid \{\}\ S$. This language of encoded graph corresponds bijectively to the set of all non-crossing graphs.

Digraphs The encoding scheme extends to digraphs. A *digraph* is a pair (V,A) where $A \subseteq V \times V$ is a set of arcs $u \to v$. Its *underlying graph*, (V,E_A), has edges $E_A = \{\{u,v\} \mid (u,v) \in A\}$. A *noncrossing digraph* is a digraph whose underlying graph is noncrossing. Any noncrossing ordered digraph $([1,\ldots,n],A)$ can be encoded with slight modifications to the encoding algorithm. Instead of printing `[ ]` for an edge $\{i,j\} \in E_A$, $i \le j$, the algorithm should now print

$$\begin{array}{ll} \texttt{/ >} & \text{if } (i,j) \in A, (j,i) \notin A; \\ \texttt{< \textbackslash} & \text{if } (i,j) \notin A, (j,i) \in A; \\ \texttt{[ ]} & \text{if } (i,j), (j,i) \in A. \end{array}$$

In this way, we can simply encode the digraph $(\{1,2,3,4\},\{(1,2),(4,1),(4,2)\})$ as the string `</{}><{}{}\\`. Again, there is a bijection between noncrossing digraphs and their encodings, L_{DIGRAPHS}. All n-vertex noncrossing digraphs are represented with the language

$$L_{\text{DIGRAPHS}} \cap W^n \tag{1}$$

where $W^n = \overline{B}^*(\{\}\overline{B}^*)^{n-1}$ describes the alternation between vertex boundaries $\{\}$ and edge brackets $\overline{B}$ that exclude these boundaries.

Yli-Jyrä and Gómez-Rodríguez (2017) construct representations of context-free languages that encode important families of digraphs and graphs. Accordingly, there are context-free languages that correspond to the rooted noncrossing trees, projective trees, noncrossing dags etc.

Dependency Parsing The *complete digraph* (V,A) *of a sentence* $S = x_1 \ldots x_n$ consists of vertices $V = \{1,\ldots,n\}$ and all possible arcs $A = \{(i,j) \mid i \neq j\}$. In the *arc-factored* model (McDonald et al., 2005), every arc $i \to j$ in this digraph is equipped with a positive weight w_{ij} that is predicted on the basis of the feature vectors associated with tokens i and j in the sentence. To facilitate local weight assignment to the pairs of brackets in the encoding, the brackets in the encoded digraphs (1) can be indexed with the corresponding vertex numbers. This indexing turns string `[[{}][{}{}]]` into $[_1[_1\{\}]_2[_2\{\}\{\}]_4]_4$. The total weight of this string is computed using a dynamically constructed semiring-weighted context-free grammar

$$S \to \varepsilon \mid \{\};$$
$$S \overset{w_{12}}{\to} [_1 S]_2 S; S \overset{w_{13}}{\to} [_1 S]_3 S; S \overset{w_{14}}{\to} [_1 S]_4 S;$$
$$S \overset{w_{23}}{\to} [_2 S]_3 S; S \overset{w_{24}}{\to} [_2 S]_4 S; S \overset{w_{34}}{\to} [_3 S]_4 S.$$

Let $\otimes$ be a commutative monoid operation used to compute the total weight of a derivation under the grammar. For the above string, the grammar then returns the total weight $w_{12} \otimes w_{24} \otimes w_{14}$.

The task of *arc-factored dependency parsing* is to find in the specified family of the graphs (e.g. noncrossing dags), the maximal subgraph (V,A') of the complete digraph (V,A). When the inference is restricted to a noncrossing family L_{FAMILY} of digraphs, the natural choice is to carry out the inference using a cubic-time algorithm that recognises the weighted context-free language and finds the indexed string $w \in L_{\text{FAMILY}} \cap W_n$ that maximizes the total weight of the derivation.

2 The Problem

For long sentences, a linear-time parsing algorithm can be considerably more efficient and attractive than a cubic-time parsing algorithm. Since the encoded digraphs support generic parsing to different families of digraphs, we would now like to provide foundations for a linear time variant of this generic parsing architecture. The most obvious approach would be to replace the context-free grammar of (di)graphs with a regular subset approximation.

Since the approximation is regular, there are at most a finite number of equivalence classes over possible parser states at any moment. The number of equivalence classes tells the size of a deterministic finite automaton. The search space of an arbitrary sentence consists of those digraphs that are encoded by the intersection of the language L'_{FAMILY} recognized by this automaton and the language W_n defining the position-indexed brackets.

A good, practical approximation must satisfy at least three requirements:

- **Complete core**. A language-independent and generic approximation for the set of digraphs should certainly contain all digraphs over a small number ($n \leq 7$) of tokens.
- **Convenient size**. The amount of the memory needed to carry out inference over long sentences should not stretch the limits of a convenient implementation.
- **Good coverage**. The approximation should have a good coverage of the existing analyses in treebanks.

