Optimal Parsing Strategies for Linear Context-Free Rewriting Systems

Daniel Gildea
Computer Science Department
University of Rochester
Rochester, NY 14627

Abstract

Factorization is the operation of transforming a production in a Linear Context-Free Rewriting System (LCFRS) into two simpler productions by factoring out a subset of the nonterminals on the production's righthand side. Factorization lowers the rank of a production but may increase its fan-out. We show how to apply factorization in order to minimize the *parsing complexity* of the resulting grammar, and study the relationship between rank, fan-out, and parsing complexity. We show that it is always possible to obtain optimum parsing complexity with rank two. However, among transformed grammars of rank two, minimum parsing complexity is not always possible with minimum fan-out. Applying our factorization algorithm to LCFRS rules extracted from dependency treebanks allows us to find the most efficient parsing strategy for the syntactic phenomena found in non-projective trees.

1 Introduction

Gómez-Rodríguez et al. (2009a) recently examined the problem of transforming arbitrary grammars in the Linear Context-Free Rewriting System (LCFRS) formalism (Vijay-Shankar et al., 1987) in order to reduce the rank of a grammar to 2 while minimizing its fan-out. The work was motivated by the desire to develop efficient chart-parsing algorithms for non-projective dependency trees (Kuhlmann and Nivre, 2006) that do not rely on the independence assumptions of spanning tree algorithms (McDonald et al., 2005). Efficient parsing algorithms for general LCFRS are also relevant in the context of Synchronous Context-Free Grammars (SCFGs) as a formalism for machine translation, as well as the desire to handle even more general synchronous grammar formalisms which allow nonterminals to cover discontinuous spans in either language (Melamed et al., 2004; Wellington et al., 2006). LCFRS provides a very general formalism which subsumes SCFGs, the Multitext Grammars of Melamed et al. (2004), as well as mildly context-sensitive monolingual formalisms such as Tree Adjoining Grammar (Joshi and Schabes, 1997). Thus, work on transforming general LCFRS grammars promises to be widely applicable in both understanding how these formalisms interrelate, and, from a more practical viewpoint, deriving efficient parsing algorithms for them.

In this paper, we focus on the problem of transforming an LCFRS grammar into an equivalent grammar for which straightforward application of dynamic programming to each rule yields a tabular parsing algorithm with minimum complexity. This is closely related, but not equivalent, to the problem considered by Gómez-Rodríguez et al. (2009a), who minimize the fan-out, rather than the parsing complexity, of the resulting grammar. In Section 4, we show that restricting our attention to factorized grammars with rank no greater than 2 comes at no cost in parsing complexity. This result may be surprising, as Gómez-Rodríguez et al. (2009a) comment that "there may be cases in which one has to find an optimal trade-off between rank and fan-out" in order to minimize parsing complexity – in fact, no such trade-off is necessary, as rank 2 is always sufficient for optimal parsing complexity. Given this fact, we show how to adapt the factorization algorithm of Gómez-Rodríguez et al. (2009a) to return a transformed grammar with minimal parsing complexity and rank 2. In Section 5, we give a

counterexample to the conjecture that minimal parsing complexity is possible among binarizations with minimal fan-out.

2 Background

A linear context-free rewriting system (LCFRS) is defined as a tuple $G = (V_N, V_T, P, S)$, where V_T is a set of terminal symbols, V_N is a set of nonterminal symbols, P is a set of productions, and $S \in V_N$ is a distinguished start symbol. Associated with each nonterminal B is a **fan-out** $\varphi(B)$, which tell how many discontinuous spans B covers. Productions $p \in P$ take the form:

$$p : A \rightarrow g(B_1, B_2, \ldots, B_r) \tag{1}$$

where $A, B_1, \ldots B_r \in V_N$, and g is a function

$$g : (V_T^*)^{\varphi(B_1)} \times \ldots \times (V_T^*)^{\varphi(B_r)} \rightarrow (V_T^*)^{\varphi(A)}$$

which specifies how to assemble the $\sum_{i=1}^{r} \varphi(B_i)$ spans of the righthand side nonterminals into the $\varphi(A)$ spans of the lefthand side nonterminal. The function g must be **linear, non-erasing**, which means that if we write

$$g(\langle x_{1,1}, \ldots, x_{1,\varphi(B_1)}\rangle, \ldots, \langle x_{1,1}, \ldots, x_{1,\varphi(B_r)}\rangle) = \langle t_1, \ldots, t_{\varphi(A)}\rangle$$

the tuple of strings $\langle t_1, \ldots, t_{\varphi(A)}\rangle$ on the righthand side contains each variable $x_{i,j}$ from the lefthand side exactly once, and may also contain terminals from V_T.

