Grammar Factorization by Tree Decomposition

Daniel Gildea*
University of Rochester

We describe the application of the graph-theoretic property known as treewidth to the problem of finding efficient parsing algorithms. This method, similar to the junction tree algorithm used in graphical models for machine learning, allows automatic discovery of efficient algorithms such as the $O(n^4)$ algorithm for bilexical grammars of Eisner and Satta. We examine the complexity of applying this method to parsing algorithms for general Linear Context-Free Rewriting Systems. We show that any polynomial-time algorithm for this problem would imply an improved approximation algorithm for the well-studied treewidth problem on general graphs.

1. Introduction

In this article, we describe meta-algorithms for parsing: algorithms for finding the optimal parsing algorithm for a given grammar, with the constraint that rules in the grammar are considered independently of one another. In order to have a common representation for our algorithms to work with, we represent parsing algorithms as weighted deduction systems (Shieber, Schabes, and Pereira 1995; Goodman 1999; Nederhof 2003). Weighted deduction systems consist of axioms and rules for building **items** or partial results. Items are identified by square brackets, with their **weights** written to the left. Figure 1 shows a rule for deducing a new item when parsing a context free grammar (CFG) with the rule $S \rightarrow A\,B$. The item below the line, called the **consequent**, can be derived if the two items above the line, called the **antecedents**, have been derived. Items have types, corresponding to grammar nonterminals in this example, and **variables**, whose values range over positions in the string to be parsed. We restrict ourselves to items containing position variables directly as arguments; no other functions or operations are allowed to apply to variables. The consequent's weight is the product of the weights of the two antecedents and the rule weight w_0. Implicit in the notation is the fact that we take the maximum weight over all derivations of the same item. Thus, the weighted deduction system corresponds to the Viterbi or max-product algorithm for parsing. Applications of the same weighted deduction system with other semirings are also possible (Goodman 1999).

The computational complexity of parsing depends on the total number of instantiations of variables in the system's deduction rules. If the total number of instantiations is M, parsing is $O(M)$ if there are no cyclic dependencies among instantiations, or, equivalently, if all instantiations can be sorted topologically. In most parsing algorithms,

* Computer Science Department, University of Rochester, Rochester NY 14627.
E-mail: gildea@cs.rochester.edu.

Submission received: 28 June 2010; revised submission received: 20 September 2010; accepted for publication: 21 October 2010.

$$\frac{\begin{array}{rl} w_1: & [A, x_0, x_1] \\ w_2: & [B, x_1, x_2] \end{array}}{w_0 w_1 w_2: \quad [S, x_0, x_2]}$$

Figure 1
CFG parsing in weighted deduction notation.

variables range over positions in the input string. In order to determine complexity in the length n of the input string, it is sufficient to count the number of unique position variables in each rule. If all rules have at most k position variables, $M = O(n^k)$, and parsing takes time $O(n^k)$ in the length of the input string. In the remainder of this article, we will explore methods for minimizing k, the largest number of position variables in any rule, among equivalent deduction systems. These methods directly minimize the parsing complexity of the resulting deduction system. Although we will assume no cyclic dependencies among rule instantiations for the majority of the article, we will discuss the cyclic case in Section 2.2.

It is often possible to improve the computational complexity of a deduction rule by decomposing the computation into two or more new rules, each having a smaller number of variables than the original rule. We refer to this process as **factorization**. One straightforward example of rule factorization is the binarization of a CFG, as shown in Figure 2. Given a deduction rule for a CFG rule with r nonterminals on the righthand side, and a total of $r + 1$ variables, an equivalent set of rules can be produced, each with three variables, storing intermediate results that indicate that a substring of the original rule's righthand side has been recognized. This type of rule factorization produces an $O(n^3)$ parser for any input CFG.

Another well-known instance of rule factorization is the hook trick of Eisner and Satta (1999), which reduces the complexity of parsing for bilexicalized CFGs from $O(n^5)$ to $O(n^4)$. The basic rule for bilexicalized parsing combines two CFG constituents marked with lexical heads as shown in Figure 3a. Here items with type C indicate constituents, with $[C, x_0, h, x_1]$ indicating a constituent extending from position x_0 to position x_1, headed by the word at position h. The item $[D, m \rightarrow h]$ is used to indicate the weight assigned by the grammar to a bilexical dependency headed by the word at

a)

$$\frac{\begin{array}{rl} w_1: & [A, x_0, x_1] \\ w_2: & [B, x_1, x_2] \\ w_3: & [C, x_2, x_3] \\ w_4: & [D, x_3, x_4] \end{array}}{w_0 w_1 w_2 w_3 w_4: \quad [S, x_0, x_4]}$$

b)

$$\frac{\begin{array}{rl} w_1: & [A, x_0, x_1] \\ w_2: & [B, x_1, x_2] \end{array}}{w_1 w_2: \quad [X, x_0, x_2]} \qquad \frac{\begin{array}{rl} w_5: & [X, x_0, x_2] \\ w_3: & [C, x_2, x_3] \end{array}}{w_3 w_5: \quad [Y, x_0, x_3]} \qquad \frac{\begin{array}{rl} w_6: & [Y, x_0, x_3] \\ w_3: & [D, x_3, x_4] \end{array}}{w_0 w_3 w_6: \quad [S, x_0, x_3]}$$

Figure 2
Binarization of the CFG rule $S \rightarrow A\,B\,C\,D$ as rule factorization: The deduction rule above can be factored into the three equivalent rule below.

a)

$$\begin{array}{rl} w_\ell: & [D, m \rightarrow h] \\ w_1: & [C, x_0, h, x_1] \\ w_2: & [C, x_1, m, x_2] \\ \hline w_\ell w_1 w_2: & [C, x_0, h, x_2] \end{array}$$

b)

$$\begin{array}{rl} w_\ell: & [D, m \rightarrow h] \\ w_2: & [C, x_1, m, x_2] \\ \hline w_\ell w_2: & [H, h, x_1, x_2] \end{array} \qquad \begin{array}{rl} w_h: & [H, h, x_1, x_2] \\ w_1: & [C, x_0, h, x_1] \\ \hline w_h w_1: & [C, x_0, h, x_2] \end{array}$$

Figure 3
Rule factorization for bilexicalized parsing.

position h with the word at position m as a modifier. The deduction rule is broken into two steps, one which includes the weight for the bilexical grammar rule, and another which identifies the boundaries of the new constituent, as shown in Figure 3b. The hook trick has also been applied to Tree Adjoining Grammar (TAG; Eisner and Satta 2000), and has been generalized to improve the complexity of machine translation decoding under synchronous context-free grammars (SCFGs) with an n-gram language model (Huang, Zhang, and Gildea 2005).