During the inference for the best parse, the memory of a typical algorithm stores the parse candidates either as a non-center-embedding grammar or a fully expanded finite-state automaton that represents the crossproduct of two finite-state representations, one for the search space and one for the strings with vertex indexed bracket. Since the convenience of the expanded representation is not at all obvious, the current work focuses on its deterministic state complexity, i.e. the number of states in a minimal DFAs recognizer.

With the encoding scheme of Yli-Jyrä and Gómez-Rodríguez (2017), the state complexity of the DFA-based search space representation grows rapidly when the length of the sentence increases. This comes from the fact that the states must keep track of four kinds of open brackets: `[`, `<`, `/`, `{`.

n	encoded digraphs	states
1	1	1
2	4	12
3	64	80
4	1 792	490
5	62 464	2 952
6	2 437 120	27 040
7	101 859 328	106 372

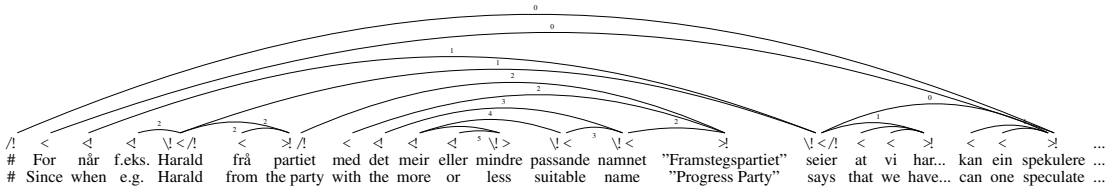


Figure 2: An example of a parse tree that requires weak bracketing with nesting depth 5.

By the **Complete core** requirement, we obtain a rough lower-bound state complexity for the required approximation L'_{FAMILY}. The search space for $n = 7$ equals to the intersection $W_{n=7} \cap L'_{\text{FAMILY}}$ whose state complexity is 106 372 states. Since the recognizer for W_7 requires exactly 14 states, the recognizer G for L'_{FAMILY} should contain at least $\lceil 106\,372/14 \rceil = 7\,598$ states as the cross product of these automata cannot have fewer than 106 372 states. With this 7 598-state automaton G, the parsing of a 500-token sentence would require at least $|W_{500}||G| = 7\,598\,000$ states.[2] But an automaton of this size is usually inconvenient to operate with and does not satisfy the **Convenient size** requirement.[3] Since only 6 overlapping edges are observed in 7-word sentences, this approximation would also fail to cover treebanks where more than 10 nested edges are quite common (Figure 2), breaking the **Good coverage** requirement. Thus, a DFA approximation based on the prior encoding is doomed to fail in the real life scenario.[4]

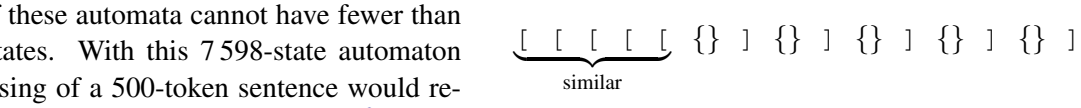
The current research problem is to improve the representation of the search space in such a way that fewer DFA states are needed and more complex structures can be captured with a convenient number of equivalence classes.

3 The Improved Encoding

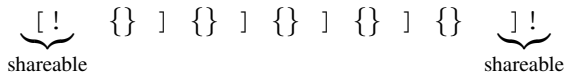
The prior encoding can be improved through weak, reduced bracketing that packs adjacent closing or opening brackets into a single symbol. The idea has historical links to superbrackets in Interlisp (Teitelman, 1978), but similar ideas have been introduced to the bracketing of phrase structure trees (Langendoen, 1975; Krauwer and des Tombe, 1981; Yli-Jyrä, 2003). In the context of edge brackets, the idea of weak bracketing appears partially in Yli-Jyrä (2004).

3.1 Nested Sibling Edges

The key observation is that nested sibling edges give rise to adjacent copies of one-sided brackets:



The idea is that if we indicate both sides of the outermost sibling edge, its nested siblings share one of its ends and thus need only a one-sided bracket:



Our current contribution to this idea is to observe that (1) both sides of the outermost brackets can be shared and (2) that every edge can be independently directed or undirected:

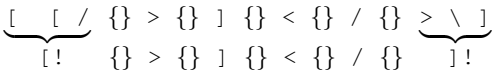


Figure 2 shows how this encoding is applied to a real dependency tree.

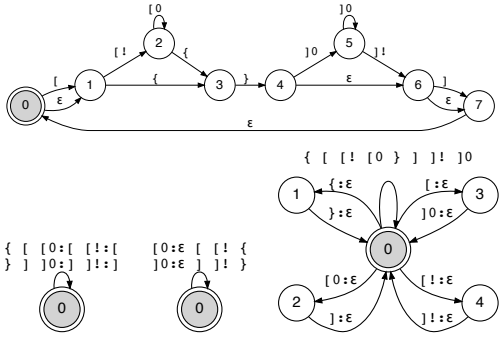


Figure 3: Language C and functions Y and E and relation T_4

[2] The seven longest sentences in the UDv2 dataset viewed by the author consist of 399, 428, 493, 496, 504, 534 and 610 tokens.