We call r, the number of nonterminals on the righthand side of a production p, the **rank** of p, $\rho(p)$. The fan-out of a production, $\varphi(p)$ is the fan-out of its lefthand side, $\varphi(A)$. The rank of a grammar is the maximum rank of its rules,

$$\rho(G) = \max_{p \in P} \rho(p)$$

and similarly the fan-out of a grammar is the maximum fan-out of its rules, or equivalently, of its nonterminals:

$$\varphi(G) = \max_{B \in V_N} \varphi(B)$$

3 Parsing LCFRS

A bottom-up dynamic programming parser can be produced from an LCFRS grammar by generalizing the CYK algorithm for context-free grammars. We convert each production of the LCFRS into a deduction rule with variables for the left and right endpoints of each of the $\varphi(B_i)$ spans of each of the nonterminals $B_i, i \in [r]$ in the righthand side of the production.

The computational complexity of the resulting parser is polynomial in the length of the input string, with the degree of the polynomial being the number of distinct endpoints in the most complex production. Thus, for input of length n, the complexity is $O(n^c)$ for some constant c which depends on the grammar.

For a given rule, each of the r nonterminals has $\varphi(B_i)$ spans, and each span has a left and right endpoint, giving an upper bound of $c \leq 2\sum_{i=1}^{r} \varphi(B_i)$. However, some of these endpoints may be shared between nonterminals on the righthand side. The exact number of distinct variables for the dynamic programming deduction rule can the written

$$c(p) = \varphi(A) + \sum_{i=1}^{r} \varphi(B_i) \tag{2}$$

where $c(p)$ is the **parsing complexity** of a production p of the form of eq. 1 (Seki et al., 1991). To see this, consider counting the left endpoint of each span on the lefthand side of the production, and the right endpoint of each span on the righthand side of the production. Any variable corresponding to the left endpoint of a span of a righthand side nonterminal will either be shared with the right endpoint of another span if two spans are being joined by the production, or, alternatively, will form the left endpoint of a span of A. Thus, each distinct endpoint in the production is counted exactly once by eq. 2.

The parsing complexity of a grammar, $c(G)$, is the maximum parsing complexity of its rules. From eq. 2, we see that $c(G) \leq (\rho(G) + 1)\varphi(G)$. While we focus on the time complexity of parsing, it is interesting to note the space complexity of the DP algorithm is $O(n^{2\varphi(G)})$, since the DP table for each nonterminal is indexed by at most $2\varphi(G)$ positions in the input string.

4 Binarization Minimizes Parsing Complexity

An LCFRS production of rank r can be **factorized** into two productions of the form:

$$p_1 : A \rightarrow g_1(B_1, \ldots, B_{r-2}, X)$$
$$p_2 : X \rightarrow g_2(B_{r-1}, B_r)$$

This operation results in new productions that have lower rank, but possibly higher fan-out, than the original production.

If we examine the DP deduction rules corresponding to the original production p, and the first new production p_1 we find that

$$c(p_1) \leq c(p)$$

regardless of the function g of the original production, or the fan-out of the production's nonterminals. This is because

$$\varphi(X) \leq \varphi(B_{r-1}) + \varphi(B_r)$$

that is, our newly created nonterminal X may join spans from B_{r-1} and B_r, but can never introduce new spans. Thus,

$$\begin{aligned} c(p_1) &= \varphi(A) + \left(\sum_{i=1}^{r-2} \varphi(B_i)\right) + \varphi(X) \\ &\leq \varphi(A) + \sum_{i=1}^{r} \varphi(B_i) \\ &= c(p) \end{aligned}$$

As similar result holds for the second newly created production:

$$c(p_2) \leq c(p)$$

In this case, the fan-out of the newly created nonterminal, $\varphi(X)$ may be greater than $\varphi(A)$. Let us consider the left endpoints of the spans of X. Each left endpoint is either also the left endpoint of a span of A, or is the right endpoint of some nonterminal not included in X, that is, one of $B_1, \ldots B_{r-2}$. Thus,

$$\varphi(X) \leq \varphi(A) + \sum_{i=1}^{r-2} \varphi(B_i)$$

and applying this inequality to the definition of $c(p_2)$ we have:

$$\begin{aligned} c(p_2) &= \varphi(X) + \varphi(B_{r-1}) + \varphi(B_{r-2}) \\ &\leq \varphi(A) + \sum_{i=1}^{r} \varphi(B_i) \\ &= c(p) \end{aligned}$$

For notational convenience, we have defined the factorization operation as factoring out the last two nonterminals of a rule; however, the same operation can be applied to factor out any subset of the original nonterminals. The same argument that parsing complexity cannot increase still applies.