Rule factorization has also been studied in the context of parsing for SCFGs. Unlike monolingual CFGs, SCFGs cannot always be binarized; depending on the permutation between nonterminals in the two languages, it may or may not be possible to reduce the rank, or number of nonterminals on the righthand side, of a rule. Algorithms for finding the optimal rank reduction of a specific rule are given by Zhang and Gildea (2007). The complexity of synchronous parsing for a rule of rank r is $O(n^{2r+2})$, so reducing rank improves parsing complexity.

Rule factorization has also been applied to Linear Context-Free Rewriting Systems (LCFRS), which generalize CFG, TAG, and SCFG to define a rewriting system where nonterminals may have arbitrary **fan-out**, which indicates the number of continuous spans that a nonterminal accounts for in the string (Vijay-Shankar, Weir, and Joshi 1987). Recent work has examined the problem of factorization of LCFRS rules in order to reduce rank without increasing grammar fan-out (Gómez-Rodríguez et al. 2009), as well as factorization with the goal of directly minimizing the parsing complexity of the new grammar (Gildea 2010).

We define factorization as a process which applies to rules of the input grammar independently. Individual rules are replaced with an equivalent set of new rules, which must derive the same set of consequent items as the original rule given the same antecedent items. While new intermediate items of distinct types may be produced, the set of items and weights derived by the original weighted deduction system is unchanged. This definition of factorization is broad enough to include all of the previous examples, but does not include, for example, the fold/unfold operation applied to grammars by Johnson (2007) and Eisner and Blatz (2007). Rule factorization corresponds to the unfold operation of fold/unfold.

If we allow unrestricted transformations of the input deduction system, finding the most efficient equivalent system is undecidable; this follows from the fact that it is undecidable whether a CFG generates the set of all strings (Bar-Hillel, Perles, and Shamir 1961), and would therefore be recognizable in constant time. Whereas the fold/unfold operation of Johnson (2007) and Eisner and Blatz (2007) specifies a narrower class of

grammar transformations, no general algorithms are known for identifying an optimal series of transformations in this setting. Considering input rules independently allows us to provide algorithms for optimal factorization.

In this article, we wish to provide a general framework for factorization of deductive parsing systems in order to minimize computational complexity. We show how to apply the graph-theoretic property of treewidth to the factorization problem, and examine the question of whether efficient algorithms exist for optimizing the parsing complexity of general parsing systems in this framework. In particular, we show that the existence of a polynomial time algorithm for optimizing the parsing complexity of general LCFRS rules would imply an improved approximation algorithm for the well-studied problem of treewidth of general graphs.

2. Treewidth and Rule Factorization

In this section, we introduce the graph-theoretic property known as treewidth, and show how it can be applied to rule factorization.

A **tree decomposition** of a graph $G = (V, E)$ is a type of tree having a subset of G's vertices at each node. We define the nodes of this tree T to be the set I, and its edges to be the set F. The subset of V associated with node i of T is denoted by X_i. A tree decomposition is therefore defined as a pair $(\{X_i \mid i \in I\}, T = (I, F))$ where each $X_i, i \in I$ is a subset of V, and tree T has the following properties:

- *Vertex cover:* The nodes of the tree T cover all the vertices of G: $\bigcup_{i \in I} X_i = V$.
- *Edge cover:* Each edge in G is included in some node of T. That is, for all edges $(u, v) \in E$, there exists an $i \in I$ with $u, v \in X_i$.
- *Running intersection:* The nodes of T containing a given vertex of G form a connected subtree. Mathematically, for all $i, j, k \in I$, if j is on the (unique) path from i to k in T, then $X_i \bigcap X_k \subseteq X_j$.

The **treewidth** of a tree decomposition $(\{X_i\}, T)$ is $\max_i |X_i| - 1$. The treewidth of a graph is the minimum treewidth over all tree decompositions:

$$\mathbf{tw}(G) = \min_{(\{X_i\}, T) \in TD(G)} \max_i |X_i| - 1$$

where $TD(G)$ is the set of valid tree decompositions of G. We refer to a tree decomposition achieving the minimum possible treewidth as being optimal.

In general, more densely interconnected graphs have higher treewidth. Any tree has treewidth $= 1$; a graph consisting of one large cycle has treewidth $= 2$, and a fully connected graph of n vertices has treewidth $= n - 1$. Low treewidth indicates some tree-like structure in the graph, as shown by the example with treewidth $= 2$ in Figure 4. As an example of the running intersection property, note that the vertex N appears in three adjacent nodes of the tree decomposition. Finding the treewidth of a graph is an NP-complete problem (Arnborg, Corneil, and Proskurowski 1987). However, given a graph of n vertices and treewidth k, a simple algorithm finds the optimal tree decomposition in time $O(n^{k+2})$ (Arnborg, Corneil, and Proskurowski 1987), and a variety of approximation algorithms and heuristics are known for the treewidth problem (Bodlaender et al. 1995; Amir 2001; Feige, Hajiaghayi, and Lee 2005). Furthermore, for fixed k, optimal tree decompositions can be computed in linear time (Bodlaender 1996).

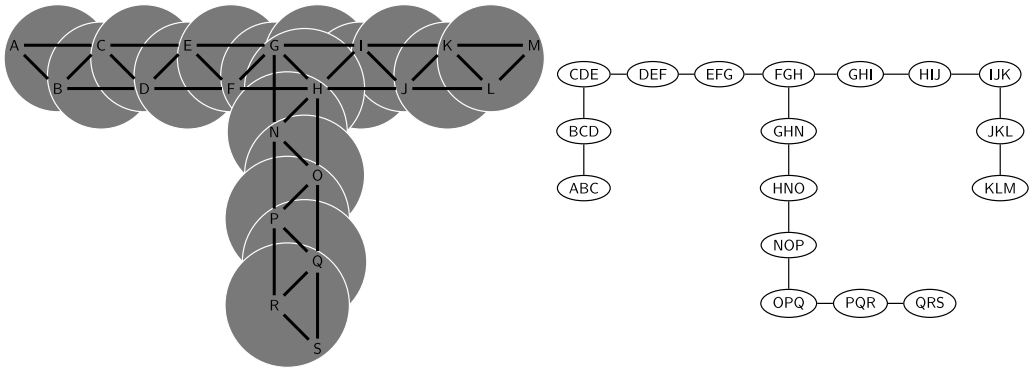

Figure 4
A tree decomposition of a graph is a set of overlapping clusters of the graph's vertices, arranged in a tree. This example has treewidth = 2.