[3] A hierarchical grammar representation would be much more succinct but it assumes richer structures that we would like to preserve for optimizations of the implementation.

[4] We did not even consider the latent encoding of Yli-Jyrä and Gómez-Rodríguez. The latent encoding has a more complex local structure and requires drastically more states in a DFA representation.

3.2 Re-encoder

There is a transducer that converts the original "strong" edge bracketing into weak edge bracketing. The transducer is represented with four components in Figure 3:

1. Sequential function Y that replaces the *super brackets* `]!`, `[!`, the *redundant brackets* `]0`, `[0` and the *weak brackets* `]`, `[` with standard edge brackets `]`, `[`, respectively.

2. Language C that states the property that weak brackets `]`,`[` are not appearing at the position of redundant brackets and that the redundant brackets `]0`,`[0` are attached locally to a similar super bracket `]!`, `[!`.

3. Sequential function E that elides the redundant brackets.

4. A reflexive regular relation T_4 whose iterated application implements the fact that balanced brackets may cancel each other.

If X is a finite set of encoded graphs, we can re-encode these graphs as the corresponding set of weakly bracketed graphs X_w. We compute

$$X_w = \mathbf{Dom}(E^{-1} \circ \underbrace{\mathbf{Dom}(Y \circ X) \circ C \circ T_4 \circ \ldots \circ T_4}_{\ldots \circ T_4 \circ T_4 \text{ until fixed point}} \circ \varepsilon)$$

where **Dom** returns the input projection and X, C and ε are viewed as identity relations. Since the composition closure $T_4 \circ T_4 \circ \ldots \circ T_4 \circ \varepsilon$ maps the balanced bracket strings to the empty word, its input projection is actually a Dyck language D_4 — a balanced language over the four kinds of balanced brackets. With this context-free language, we can define the re-encoder as function

$$X_w = E(Y^{-1}(X) \cap C \cap D_4).$$

that maps the strongly bracketed strings X to the weakly bracketed strings X_w. This re-encoder function extends to encoded digraphs in the natural way by extending the set of brackets.

3.3 Improved State Complexity

The state complexity of the set of all n-vertex noncrossing digraphs for the streamlined encoding is essentially smaller than previously:

n	encoded digraphs	states
...		
5	62 464	207
6	2 437 120	704
7	101 859 328	1327

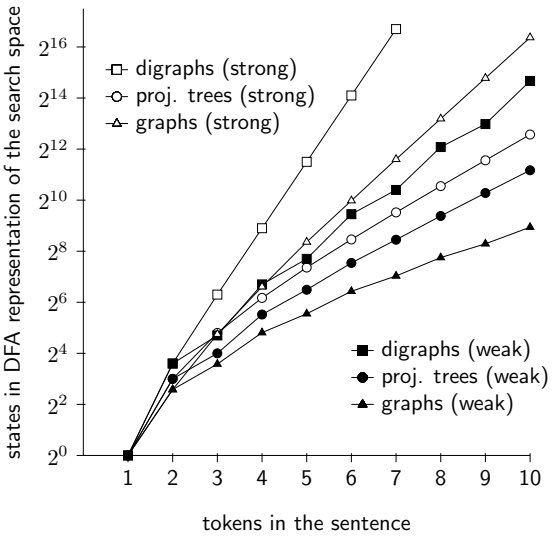


Figure 4: Weak bracketing brings exponential savings in the size of the DFA representing the search space of a sentence when the search space consists of digraphs, projective trees, or graphs

We are also able to go beyond $n = 7$ and build complete search spaces of undirected graphs with more vertices:

n	encoded graphs	states
10	21 292 032	490
19	29 312 424 612 462 592	12 395
20	314 739 971 287 154 688	18 276
21	3 393 951 437 605 044 224	24 925
30	$\approx 7.681 \cdot 10^{27}$	589 598

Figure 4 shows that the state complexity difference between the original and the new encoding scheme for digraphs, projective trees and graphs is indeed exponential to the length of the sentences. With the simple trick that replaces `/!` and `<!` with `[!`, but otherwise separates the brackets `/`, `>`, `]`, `[`, `<`, `\`, `[!`, `]!`, `\!`, `>!`, the search space generalizes from noncrossing graphs to noncrossing digraphs *without any increase in the state complexity*. This gives another significant saving in the state complexity of digraph search space compared to the original encoding.[5]

3.4 The Context-Free Set of Digraphs

There is an extended context-free grammar that generates all encoded undirected graphs using the

[5]Some further savings could be obtained by joining the adjacent curly brackets { } into an atomic symbol.