We may apply the factorization operation repeatedly until all rules have rank 2; we refer to the resulting grammar as a **binarization** of the original LCFRS. The factorization operation may increase the fan-out of a grammar, but never increases its parsing complexity. This guarantees that, if we wish to find the transformation of the original grammar having the lowest parsing complexity, it is sufficient to consider only binarizations. This is because any transformed grammar having more than two nonterminals on the righthand side can be binarized without increasing its parsing complexity.

5 The relationship between fan-out and parsing complexity

Gómez-Rodríguez et al. (2009a) provide an algorithm for finding the binarization of an LCFRS having minimal fan-out. The key idea is to search over ways of combining subsets of a rule's righthand side nonterminals such that subsets with low fan-out are considered first; this results in an algorithm with complexity polynomial in the rank of the input rule, with the exponent depending on the resulting minimum fan-out.

This algorithm can be adapted to find the binarization with minimum parsing complexity, rather than minimum fan-out. We simply use $c(p)$ rather than $\varphi(p)$ as the score for new productions, controlling both which binarizations we prefer and the order in which they are explored.

An interesting question then arises: does the binarization with minimal parsing complexity also have minimal fan-out? A binarization into a grammar of

$$A \rightarrow g(B_1, B_2, B_3, B_4)$$
$$g(\langle x_{1,1}, x_{1,2}\rangle,\ \langle x_{2,1}, x_{2,2}, x_{2,3}\rangle,\ \langle x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5}\rangle,\ \langle x_{4,1}, x_{4,2}, x_{4,3}\rangle) =$$
$$\langle x_{4,1}x_{3,1}, x_{2,1}, x_{4,2}x_{1,1}x_{2,2}x_{4,3}x_{3,2}x_{2,3}x_{3,3}, x_{1,2}x_{3,4}, x_{3,5}\rangle$$

Figure 2: A production for which minimizing fan-out and minimizing parsing complexity are mutually exclusive.