We can factorize a deduction rule by representing the rule as a graph, which we call a **dependency graph**, and searching for tree decompositions of this graph. For a rule r having n variables $V = \{v_i \mid i \in \{1, \ldots, n\}\}$, m antecedent items A_i, $i \in \{1, \ldots, m\}$, and consequent C, let $V(A_i) \subset V$ be the variables appearing in antecedent A_i, and $V(C)$ be the variables appearing in the consequent. The dependency graph representation of the rule is $G_r = (V, E = \bigcup_{S:A_1,\ldots,A_m,C}\{(v_i, v_j) \mid v_i, v_j \in V(S)\})$. That is, we have a vertex for each variable in the rule, and connect any two vertices that appear together in the same antecedent, or that appear together in the consequent.

The dependency graph representation allows us to prove the following result concerning parsing complexity:

Theorem 1
Given a deduction rule r for parsing where the input string is referenced only through position variables appearing as arguments of antecedent and consequent items, the optimal complexity of any factorization of rule r is $O(n^{\mathbf{tw}(G_r)+1})$, where G_r is the dependency graph derived from r.

Proof
One consequence of the definition of a tree decomposition is that, for any clique appearing in the original graph G_r, there must exist a node in the tree decomposition T which contains all the vertices in the clique. We use this fact to show that there is a one-to-one correspondence between tree decompositions of a rule's dependency graph G_r and factorizations of the rule.

First, we need to show that any tree decomposition of G_r can be used as a factorization of the original deduction rule. By our earlier definition, a factorization must derive the same set of consequent items from a given set of antecedent items as the original rule. Because G_r includes a clique connecting all variables in the consequent C, the tree decomposition T must have a node X_c such that $V(C) \subseteq X_c$. We consider this node to be the root of T. The original deduction rule can be factorized into a new set of rules, one for each node in T. For node X_c, the factorized rule has C as a consequent, and all other nodes X_i have a new partial result as a consequent, consisting of the variables $X_i \cap X_j$, where X_j is X_i's neighbor on the path to the root node X_c. We must guarantee that the factorized rule set yields the same result as the original rule, namely, the semiring sum

over all variable values of the semiring product of the antecedents' weights. The tree structure of T corresponds to a factorization of this semiring expression. For example, if we represent the CFG rule of Figure 2a with the generalized semiring expression:

$$\bigoplus_{x_1 x_2 x_3} A(x_0, x_1) \otimes B(x_1, x_2) \otimes C(x_2, x_3) \otimes D(x_3, x_4)$$

the factorization of this expression corresponding to the binarized rule is

$$\bigoplus_{x_3} \left(\bigoplus_{x_2} \left(\bigoplus_{x_1} A(x_0, x_1) \otimes B(x_1, x_2) \right) \otimes C(x_2, x_3) \right) \otimes D(x_3, x_4)$$

where semiring operations $\oplus$ and $\otimes$ have been interchanged as allowed by the dependency graph for this rule.

Because each antecedent A_i is represented by a clique in the graph G_r, the tree decomposition T must contain at least one node which includes all variables $V(A_i)$. We can choose one such node and multiply in the weight of A_i, given the values of variables $V(A_i)$, at this step of the expression. The running intersection property of the tree decomposition guarantees that each variable has a consistent value at each point where it is referenced in the factorization.

The same properties guarantee that any valid rule factorization corresponds to a tree decomposition of the graph G_r. We consider the tree decomposition with a set X_i for each new rule r_i, consisting of all variables used in r_i, and with tree edges T defined by the producer/consumer relation over intermediate results in the rule factorization. Each antecedent of the original rule must appear in some new rule in the factorization, as must the consequent of the original rule. Therefore, all edges in the original rule's dependency graph G_r appear in some tree node X_i. Any variable that appears in two rules in the factorization must appear in all intermediate rules in order to ensure that the variable has a consistent value in all rules that reference it. This guarantees the running intersection property of the tree decomposition $(\{X_i\}, T)$. Thus any rule factorization, when viewed as a tree of sets of variables, has the properties that make it a valid tree decomposition of G_r.

The theorem follows as a consequence of the one-to-one correspondence between rule factorizations and tree decompositions. ■

2.1 Computational Complexity

Factorization produces, for each input rule having m antecedents, at most $m - 1$ new rules, each containing at most the same number of nonterminals and the same number of variables as the input rule. Hence, the size of the new factorized grammar is $O(|G|^2)$, and we avoid any possibility of an exponential increase in grammar size. Tighter bounds can be achieved for specific classes of input grammars.

The computational complexity of optimal factorization with tree decomposition is exponential in the size of the input rules. However, optimal factorization is generally feasible whenever parsing with the unfactorized grammar is feasible. This is because, for an input rule with ℓ variables, parsing is $O(n^\ell)$ in the sentence length n. The treewidth of this rule is at most $\ell - 1$, and can be computed in time $O(\ell^{\ell+1})$; generally we expect n to be greater than ℓ. One may also wish to accept only rules having treewidth k and disregard the remainder, for example, when factorizing rules automatically

extracted from word-aligned bitext (Wellington, Waxmonsky, and Melamed 2006; Huang et al. 2009) or from dependency treebanks (Kuhlmann and Nivre 2006; Gildea 2010). In this setting, the rules having treewidth k can be identified in time $O(\ell^{k+2})$ using the simple algorithm of Arnborg, Corneil, and Proskurowski (1987), (where again ℓ is the number of variables in the input rules), or in time $O(\ell)$ using the algorithm of Bodlaender (1996).

2.2 Cyclic Dependencies

Although this article primarily addresses the case where there are no cyclic dependencies between rule instantiations, we note here that our techniques carry over to the cyclic case under certain conditions. If there are cycles in the rule dependencies, but the semiring meets Knuth's (1977) definition of a **superior** function, parsing takes time $O(M \log M)$, where M is the number of rule instantiations, and the extra $\log M$ term accounts for maintaining an agenda as a priority queue (Nederhof 2003). Cycles in the rule dependencies may arise, for example, from chains of unary productions in a CFG; the properties of superior functions guarantee that unbounded chains need not be considered. The max-product semiring used in Viterbi parsing has this property, assuming that all rule weights are less than one, whereas for exact computation with the sum-product semiring, unbounded chains must be considered. As in the acyclic case, $M = O(n^k)$ for parsing problems where rules have at most k variables. Under the assumption of superior functions, parsing takes time $O(n^k k \log n)$ with Knuth's algorithm. In this setting, as in the acyclic case, minimizing k with tree decomposition minimizes parsing complexity.