$$
\begin{aligned}
S' \rightarrow\ & \big(\{\} \mid S\big)^* \\
S \rightarrow\ & [\!\,!\ \Big(\{\} \mid T\big(\{\} \mid S\big)^* U\Big)\ ]\!\,! \text{ where} \\
T =\ & \{\} \mid \{\}\big(\{\} \mid S\big)^*\ ]\ \Big(\big(\{\} \mid S\big)^+\ ]\Big)^* \\
U =\ & \{\} \mid \Big([\big(\{\} \mid S\big)^+\Big)^*\ [\big(\{\} \mid S\big)^*\{\}
\end{aligned}
$$

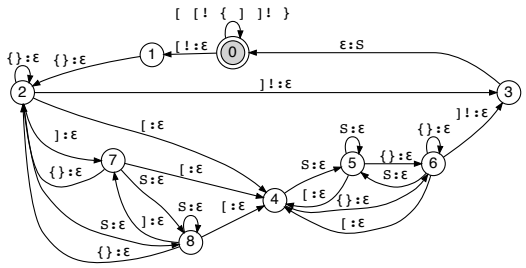


Figure 5: Top left: the level grammar. Top right: a bottom-up recognizer G_S for S.

weak edge bracketing:

$$S' \rightarrow S^* \tag{2}$$

$$S \rightarrow \{\} \mid [\!\,!\ \{\}\ ]\!\,! \mid [\!\,!\ T S^* U\ ]\!\,! \tag{3}$$

$$T \rightarrow \{\} \mid \{\} S^*\ ]\ \big(S S^*\ ]\big)^* \tag{4}$$

$$U \rightarrow \{\} \mid \big([\, S S^*\big)^*\ [\, S^* \{\} \tag{5}$$

The derivation steps of this grammar do not correspond exactly to nesting of brackets. The grammar can, however, be converted to an extended context-free grammar where each derivation step corresponds to a new nesting level. This is illustrated in the left of Figure 5.

We implement the grammar as a transducer that is iterated until the sentential form is in the language $\big(\{\} \mid S\big)^*$. The iterated transducer G_S is shown in the right of Figure 5.

4 Finite-State Search Space

A finite-state approximation is obtained from the grammar by composing copies of the bottom-up recognizer G_S and by restricting the output language to $\big(\{\} \mid \mathtt{S}\big)^*$. The language $L'_{\text{GRAPHS(5)}}$ is constructed using 6 copies of the G_S transducer:

$$\mathbf{Dom}(G_S \circ G_S \circ G_S \circ G_S \circ G_S \circ G_S \circ \big(\{\} \mid \mathtt{S}\big)^*)$$

The language $L'_{\text{GRAPHS(5)}}$ has 442 distinguishable DFA states and 1388 transitions. This approximation of the grammar can be used to represent most of the search space of n-vertex sentences with fewer states than the length-limited finite subset of context-free language L_{GRAPHS} that contains all graphs. When $L'_{\text{GRAPHS(5)}}$ and L_{GRAPHS} are constrained with W_n, we obtain regular languages whose state complexity can be measured. The difference between these is illustrated in Figure 6.

It is interesting that although the approximation captures only 6 levels (5 nested ones) of super brackets, it gives exact results until the graphs have 14 vertices. This is because each pair of superbrackets encode an edge whose two endpoints are non-incident with the edges that correspond to nested superbrackets.

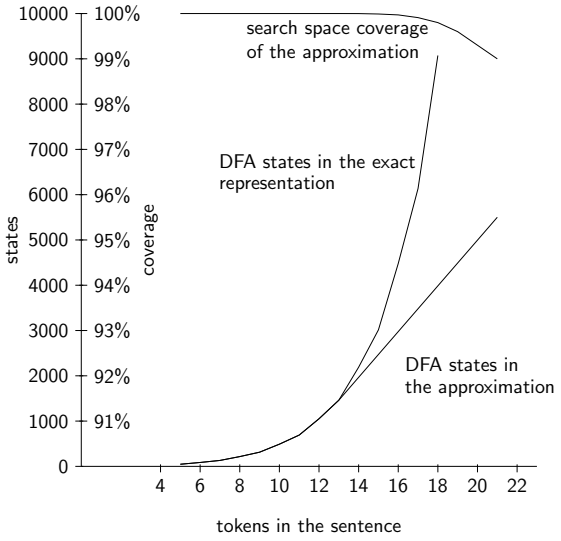


Figure 6: In terms of space complexity, the approximation grows only linearly to the sentence length while the exact search space representation explodes quickly. Meanwhile, the finite nesting depth has only a slowly growing proportional effect on the the search space when the length of the sentence increases.

Since the approximated search space grows linearly to the sentence length, the approximated search space of 21-vertex graphs requires 5 501 states. Interestingly, this subset approximation still covers 99% of the complete search space as it contains 3 358 682 892 406 358 016 graphs.