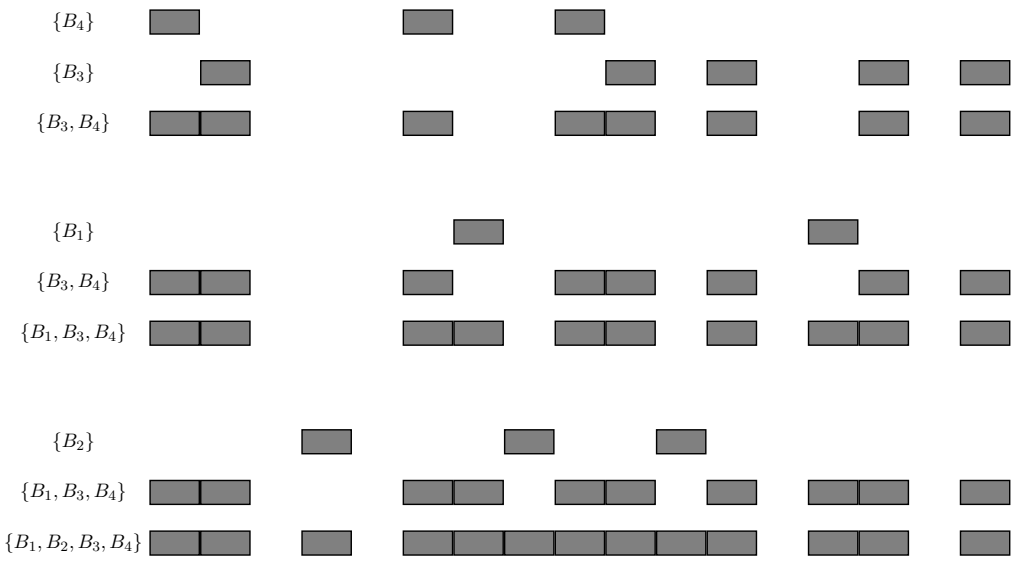


Figure 3: The binarization of the rule from Figure 2 that minimizes parsing complexity. In each of the three steps, we show the spans of each of the two subsets of the rule's righthand-side nonterms being combined, with the spans of their union (corresponding to a nonterminal created by the binarization) below.

```
1: function MINIMAL-BINARIZATION(p, ≺)
2:   workingSet ← ∅;
3:   agenda ← priorityQueue(≺);
4:   for i from 1 to ρ(p) do
5:     workingSet ← workingSet ∪{B_i};
6:     agenda ← agenda ∪{B_i};
7:   while agenda ≠ ∅ do
8:     p' ← pop minimum from agenda;
9:     if nonterms(p') = {B_1, ... B_ρ(p)} then
10:      return p';
11:    for p_1 ∈ workingSet do
12:      p_2 ← newProd(p', p_1);
13:      find p'_2 ∈ workingSet :
14:                  nonterms(p'_2) = nonterms(p_2);
15:      if p_2 ≺ p'_2 then
16:        workingSet ← workingSet ∪{p_2}\{p'_2};
17:        push(agenda, p_2);
```

Figure 1: Algorithm to compute best binarization according to a user-specified ordering $\prec$ over productions.

fan-out f' cannot have parsing complexity higher than $3f'$, according to eq. 2. Thus, minimizing fan-out puts an upper bound on parsing complexity, but is not guaranteed to minimize it absolutely. Binarizations with the same fan-out may in fact vary in their parsing complexity; similarly binarizations with the same parsing complexity may vary in their fan-out. It is not immediately apparent whether, in order to find a binarization of minimal parsing complexity, it is sufficient to consider only binarizations of minimal fan-out.

To test this conjecture, we adapted the algorithm of Gómez-Rodríguez et al. (2009a) to use a priority queue as the agenda, as shown in Figure 1. The algorithm takes as an argument an arbitrary partial ordering relation on productions, and explores possible binarized rules in the order specified by this relation. In Figure 1, "workingSet" is a set of singleton nonterminals and binarized productions which are guaranteed to be optimal for the subset of nonterminals that they cover. The function "nonterms" returns, for a newly created production, the subset of the original nonterminals $B_1, \ldots B_r$ that it generates, and returns subsets of singleton nonterminals directly.

To find the binarization with the minimum fan-out f' and the lowest parsing complexity among binarizations with fan-out f', we use the following comparison operation in the binarization algorithm:

$$p_1 \prec_{\varphi c} p_2 \text{ iff } \varphi(p_1) < \varphi(p_2) \vee (\varphi(p_1) = \varphi(p_2) \wedge c(p_1) < c(p_2))$$

guaranteeing that we explore binarizations with lower fan-out first, and, among binarizations with equal fan-out, those with lower parsing complexity first. Similarly, we can search for the binarization with the lowest parsing complexity c' and the lowest fan-out among binarizations with complexity c', we use

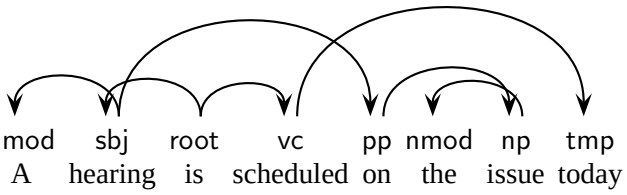
$$p_1 \prec_{c\varphi} p_2 \text{ iff } c(p_1) < c(p_2) \vee (c(p_1) = c(p_2) \wedge \varphi(p_1) < \varphi(p_2))$$

We find that, in fact, it is sometimes necessary to sacrifice minimum fan-out in order to achieve minimum parsing complexity. An example of an LCFRS rule for which this is the case is shown in Figure 2. This production can be binarized to produce a set of productions with parsing complexity 14 (Figure 3); among binarizations with this complexity the minimum fan-out is 6. However, an alternative binarization with fan-out 5 is also possible; among binarizations with this fan-out, the minimum parsing complexity is 15. This binarization (not pictured) first joins B_1 and B_2, then adds B_4, and finally adds B_3.

Given the incompatibility of optimizing time complexity and fan-out, which corresponds to space complexity, which should we prefer? In some situations, it may be desirable to find some trade-off between the two. It is important to note, however, that if optimization of space complexity is the sole objective, factorization is unnecessary, as one can never improve on the fan-out required by the original grammar nonterminals.