2.3 Related Applications of Treewidth

The technique of using treewidth to minimize complexity has been applied to constraint satisfaction (Dechter and Pearl 1989), graphical models in machine learning (Jensen, Lauritzen, and Olesen 1990; Shafer and Shenoy 1990), and query optimization for databases (Chekuri and Rajaraman 1997). Our formulation of parsing is most closely related to logic programming; in this area treewidth has been applied to limit complexity in settings where either the deduction rules or the input database of ground facts have fixed treewidth (Flum, Frick, and Grohe 2002). Whereas Flum, Frick, and Grohe (2002) apply treewidth to nonrecursive datalog programs, our parsing programs have unbounded recursion, as the depth of the parse tree is not fixed in advance. Our results for parsing can be seen as a consequence of the fact that, even in the case of unbounded recursion, the complexity of (unweighted) datalog programs is linear in the number of possible rule instantiations (McAllester 2002).

3. Examples of Treewidth for Parsing

In this section, we show how a few well-known parsing algorithms can be derived automatically by finding the optimal tree decomposition of a dependency graph.

To aid in visualization of the graphical representation of deduction rules, we use a factor graph representation based on that of Kschischang, Frey, and Loeliger (2001) for Markov Random Fields. Our graphs have three types of nodes: variables, antecedents, and consequents. Each antecedent node is connected to the variables it contains, and represents the antecedent's weight as a function of those variables. Antecedent nodes are analogous to the factor nodes of Kschischang, Frey, and Loeliger (2001), and

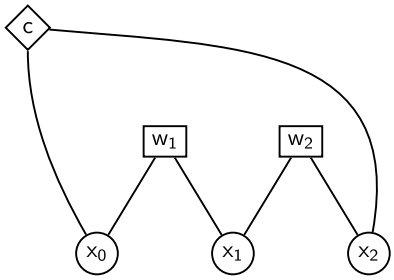

Figure 5
Factor graph for the binary CFG deduction rule of Figure 1.

consequent nodes are a new feature of this representation. We can think of consequents as factors with weight = 1; they do not affect the weights computed, but serve to guarantee that the consequent of the original rule can be found in one node of the tree decomposition. We refer to both antecedent and consequent nodes as factor nodes. Replacing each factor node with a clique over its neighbor variables yields the dependency graph G_r defined earlier. We represent variables with circles, antecedents with squares labeled with the antecedent's weight, and consequents with diamonds labeled c. An example factor graph for the simple CFG rule of Figure 1 is shown in Figure 5.

3.1 CFG Binarization

Figure 6a shows the factor graph derived from the monolingual CFG rule with four children in Figure 2a. The dependency graph obtained by replacing each factor with a clique of size 2 (a single edge) is a graph with one large cycle, shown in Figure 6b. Finding the optimal tree decomposition yields a tree with nodes of size 3, $\{x_0, x_i, x_{i+1}\}$ for each i, shown in Figure 6c. Each node in this tree decomposition corresponds to one of the factored deduction rules in Figure 2b. Thus, the tree decomposition shows us how

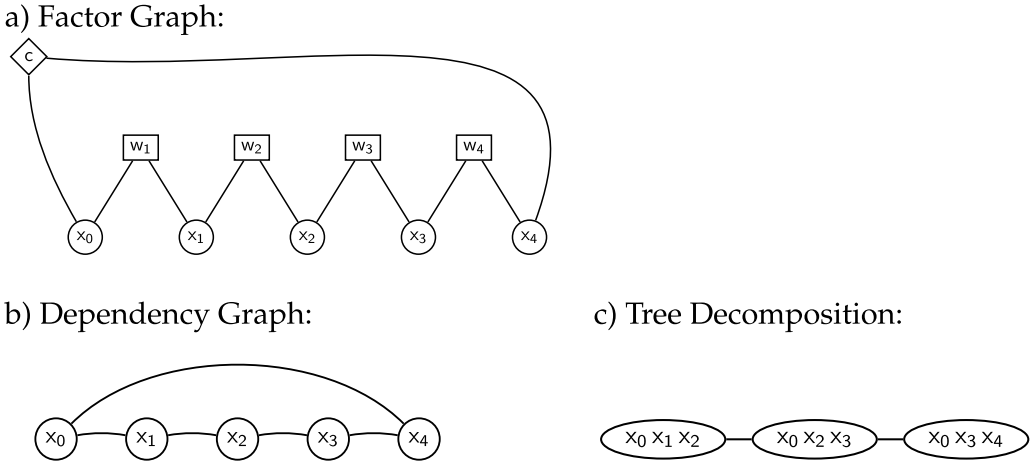

Figure 6
Treewidth applied to CFG binarization.

to parse in time $O(n^3)$; finding the tree decomposition of a long CFG rule is essentially equivalent to converting to Chomsky Normal Form.

3.2 The Hook Trick

The deduction rule for bilexicalized parsing shown in Figure 3a translates into the factor graph shown in Figure 7a. Factor nodes are created for the two existing constituents from the chart, with the first extending from position x_0 in the string to x_1, and the second from x_1 to x_2. Both factor nodes are connected not only to the start and end points, but also to the constituent's head word, h for the first constituent and m for the second (we show the construction of a left-headed constituent in the figure). An additional factor is connected only to h and m to represent the bilexicalized rule weight, expressed as a function of h and m, which is multiplied with the weight of the two existing constituents to derive the weight of the new constituent. The new constituent is represented by a consequent node at the top of the graph—the variables that will be relevant for its further combination with other constituents are its end points x_0 and x_2 and its head word h.

Placing an edge between each pair of variable nodes that share a factor, we get Figure 7b. If we compute the optimal tree decomposition for this graph, shown in Figure 7c, each of the two nodes corresponds to one of the factored rules in Figure 3b. The largest node of the tree decomposition has four variables, giving the $O(n^4)$ algorithm of Eisner and Satta (1999).

3.3 SCFG Parsing Strategies

SCFGs generalize CFGs to generate two strings with isomorphic hierarchical structure simultaneously, and have become widely used as statistical models of machine translation (Galley et al. 2004; Chiang 2007). We write SCFG rules as productions with one

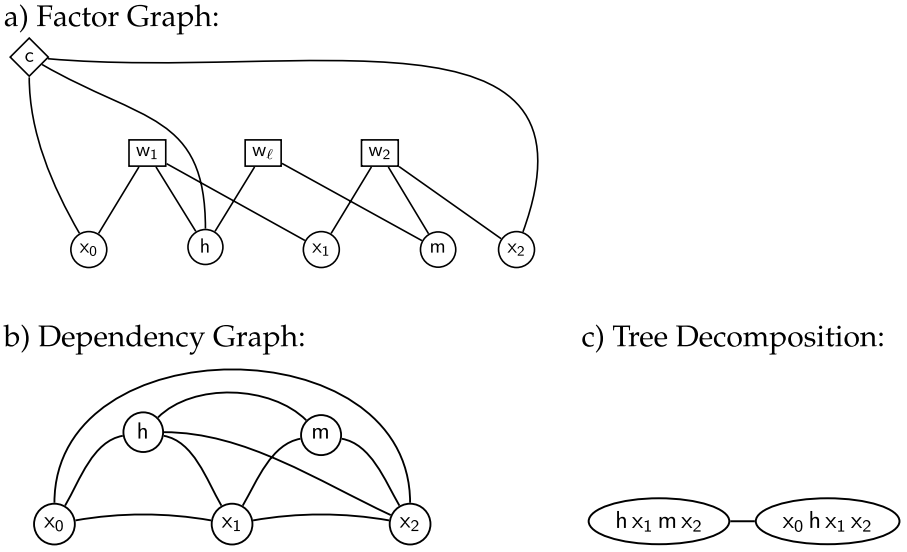

Figure 7
Treewidth applied to bilexicalized parsing.