5 Data Coverage

The length of real sentences in treebanks and in texts varies a lot. At very high lengths, it is not easy to tell without real data how probable it is

Table 1: The coverage of UD v.2 data with depth bounded weak edge bracketing

lang	N	depth 0	depth 1	depth 2	depth 3	depth 4	depth 5	depth 6	depth 7
Arabic	26722	4.42%	20.93%	65.09%	94.39%	99.64%	99.99%	+0.011% (3)	
Catalan	14832	1.27%	19.01%	70.39%	96.07%	99.62%	99.99%	+0.007% (1)	
Czech	102660	10.60%	43.11%	86.77%	98.47%	99.89%	99.99%	+0.010% (10)	
German	14917	2.06%	43.11%	87.52%	98.56%	99.91%	99.97%	+0.027% (4)	
English	19785	16.61%	53.59%	91.10%	99.21%	99.96%	100.00%		
Spanish	31546	1.59%	24.61%	77.27%	97.24%	99.79%	100.00%	+0.003% (1)	
Finnish	30437	30.03%	77.46%	96.54%	99.55%	99.96%	99.99%	+0.007% (2)	
French	19294	2.86%	37.87%	88.24%	98.89%	99.94%	99.99%	+0.005% (1)	
Greek	28478	16.47%	67.77%	95.75%	99.73%	99.98%	100.00%		
Hebrew	5725	1.61%	24.63%	80.61%	98.72%	99.93%	100.00%		
Hindi	14963	0.45%	37.19%	82.74%	98.04%	99.90%	99.98%	+0.020% (3)	
Croatian	8289	2.23%	33.78%	86.39%	98.90%	99.93%	100.00%		
Hungarian	1351	1.11%	22.95%	68.91%	94.89%	98.96%	99.78%	+0.148% (2)	+0.074% (1)
Italian	14992	4.93%	47.62%	88.22%	98.63%	99.92%	100.00%		
Japanese	7675	1.98%	48.60%	96.73%	99.96%	100.00%	100.00%		
Latin	33166	20.14%	61.14%	90.57%	98.86%	99.94%	100.00%		
Latvian	3054	16.50%	57.50%	88.93%	98.23%	99.80%	99.97%		+0.033% (1)
Dutch	19891	18.32%	60.86%	93.81%	99.56%	99.98%	100.00%		
Polish	7127	13.03%	75.28%	98.58%	99.94%	99.99%	100.00%		
Portuguese	19765	4.61%	32.02%	81.33%	97.98%	99.83%	99.99%	+0.005% (1)	
Romanian	8795	0.80%	25.78%	84.51%	98.53%	99.87%	99.99%	+0.011% (1)	
Russian	59827	12.57%	73.86%	97.03%	99.76%	99.98%	100.00%	+0.002% (1)	
Slovenian	9349	17.82%	57.60%	94.34%	99.68%	99.99%	100.00%		
Scandinavian	47574	12.46%	55.27%	92.52%	99.35%	99.96%	100.00%		
Turkish	4660	22.08%	71.20%	92.68%	98.88%	99.89%	99.98%	+ 0.021% (1)	
Chinese	4497	0.00%	19.64%	70.49%	94.53%	99.51%	99.96%	+ 0.044% (2)	
other UD treebanks	71147	12.55%	57.43%	91.58%	99.21%	99.96%	100.00%		
	630518	11.07%	50.38%	88.38%	98.66%	99.91%	99.994%	+ 0.005% (33)	+ 0.0003% (2)

that a finite-state approximation does not contain the correct analysis graph or digraph. In order to learn about the probability of the out-of-the-search-space event, we used the UD treebanks as the first proxy to find out how often a given nesting depth is exceeded in gold trees.

Our current encoding can handle only noncrossing trees such as projective trees. However, the trees in UD version 2 treebanks do not have this restriction. It is well known that the proportion of non-projective, and thus crossing, trees is relatively high for some languages (Gómez-Rodríguez, 2016). If all nonprojective analyses were discarded, most of the long sentences would have been excluded, with significant effect on our experiments.

For the experiments, we had to enforce noncrossing structure to the data. The standard approach is to perform lift transformations that move crossing edges higher in the dependency tree. Although there are methods to minimize the number of lifts either heuristically or exactly, the output of such a transformation is not uniquely determined. The second way to projectivise trees is to keep the dominance tree intact but reorder the nodes of the tree into a canonical order that maintains the relative order of the immediate dependents of each head. The third approach, actually used by us, views the trees as undirected trees and takes advantage of the algebraic properties of balanced bracketing. For example, if we combine the edge brackets in `[{}{}] [{}]` and `{} [{}{}]`, we obtain `[{} [{}] [{}] ]` that encodes a slightly different set of edges. This method works also with reduced bracketing where the opening and closing superbrackets cancel one another. The resulting code string preserves the number of edges but some ends of these edges may change. The result may be a cyclic or disconnected graph but it preserves noncrossing parts of the graph intact because there the brackets match the two ends of each edge.

The total number of sentences in the sample was 630 518. This includes both the training and the development sections of the UD v.2 treebanks. The data was encoded with our new encoding scheme and automatically converted to noncrossing undirected graphs before the nesting depth of each sentence was computed.[6]

Table 1 describes how the nesting depth corresponds to coverage in the UD2 data set. It indicates that the depth 5 is a good compromise between depth and coverage. Under this depth, the search space contains 99.994% of all the trees in the treebanks. Only 35 sentences require weak bracketing whose nesting depth is more than 5. Thus the flat out-of-the-search-space failure rate is 0.006% of the sentences only.