6 A Note on Generative Capacity

Rambow and Satta (1999) categorize the generative capacity of LCFRS grammars according to their rank and fan-out. In particular, they show that grammars can be arranged in a two-dimensional grid, with languages of rank r and fan-out f having greater generative capacity than both grammars of rank r and fan-out $f-1$ and grammars of rank $r-1$



$$
\begin{aligned}
\mathsf{nmod} &\rightarrow g_1 & g_1 &= \langle\, A \,\rangle \\
\mathsf{sbj} &\rightarrow g_2(\mathsf{nmod}, \mathsf{pp}) & g_2(\langle x_{1,1}\rangle, \langle x_{2,1}\rangle) &= \langle x_{1,1}\ hearing\ , x_{2,1}\rangle \\
\mathsf{root} &\rightarrow g_3(\mathsf{sbj}, \mathsf{vc}) & g_3(\langle x_{1,1}, x_{1,2}\rangle, \langle x_{2,1}, x_{2,2}\rangle) &= \langle x_{1,1}\ is\ x_{2,1} x_{1,2} x_{2,2}\rangle \\
\mathsf{vc} &\rightarrow g_4(\mathsf{tmp}) & g_4(\langle x_{1,1}\rangle) &= \langle\, scheduled\ , x_{1,1}\rangle \\
\mathsf{pp} &\rightarrow g_5(\mathsf{tmp}) & g_5(\langle x_{1,1}\rangle) &= \langle\, on\ x_{1,1}\rangle \\
\mathsf{nmod} &\rightarrow g_6 & g_6 &= \langle\, the \,\rangle \\
\mathsf{np} &\rightarrow g_7(\mathsf{nmod}) & g_7(\langle x_{1,1}\rangle) &= \langle x_{1,1}\ issue \,\rangle \\
\mathsf{tmp} &\rightarrow g_8 & g_8 &= \langle\, today \,\rangle
\end{aligned}
$$

Figure 4: A dependency tree with the LCFRS rules extracted for each word (Kuhlmann and Satta, 2009).

and fan-out f, with two exceptions: with fan-out 1, all ranks greater than one are equivalent (context-free languages), and with fan-out 2, rank 2 and rank 3 are equivalent.

This classification is somewhat unsatisfying because minor changes to a grammar can change both its rank and fan-out. In particular, through factorizing rules, it is always possible to decrease rank, potentially at the cost of increasing fan-out, until a binarized grammar of rank 2 is achieved.

Parsing complexity, as defined above, also provides a method to compare the generative capacity of LCFRS grammars. From Rambow and Satta's result that grammars of rank two and increasing fan-out provide an infinite hierarchy of increasing generative capacity, we see that parsing complexity also provides such an infinite hierarchy. Comparing grammars according to the parsing complexity amounts to specifying a normalized binarization for grammars of arbitrary rank and fan-out, and comparing the resulting binarized grammars. This allows us to arrange LCFRS grammars into total ordering over generative capacity, that is a one-dimensional hierarchy, rather than a two-dimensional grid. It also gives a way of categorizing generative capacity that is more closely tied to algorithmic complexity.

It is important to note, however, that parsing complexity as calculated by our algorithm remains a function of the grammar, rather than an intrinsic function of the language. One can produce arbitrarily complex grammars that generate the simple language a^*. Thus the parsing complexity of a grammar, like its rank and fan-out, can be said to categorize its *strong* generative capacity.

7 Experiments

A number of recent papers have examined dynamic programming algorithms for parsing non-projective dependency structures by exploring how well various categories of polynomially-parsable grammars cover the structures found in dependency treebanks for various languages (Kuhlmann and Nivre, 2006; Gómez-Rodríguez et al., 2009b).

Kuhlmann and Satta (2009) give an algorithm for extracting LCFRS rules from dependency structures. One rule is extracted for each word in the dependency tree. The rank of the rule is the number of children that the word has in the dependency tree, as shown by the example in Figure 4. The fan-out of the symbol corresponding to a word is the number of continuous intervals in the sentence formed by the word and its descendants in the tree. Projec-

complexity	arabic	czech	danish	dutch	german	port	swedish
20							1
18							1
16							1
15							1
13							1
12						2	3
11					1	1	1
10		2			6	16	3
9					7	4	1
8		4		7	129	65	10
7		3		12	89	30	18
6		178	11	362	1811	492	59
5	48	1132	93	411	1848	172	201
4	250	18269	1026	6678	18124	2643	1736
3	10942	265202	18306	39362	154948	41075	41245

Table 1: Number of productions with specified parsing complexity

tive trees yield LCFRS rules of fan-out one and parsing complexity three, while the fan-out and parsing complexity from non-projective trees are in principle unbounded.