lefthand side nonterminal and two righthand side strings. Nonterminals in the two strings are linked with superscript indices; symbols with the same index must be further rewritten synchronously. For example,

$$X \rightarrow A^{(1)}\, B^{(2)}\, C^{(3)}\, D^{(4)},\ A^{(1)}\, B^{(2)}\, C^{(3)}\, D^{(4)} \tag{1}$$

is a rule with four children and no reordering, whereas

$$X \rightarrow A^{(1)}\, B^{(2)}\, C^{(3)}\, D^{(4)},\ B^{(2)}\, D^{(4)}\, A^{(1)}\, C^{(3)} \tag{2}$$

expresses a more complex reordering. In general, we can take indices in the first righthand-side string to be consecutive, and associate a permutation π with the second string. If we use X_i for $0 \leq i \leq n$ as a set of variables over nonterminal symbols (for example, X_1 and X_2 may both stand for nonterminal A), we can write rules in the general form:

$$X_0 \rightarrow X_1^{(1)} \cdots X_n^{(n)},\ X_{\pi(1)}^{(\pi(1))} \cdots X_{\pi(n)}^{(\pi(n))}$$

Unlike monolingual CFGs, SCFGs cannot always be binarized. In fact, the languages of string pairs generated by a synchronous grammar can be arranged in an infinite hierarchy, with each rank ≥ 4 producing languages not possible with grammars restricted to smaller rules (Aho and Ullman 1972). For any grammar with maximum rank r, converting each rule into a single deduction rule yields an $O(n^{2r+2})$ parsing algorithm, because there are $r+1$ boundary variables in each language. More efficient parsing algorithms are often possible for specific permutations, and, by Theorem 1, the best algorithm for a permutation can be found by computing the minimum-treewidth tree decomposition of the graph derived from the SCFG deduction rule for a specific permutation. For example, for the non-binarizable rule of Equation (2), the resulting factor graph is shown in Figure 8a, where variables $x_0, \ldots, x_4$ indicate position variables in one language of the synchronous grammar, and $y_0, \ldots, y_4$ are positions in the other language. The optimal tree decomposition for this rule is shown in Figure 8c. For this permutation, the optimal parsing algorithm takes time $O(n^8)$, because the largest node in the tree decomposition of Figure 8c includes eight position variables. This result is intermediate between the $O(n^6)$ for binarizable SCFGs, also known as Inversion Transduction Grammars (Wu 1997), and the $O(n^{10})$ that we would achieve by recognizing the rule in a single deduction step.

Gildea and Štefankovič (2007) use a combinatorial argument to show that as the number of nonterminals r in an SCFG rule grows, the parsing complexity grows as $\Omega(n^{cr})$ for some constant c. In other words, some very difficult permutations exist of all lengths.

It is interesting to note that although applying the tree decomposition technique to long CFG rules results in a deduction system equivalent to a binarized CFG, the individual deduction steps in the best parsing strategy for an SCFG rule do not in general correspond to SCFG rules. This is because the intermediate results may include more than one span in each language. These intermediate deduction steps do, however, correspond to LCFRS rules. We now turn to examine LCFRS in more detail.

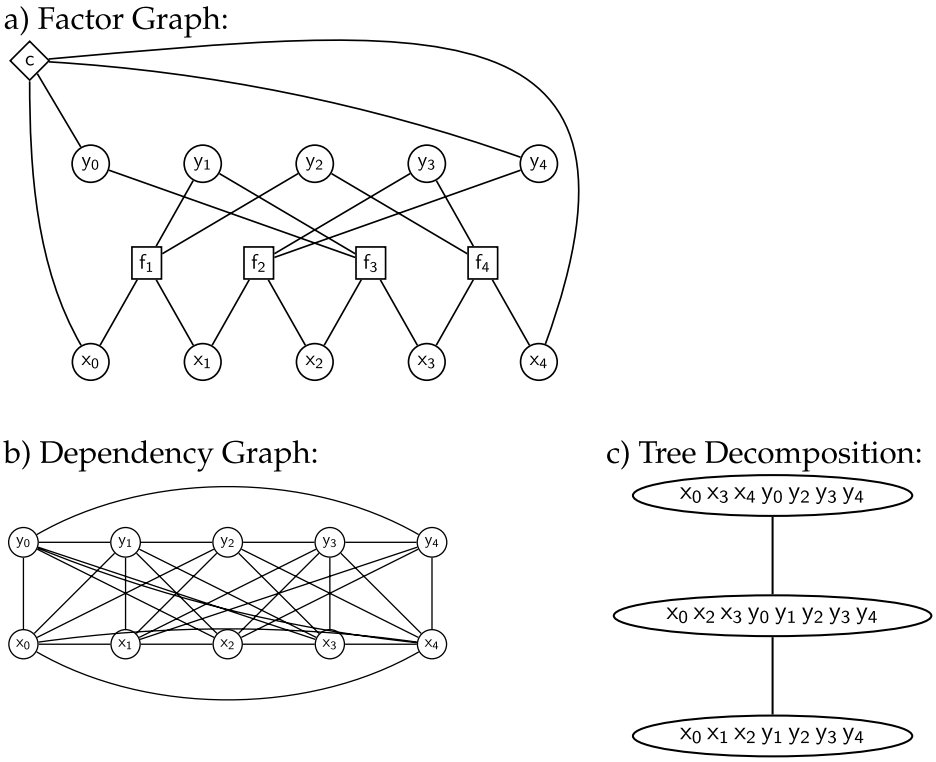

Figure 8
Treewidth applied to the SCFG rule of Equation (2).