[6]The code for the encoding script is in https://github.com/amikael/depconv.

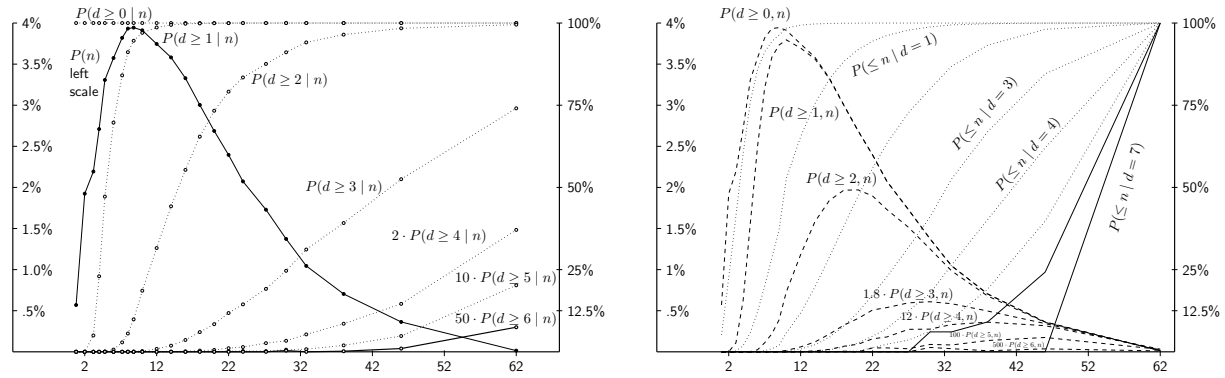


Figure 7: Left: the sentence length distribution $P(n\text{-bucket})$ =count($n-$bucket)/all sentences and the conditional probability $P(d \geq k \mid n)$ =count($d,n-$bucket)/count($n-$bucket) of exceeding depth of weak bracketing, given the sentence length. Right: the joint distribution $P(d \geq k \mid n\text{-bucket})$ =count($d,n-$bucket)/bucketsize/total for the event of exceeding depth, given sentence length and the conditional distribution $P(\leq n \mid d)$ =count($\leq n-$bucket,d)/count(d) of maximum sentence length, given the bracketing depth. Scaling of some distributions was necessary as the numeric values of some nearby distributions are of different orders of magnitude.

The error rate leaves us with doubts about the representativeness of the sample. To gain more insight into the underlying process, a more sophisticated statistical model was constructed. For this purpose, the multilingual data was smoothed by splitting the sentence lengths into buckets that contained typically some 30k sentences. The length ranges were $[2,2],...,\ [9,9],\ [10,11],...,\ [24,25],\ [26,28],[29,31],[32,35],[36,41],[42,52],[53,610]$. Each was represented by the length of the median sentence in the range. E.g. the range [53,610] was represented by the median length 62.

The sentence length and nesting depth combine to form a joint distribution $P(d > k,n)$ that can be factored in different ways. Figure 7 presents the distribution of sentence lengths and the related conditional and joint distributions as they could be observed. The statistics reveal the following observations:

1. The shape of the sentence length distribution resembles a Poisson distribution.[7]
2. Although the sentence length increases, the probability of shallow bracketing still remains high (e.g. $P(d < 4 \mid n \in [42,52]) = 0.93\%$), as the probability of a deep nesting event increases slowly according to the sentence length.
3. The observed probability of a deep sentence grows almost linearly when the sentence length increases. Thus, high sentence length does not necessarily mean very deep nesting ($P(d \geq 6 \mid n \in [42,610]) < 0.2\%$). Even extremely long sentences are rarely deep ($P(d \geq 6 \mid n \in [53,610]) = 0.15\%$).
4. The current experiments seem to break the direct link from long sentences to deep nesting but supports the opposite tendency. The probability of a sentence being deep is zero when the number of tokens is less than 14, but very deep nesting predicts high sentence length ($P(n > 45 \mid d = 6) > 99\%$).
5. The proportion of deep analyses in the search space is higher than the corresponding proportion in the real data for each length. In particular, deep nesting in sentences with 20-21 tokens is expected to happen almost at the probability 1% if all analyses were equally probable (Figure 6), but observed probability of the event is under 0.002% ($P(d \geq 6 \mid n \in [20,21]) = 0.002\%$).

6 Discussion

There are three factors that contribute to deep nesting structures in bracketed binarised trees:

- **Tail Recursion**. Repeated left- and right-branching structures generate initial and final forms of tail recursion, as well as zigzag embedding that involves both.

[7] For a sophisticated sentence length model, see e.g. Sichel (1974).

- **Unit Rules**. Non-branching trees embed trees in a way that may generate new nesting levels in bracketing.