Extracting LCFRS rules from treebanks allows us to study how many of the rules fall within certain constraints. Kuhlmann and Satta (2009) give an algorithm for binarizing LCRFS rules without increasing the rules' fan-out; however, this is not always possible, and the algorithm does not succeed even in some cases for which such a binarization is possible. Kuhlmann and Satta (2009) find that all but 0.02% of productions in the CoNLL 2006 training data, which includes various languages, can be binarized by their algorithm, but they do not give the fan-out or parsing complexity of the resulting rules. In related work, Gómez-Rodríguez et al. (2009b) define the class of *mildly ill-nested dependency structures* of varying gap degrees; gap degree is essentially fan-out minus one. For a given gap degree k, this class of grammars can be parsing in time $O(n^{3k+4})$ for lexicalized grammars. Gómez-Rodríguez et al. (2009b) study dependency treebanks for nine languages and find that all dependency structures meet the mildly ill-nested condition in the dependency treebanks for some gap degree. However, they do not report the maximum gap degree or parsing complexity.

We extracted LCFRS rules from dependency treebanks using the same procedure as Kuhlmann and Satta (2009), and applied the algorithm of Figure 1 directly to calculate their minimum parsing complexity. This allows us to characterize the parsing complexity of the rules found in the treebank without needing to define specific conditions on the rules, such as well-nestedness (Kuhlmann and Nivre, 2006) or mildly ill-nestedness, that may not be necessary for all efficiently parsable grammars. The numbers of rules of different complexities are shown in Table 1.

As found by previous studies, the vast majority of productions are context-free (projective trees, parsable in $O(n^3)$). Of non-projective rules, the vast majority can be parsed in $O(n^6)$, including the well-nested structures of gap degree one defined by Kuhlmann and Nivre (2006). The single most complex rule had parsing complexity of $O(n^{20})$, and was derived from a Swedish sentence which turns out to be so garbled as to be incomprehensible (taken from the high school essay portion of the Swedish treebank). It is interesting to note that, while the binarization algorithm is exponential in the worst case, it is practical for real data: analyzing all the rules extracted from the various treebanks takes only a few minutes. We did not find any cases in rules extracted from Treebank data of rules where minimizing parsing complexity is inconsistent with minimizing fan-

out, as is the case for the rule of Figure 2.

8 Conclusion

We give an algorithm for finding the optimum parsing complexity for an LCFRS among grammars obtained by binarization. We find that minimum parsing complexity is always achievable with rank 2, but is not always achievable with minimum fan-out. By applying the binarization algorithm to productions found in dependency treebanks, we can completely characterize the parsing complexity of the extracted LCFRS grammar.

Acknowledgments This work was funded by NSF grants IIS-0546554 and IIS-0910611. We are grateful to Joakim Nivre for assistance with the Swedish treebank.

References

Carlos Gómez-Rodríguez, Marco Kuhlmann, Giorgio Satta, and David Weir. 2009a. Optimal reduction of rule length in linear conext-free rewriting systems. In *Proceedings of the 2009 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-09)*, pages 539–547.

Carlos Gómez-Rodríguez, David Weir, and John Carroll. 2009b. Parsing mildly non-projective dependency structures. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL-09)*, pages 291–299.

A.K. Joshi and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin.

Marco Kuhlmann and Joakim Nivre. 2006. Mildly non-projective dependency structures. In *Proceedings of the International Conference on Computational Linguistics/Association for Computational Linguistics (COLING/ACL-06)*, pages 507–514.

Marco Kuhlmann and Giorgio Satta. 2009. Treebank grammar techniques for non-projective dependency parsing. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL-09)*, pages 478–486.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 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 (HLT/EMNLP)*.

I. Dan Melamed, Giorgio Satta, and Ben Wellington. 2004. Generalized multitext grammars. In *Proceedings of the 42nd Annual Conference of the Association for Computational Linguistics (ACL-04)*, Barcelona, Spain.

Owen Rambow and Giorgio Satta. 1999. Independent parallelism in finite copying parallel rewriting systems. *Theor. Comput. Sci.*, 223(1-2):87–120.

H. Seki, T. Matsumura, M. Fujii, and T. Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

K. Vijay-Shankar, D. L. Weir, and A. K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Conference of the Association for Computational Linguistics (ACL-87)*.

Benjamin Wellington, Sonjia Waxmonsky, and I. Dan Melamed. 2006. Empirical lower bounds on the complexity of translational equivalence. In *Proceedings of the International Conference on Computational Linguistics/Association for Computational Linguistics (COLING/ACL-06)*, pages 977–984, Sydney, Australia.