4. LCFRS Parsing Strategies

LCFRS provides a generalization of a number of widely used formalisms in natural language processing, including CFG, TAG, SCFG, and synchronous TAG. LCFRS has also been used to model non-projective dependency grammars, and the LCFRS rules extracted from dependency treebanks can be quite complex (Kuhlmann and Satta 2009), making factorization important. Similarly, LCFRS can model translation relations beyond the power of SCFG (Melamed, Satta, and Wellington 2004), and grammars extracted from word-aligned bilingual corpora can also be quite complex (Wellington, Waxmonsky, and Melamed 2006). An algorithm for factorization of LCFRS rules is presented by Gildea (2010), exploiting specific properties of LCFRS. The tree decomposition method achieves the same results without requiring analysis specific to LCFRS. In this section, we examine the complexity of rule factorization for general LCFRS grammars.

The problem of finding the optimal factorization of an arbitrary deduction rule is NP-complete. This follows from the NP-completeness of treewidth using the following construction: Given a graph, create a deduction rule with a variable for each vertex in the graph and an antecedent for each edge, containing the two variables associated with the edge's endpoints. The graphs produced by LCFRS grammar rules, however, have certain properties which may make more efficient factorization algorithms possible. We first define LCFRS precisely before examining the properties of these graphs.

An LCFRS is defined as a tuple $G = (V_T, V_N, P, S)$, 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 tells how many continuous spans B covers. Productions $p \in P$ take the form:

$$p : A \rightarrow g(B_1, B_2, \ldots, B_r) \tag{3}$$

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

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

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** and **non-erasing**, which means that if we write

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

the tuple of strings $\langle t_1, \ldots, t_{\varphi(A)} \rangle$ on the righthand side contains each variable $s_{i,j}$ from the lefthand side exactly once, and may also contain terminals from V_T. The process of generating a string from an LCFRS grammar can be thought of as first choosing, top-down, a production to expand each nonterminal, and then, bottom–up, applying the functions associated with each production to build the string. As an example, the CFG

$$S \rightarrow A\,B$$
$$A \rightarrow a$$
$$B \rightarrow b$$

corresponds to the following grammar in LCFRS notation:

$$S \rightarrow g_S(A, B) \qquad g_S(\langle s_A \rangle, \langle s_B \rangle) = \langle s_A s_B \rangle$$
$$A \rightarrow g_A() \qquad g_A() = \langle a \rangle$$
$$B \rightarrow g_B() \qquad g_B() = \langle b \rangle$$

Here, all nonterminals have fan-out = 1, reflected in the fact that all tuples defining the productions' functions contain just one string. As CFG is equivalent to LCFRS with fan-out = 1, SCFG and TAG can be represented as LCFRS with fan-out = 2. Higher values of fan-out allow strictly more powerful grammars (Rambow and Satta 1999). Polynomial-time parsing is possible for any fixed LCFRS grammar, but the degree of the polynomial depends on the grammar. Parsing general LCFRS grammars, where the grammar is considered part of the input, is NP-complete (Satta 1992).

4.1 Graphs Derived from LCFRS Rules

Given an LCFRS rule as defined previously, a weighted deduction rule for a bottom–up parser can be derived by creating an antecedent for each righthand nonterminal, a consequent for the lefthand side, and variables for all the boundaries of the nonterminals in the rule. A nonterminal of fan-out f has $2f$ boundaries. Each boundary

variable will occur exactly twice in the deduction rule: either in two antecedents, if two nonterminals on the rule's righthand side are adjacent, or once in an antecedent and once in the consequent, if the variable indicates a boundary of any segment of the rule's lefthand side.

Converting such deduction rules into dependency graphs, we see that the cliques of the dependency graph may be arbitrarily large, due to the unbounded fan-out of LCFRS nonterminals. However, each vertex appears in only two cliques, because each boundary variable in the rule is shared by exactly two nonterminals. In the remainder of this section, we consider whether the problem of finding the optimal tree decomposition of this restricted set of graphs is also NP-complete, or whether efficient algorithms may be possible in the LCFRS setting.

4.2 Approximation of Treewidth for General Graphs

We will show that an efficient algorithm for finding the factorization of an arbitrary LCFRS production that optimizes parsing complexity would imply the existence of an algorithm for treewidth that returns a result within a factor of $4\Delta(G)$ of the optimum, where $\Delta(G)$ is the maximum degree of the input graph. Although such an approximation algorithm may be possible, it would require progress in fundamental problems in graph theory.

Consider an arbitrary graph $G = (V, E)$, and define k to be its treewidth, $k = \mathbf{tw}(G)$. We wish to construct a new graph $G' = (V', E')$ from G in such a way that $\mathbf{tw}(G') = \mathbf{tw}(G)$ and every vertex in G' has even degree. This can be accomplished by doubling the graph's edges in the manner shown in Figure 9. To double the edges, for every edge $e = (u, v)$ in E, we add a new vertex $\hat{e}$ to G' and add edges $(u, \hat{e})$ and $(v, \hat{e})$ to G'. We also include every edge in the original graph G in G'. Now, every vertex v in G' has degree = 2, if it is a newly created vertex, or twice the degree of v in G otherwise, and therefore

$$\Delta(G') = 2\Delta(G) \tag{4}$$

We now show that $\mathbf{tw}(G') = \mathbf{tw}(G)$, under the assumption that $\mathbf{tw}(G) \geq 3$. Any tree decomposition of G can be adapted to a tree decomposition of G' by adding a node containing $\{u, v, \hat{e}\}$ for each edge e in the original graph, as shown in Figure 10. The new node can be attached to a node containing u and v; because u and v are connected by an edge in G, such a node must exist in G's tree decomposition. The vertex $\hat{e}$ will not occur anywhere else in the tree decomposition, and the occurrences of u and v still form a connected subtree. For each edge $e = (u, v)$ in G', the tree decomposition must have a node containing u and v; this is the case because, if e is an original edge from G, there is already a node in the tree decomposition containing u and v, whereas if e is an edge to a newly added vertex in G', one of the newly added nodes in the tree decomposition

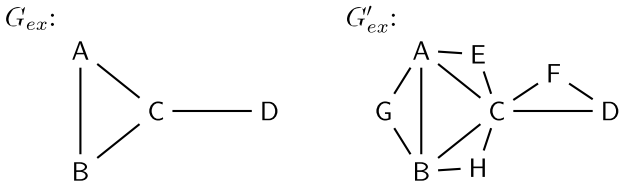

Figure 9
An example graph G_{ex} and the result G'_{ex} of doubling G_{ex}'s edges.