- **Local Factorization**. Local factorization of unranked parse trees and unbounded sibling edges can generate unbounded recursion.

Unit rules and tail recursion have been addressed in several prior studies as their naive bracketing may involve unbounded stack or nested brackets. The prior approaches include grammar transformations in parsers (Langendoen, 1975; Johnson, 1998; Nederhof, 2000), weak bracketing of constituent trees (Langendoen and Langsam, 1984; Krauwer and des Tombe, 1981; Black, 1989; Koskenniemi, 1990; Yli-Jyrä, 2003), edge bracketing of dependency trees (Oflazer, 2003; Yli-Jyrä, 2005, 2012; Yli-Jyrä et al., 2012) and further ideas (Church, 1980; Langendoen, 2008; Hulden and Silfverberg, 2014).

The current work goes another mile in the study of bounded nesting by introducing weak edge bracketing that avoids unbounded recursion when unbounded local trees are binarized and bracketed.[8] The work also shows that the classic technique can be extended to all noncrossing digraphs. This extends the relevance of finite-state transducer techniques from syntax to semantic dependency parsing.

The weak edge bracketing is effective in improving the state complexity of finite sets of noncrossing digraphs (Figure 4). Thanks to the new bracketing scheme, the coverage of depth-bounded approximations is very high and seems to deteriorate slower than the state complexity of the exact search space grows (Figure 6). A further improvement was obtained by the trick that maintained the state complexity when the encoding for graphs was generalized to digraphs.

A careful comparison between methods that ensure noncrossing structures could reveal some interesting differences. While the nonprojective trees can be projectivised (and thus turned into noncrossing graphs) with optimal number of lifts, the lifted tree is still not unique and it may be intractable to optimize the nesting depth over the choices. It may also be a problem if the input is not a tree and therefore the existing graph banks should be also explored in the statistics.

[8] The idea started to develop already in Yli-Jyrä (2004).

Natural next steps in the proposed methodology would be to adapt the parametrizable parsing framework of Yli-Jyrä and Gómez-Rodríguez (2017) to weak bracketing and to implement a linear-time arc-factored parser to important families of non-crossing digraphs. It is also important to find ways to combine the current encoding with higher-order dependency parsing and neural network grammars that weight local trees. The main challenge is, however, to extend the current work to crossing graphs and nonprojective trees.

7 Conclusion

In this paper, we have presented the first weak edge bracketing scheme for noncrossing digraphs and described the bijectively related context-free language of code strings. Our scheme improves over (Yli-Jyrä and Gómez-Rodríguez, 2017) as the minimum DFA state complexity of search space representations is smaller and unranked dependencies can be processed without recursion.

The experiments indicate that 5 nested levels of balanced brackets are sufficient to cover nearly all noncrossing UD v.2 data. According the joint distribution of length and depth, deep nesting is a rare event that correlates with an exceptional sentence length.

The language $L'_{\mathrm{GRAPHS}(5)}$ representing the subset approximation of the search space for all noncrossing graphs with nesting depth 5 can be applied beyond the Yli-Jyrä and Gómez-Rodríguez (2017) framework to guide and restrict syntactic parsing and generation. This is expected to lead to linear time syntactic and semantic dependency parsing with noncrossing output structures. If combined with multiplanar/book embedding methods (Yli-Jyrä, 2003; Kuhlmann and Johnsson, 2015), the techniques might extend to non-projective parsing and crossing graphs.

Acknowledgements

The author has received funding as Research Fellow from the Academy of Finland (dec. No 270354 - A Usable Finite-State Model for Adequate Syntactic Complexity) and Clare Hall Fellow from the University of Helsinki (dec. RP 137/2013 - SMT based on D-Tree Contraction Grammars and FSTs). He is indebted to Kimmo Koskenniemi, Jussi Piitulainen, Carlos Gómez-Rodríguez and the anonymous reviewers for helpful comments on earlier stages of the research.

References

Alan W. Black. 1989. Finite state machines from feature grammars. In Masaru Tomita, editor, *International Workshop on Parsing Technologies*, Carnegie Mellon Univeristy Press, Pittsburgh, Pennsylvania, pages 277–285.

Bernd Bohnet, Emily Pitler, Ji Ma, and Ryan Mcdonald. 2016. Generalized transition-based dependency parsing. In *Association for Computational Linguistics (ACL)*. https://www.aclweb.org/anthology/P16-1015.

Kenneth Church. 1980. On parsing strategies and closure. In *18th ACL 1980, Proceedings of the Conference*. Philadelphia, Pennsylvania, USA, pages 107–111. http://www.aclweb.org/anthology/P80-1028.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and Head Automaton Grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, College Park, Maryland, USA, pages 457–464. https://doi.org/10.3115/1034678.1034748.

Carlos Gómez-Rodríguez. 2016. Restricted non-projectivity: Coverage vs. efficiency. *Computational Linguistics* 42(4):809–817. https://doi.org/10.1162/COLI_a_00267.