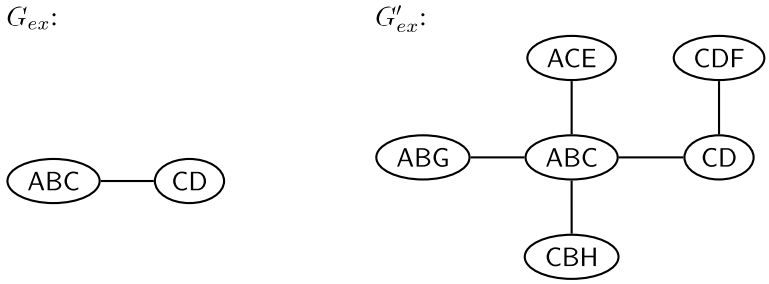

Figure 10
Tree decompositions of G_{ex} and G'_{ex}.

will contain its endpoints. We constructed the new tree decomposition by adding nodes of size 3. Therefore, as long as the treewidth of G was at least 3, $\mathbf{tw}(G') \leq \mathbf{tw}(G)$. In the other direction, because G is a subgraph of G', any tree decomposition of G' forms a valid tree decomposition of G after removing the vertices in $G' - G$, and hence $\mathbf{tw}(G') \geq \mathbf{tw}(G)$. Therefore,

$$\mathbf{tw}(G') = \mathbf{tw}(G) \tag{5}$$

Because every vertex in G' has even degree, G' has an Eulerian tour, that is, a path visiting every edge exactly once, beginning and ending at the same vertex. Let $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ be the sequence of vertices along such a tour, with $\pi_1 = \pi_n$. Note that the sequence π contains repeated elements. Let $\mu_i, i \in \{1, \ldots, n\}$ indicate how many times we have visited π_i on the ith step of the tour: $\mu_i = |\{j \mid \pi_j = \pi_i, j \in \{1, \ldots, i\}\}|$. We now construct an LCFRS production P with $|V'|$ righthand side nonterminals from the Eulerian tour:

$$P : X \rightarrow g(B_1, \ldots, B_{|V'|})$$
$$g(\langle s_{1,1}, \ldots, s_{1,\phi(B_1)} \rangle, \ldots, \langle s_{|V'|,1}, \ldots, s_{|V'|,\phi(B_{|V'|})} \rangle) = \langle s_{\pi_1,\mu_1} \cdots s_{\pi_n,\mu_n} \rangle$$

The fan-out $\phi(B_i)$ of each nonterminal B_i is the number of times vertex i is visited on the Eulerian tour. The fan-out of the lefthand side nonterminal X is one, and the lefthand side is constructed by concatenating the spans of each nonterminal in the order specified by the Eulerian tour.

For the example graph in Figure 9, one valid tour is

$$\pi_{ex} = \langle A, B, C, D, F, C, E, A, G, B, H, C, A \rangle$$

This tour results in the following LCFRS production:

$$P_{ex} : X \rightarrow g_{ex}(A, B, C, D, E, F, G, H)$$
$$g_{ex}(\langle s_{A,1}, s_{A,2}, s_{A,3} \rangle, \langle s_{B,1}, s_{B,2} \rangle, \langle s_{C,1}, s_{C,2}, s_{C,3} \rangle, \langle s_{D,1} \rangle, \langle s_{E,1} \rangle, \langle s_{F,1} \rangle, \langle s_{G,1} \rangle, \langle s_{H,1} \rangle) =$$
$$\langle s_{A,1} s_{B,1} s_{C,1} s_{D,1} s_{F,1} s_{C,2} s_{E,1} s_{A,2} s_{G,1} s_{B,2} s_{H,1} s_{C,3} s_{A,3} \rangle$$

We now construct dependency graph G'' from the LCFRS production P by applying the technique of Section 2. G'' has $n + 1$ vertices, corresponding to the beginning and

end points of the nonterminals in P. The edges in G'' are formed by adding a clique for each nonterminal in P connecting all its beginning and end points, that is, $\binom{2f}{2}$ edges for a nonterminal of fan-out f. We must include a clique for X, the lefthand side of the production. However, because the righthand side of the production begins and ends with the same nonterminal, the vertices for the beginning and end points of X are already connected, so the lefthand side does not affect the graph structure for the entire production. By Theorem 1, the optimal parsing complexity of P is $\mathbf{tw}(G'') + 1$.

The graphs G' and G'' are related in the following manner: Every edge in G' corresponds to a vertex in G'', and every vertex in G' corresponds to a clique in G''. We can identify vertices in G'' with unordered pairs of vertices $\{u, v\}$ in G'. The edges in G'' are $(\{u, v\}, \{u, w\})\ \forall u, v, w : u \neq v, u \neq w, v \neq w$. An example of G'' derived from our example production P_{ex} is shown in Figure 11.

Any tree decomposition T'' of G'' can be transformed into a valid tree decomposition T' of G' by simply replacing each vertex in each node of T'' with both corresponding vertices in G'. If T'' witnesses a tree decomposition of optimal width $k'' = \mathbf{tw}(G'')$, each node in T'' will produce a node of size at most $2k''$ in T'. For any vertex v in G', one node in T'' must contain the clique corresponding to v in G''. Each vertex $\{v, w\}$ in G'' must be found in a contiguous subtree of T'', and these subtrees all include the node containing the clique for v. The occurrences of v in T' are the union of these contiguous subtrees, which must itself form a contiguous subtree. Furthermore, each edge (u, v) in G' corresponds to some vertex in G'', so u and v must occur together in some node of T'. Combining these two properties, we see that T' is a valid tree decomposition of G'. From the construction, if SOL is the treewidth of T', we are guaranteed that

$$SOL \leq 2\mathbf{tw}(G'') \tag{6}$$

In the other direction, any tree decomposition T' of G' can be transformed into a tree decomposition T'' of G'' by simply replacing each occurrence of vertex v in a node of T' with all vertices $\{v, w\}$ in T''. The number of such vertices is the degree of v, $\Delta(v)$.

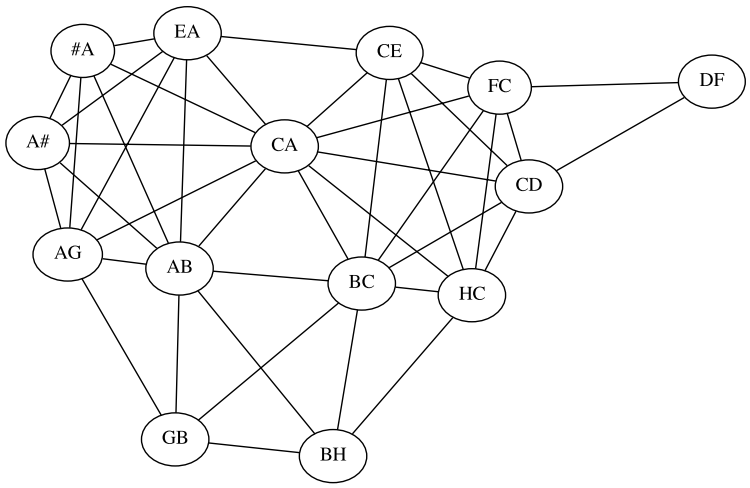

Figure 11
Dependency graph G''_{ex} derived from the example of Figure 9. Vertex #A corresponds to the beginning of the Eulerian tour through G'_{ex} and A# corresponds to the end of the tour; all other vertices correspond to edges in G'_{ex}.