Mans Hulden and Miikka Silfverberg. 2014. Finite-state subset approximation of phrase structure. In *International Symposium on Artificial Intelligence and Mathematics*. Fort Lauderdale, USA.

Mark Johnson. 1998. Finite-state approximation of constraint-based grammars using left-corner grammar transforms. In *36th ACL 1998, 17th COLING 1998, Proceedings of the Conference*. Montréal, Quebec, Canada, volume 1, pages 619–623. http://aclweb.org/anthology/P/P98/P98-1101.pdf.

Kimmo Koskenniemi. 1990. Finite-state parsing and disambiguation. In Hans Karlgren, editor, *13th COLING 1990, Proceedings of the Conference*. Helsinki, Finland, volume 2, pages 229–232. https://doi.org/0.3115/997939.997979.

Steven Krauwer and Louis des Tombe. 1981. Transducers and grammars as theories of language. *Theoretical Linguistics* 8:173–202. https://doi.org/10.1515/thli.1981.8.1-3.173.

Marco Kuhlmann and Peter Johnsson. 2015. Parsing to noncrossing dependency graphs. *Transactions of the Association for Computational Linguistics* 3:559–570. http://aclweb.org/anthology/Q/Q15/Q15-1040.pdf.

D. Terence Langendoen. 1975. Finite-state parsing of phrase-structure languages and the status of readjustment rules in grammar. *Linguistic Inquiry* VI(4):533–554. http://www.jstor.org/stable/4177899.

D. Terence Langendoen. 2008. Coordinate grammar. *Language* 84(4):691–709. http://www.jstor.org/stable/40071100.

D. Terence Langendoen and Yedidyah Langsam. 1984. The representation of constituent structures for finite-state parsing. In *22th ACL 1984, 10th COLING 1984, Proceedings of the Conference*. Stanford, CA, USA, pages 24–27. http://www.aclweb.org/anthology/P84-1007.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Vancouver, British Columbia, Canada, pages 523–530. http://www.aclweb.org/anthology/H/H05/H05-1066.pdf.

Mark-Jan Nederhof. 2000. Practical experiments with regular approximation of context-free languages. *Computational Linguistics* 26(1):17–44. https://doi.org/10.1162/089120100561610.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34(4):513–553. http://wing.comp.nus.edu.sg/ antho/J/J08/J08-4003.pdf.

Kemal Oflazer. 2003. Dependency parsing with an extended finite-state approach. *Computational Linguistics* 29(4):515–544. https://doi.org/10.1162/089120103322753338.

H. S. Sichel. 1974. On a distribution representing sentence-length in written prose. *Journal of the Royal Statistical Society* 137(1):25–34. http://www.jstor.org/stable/2345142.

Warren Teitelman. 1978. *INTERLISP Reference Manual*. Xerox Palo Alto Research Center, Xerox Corporation. (Section 2.2 Using Interlisp – an Overview, 2.4).

Anssi Yli-Jyrä. 2003. Regular approximations through labeled bracketing. In Gerald Penn, editor, *Proceedings of FGVienna: The 8th Conference on Formal Grammar*. CSLI Publications, Stanford University, Stanford, CA, pages 153–166. http://cslipublications.stanford.edu/FG/2003/ylijyra.pdf.

Anssi Yli-Jyrä. 2004. Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyse crossing bracketings. In Geert-Jan M. Kruijff and Denys Duchier, editors, *COLING 2004 Recent Advances in Dependency Grammar*. COLING, Geneva, Switzerland, pages 25–32. https://www.aclweb.org/anthology/W/W04/W04-1504.pdf.

Anssi Yli-Jyrä. 2005. Approximating dependency grammars through intersection of star-free regular languages. *Int. J. Found. Comput. Sci.* 16(3):565–579. https://doi.org/10.1142/S0129054105003169.

Anssi Yli-Jyrä. 2012. On dependency analysis via contractions and weighted FSTs. In Diana Santos, Krister Lindén, and Wanjiku Ng'ang'a, editors, *Shall We Play the Festschrift Game?, Essays on the Occasion of Lauri Carlson's 60th Birthday*. Springer, Berlin Heidelberg, pages 133–158. https://doi.org/10.1007/978-3-642-30773-7_10.

Anssi Yli-Jyrä and Carlos Gómez-Rodríguez. 2017. Generic axiomatization of families of noncrossing graphs in dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Vancouver, Canada, pages 1745–1755. http://aclweb.org/anthology/P17-1160.

Anssi Yli-Jyrä, Jussi Piitulainen, and Atro Voutilainen. 2012. Refining the design of a contracting finite-state dependency parser. In Iñaki Alegria and Mans Hulden, editors, *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*. Donostia–San Sebastián, Spain, pages 108–115. http://www.aclweb.org/anthology/W12-6218.

Anssi Mikael Yli-Jyrä. 2003. Multiplanarity – a model for dependency structures in treebanks. In Joakim Nivre and Erhard Hinrichs, editors, *TLT 2003. Proceedings of the Second Workshop on Treebanks and Linguistic Theories*. Växjö University Press, pages 189–200.