Each vertex $\{v, w\}$ occurs in a contiguous subtree of T'' because v and w occurred in contiguous subtrees of T', and had to co-occur in at least one node of T'. Each edge in G'' comes from a clique for some vertex v in G', so the edge has both its endpoints in any node of T'' corresponding to a node of T' that contained v. Thus T'' is a valid tree decomposition of G''. We expand each node in the tree decomposition by at most the maximum degree of the graph $\Delta(G')$, and therefore

$$\mathbf{tw}(G'') \leq \Delta(G')\mathbf{tw}(G') \tag{7}$$

Assume that we have an efficient algorithm for computing the optimal parsing strategy of an arbitrary LCFRS rule. Consider the following algorithm for finding a tree decomposition of an input graph G:

- Transform G to G' of even degree, and construct LCFRS production P from an Eulerian tour of G'.
- Find the optimal parsing strategy for P.
- Translate this strategy into a tree decomposition of G'' of treewidth k'', and map this into a tree decomposition of G', and then remove all new nodes $\hat{e}$ to obtain a tree decomposition of G of treewidth SOL.

If $\mathbf{tw}(G'') = k''$, we have $SOL \leq 2k''$ from Equation (6), and $k'' \leq \Delta(G')\mathbf{tw}(G')$ from Equation (7). Putting these together:

$$SOL \leq 2\Delta(G')\mathbf{tw}(G')$$

and using Equations (4) and (5) to relate our result to the original graph G,

$$SOL \leq 4\Delta(G)\mathbf{tw}(G)$$

This last inequality proves the main result of this section

Theorem 2
An algorithm for finding the optimal parsing strategy of an arbitrary LCFRS production would imply a $4\Delta(G)$ approximation algorithm for treewidth.

Whether such an approximation algorithm for treewidth is possible is an open problem. The best-known result is the $O(\sqrt{\log k})$ approximation result of Feige, Hajiaghayi, and Lee (2005), which improves on the $O(\log k)$ result of Amir (2001). This indicates that, although polynomial-time factorization of LCFRS rules to optimize parsing complexity may be possible, it would require progress on general algorithms for treewidth.

5. Conclusion

We have demonstrated that a number of techniques used for specific parsing problems can be found algorithmically from declarative specifications of the grammar. Our method involves finding the optimal tree decomposition of a graph, which is in general an NP-complete problem. However, the relation to tree decomposition allows us to exploit existing algorithms for this problem, such as the linear time algorithm of Bodlaender (1996) for graphs of bounded treewidth. In practice, grammar rules are

typically small, and finding the tree decomposition is not computationally expensive, and in fact is trivial in comparison to the original parsing problem. Given the special structure of the graphs derived from LCFRS productions, however, we have explored whether finding optimal tree decompositions of these graphs, and therefore optimal parsing strategies for LCFRS productions, is also NP-complete. Although a polynomial time algorithm for this problem would not necessarily imply that $P = NP$, it would require progress on fundamental, well-studied problems in graph theory. Therefore, it does not seem possible to exploit the special structure of graphs derived from LCFRS productions.

Acknowledgments

This work was funded by NSF grants IIS-0546554 and IIS-0910611. We are grateful to Giorgio Satta for extensive discussions on grammar factorization, as well as for feedback on earlier drafts from Mehdi Hafezi Manshadi, Matt Post, and four anonymous reviewers.

References

Aho, Albert V. and Jeffery D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

Amir, Eyal. 2001. Efficient approximation for triangulation of minimum treewidth. In *17th Conference on Uncertainty in Artificial Intelligence*, pages 7–15, Seattle, WA.

Arnborg, Stefen, Derek G. Corneil, and Andrzej Proskurowski. 1987. Complexity of finding embeddings in a *k*-tree. *SIAM Journal of Algebraic and Discrete Methods*, 8:277–284.

Bar-Hillel, Yehoshua, M. Perles, and E. Shamir. 1961. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172. Reprinted in Y. Bar-Hillel. (1964). *Language and Information: Selected Essays on Their Theory and Application*, Addison-Wesley Reading, MA, pages 116–150.

Bodlaender, H. L. 1996. A linear time algorithm for finding tree decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317.

Bodlaender, Hans L., John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. 1995. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255.

Chekuri, Chandra and Anand Rajaraman. 1997. Conjunctive query containment revisited. In *Database Theory – ICDT '97*, volume 1186 of *Lecture Notes in Computer Science*. Springer, Berlin, pages 56–70.

Chiang, David. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

Dechter, Rina and Judea Pearl. 1989. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366.

Eisner, Jason and John Blatz. 2007. Program transformations for optimization of parsing algorithms and other weighted logic programs. In Shuly Wintner, editor, *Proceedings of FG 2006: The 11th Conference on Formal Grammar*. CSLI Publications, pages 45–85, Malaga.

Eisner, Jason and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Conference of the Association for Computational Linguistics (ACL-99)*, pages 457–464, College Park, MD.

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

Feige, Uriel, MohammadTaghi Hajiaghayi, and James R. Lee. 2005. Improved approximation algorithms for minimum-weight vertex separators. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 563–572, Baltimore, MD.

Flum, Jörg, Markus Frick, and Martin Grohe. 2002. Query evaluation via tree-decompositions. *Journal of the ACM*, 49(6):716–752.

Galley, Michel, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of the 2004 Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-04)*, pages 273–280, Boston, MA.

Gildea, Daniel. 2010. Optimal parsing strategies for Linear Context-Free

Rewriting Systems. In *Proceedings of the 2010 Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-10)*, pages 769–776, Los Angeles, CA.

Gildea, Daniel and Daniel Štefankovič. 2007. Worst-case synchronous grammar rules. In *Proceedings of the 2007 Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-07)*, pages 147–154, Rochester, NY.

Gómez-Rodríguez, Carlos, Marco Kuhlmann, Giorgio Satta, and David Weir. 2009. Optimal reduction of rule length in Linear Context-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, Boulder, CO.

Goodman, Joshua. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–605.

Huang, Liang, Hao Zhang, and Daniel Gildea. 2005. Machine translation as lexicalized parsing with hooks. In *International Workshop on Parsing Technologies (IWPT05)*, pages 65–73, Vancouver.

Huang, Liang, Hao Zhang, Daniel Gildea, and Kevin Knight. 2009. Binarization of synchronous context-free grammars. *Computational Linguistics*, 35(4):559–595.

Jensen, Finn V., Steffen L. Lauritzen, and Kristian G. Olesen. 1990. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282.

Johnson, Mark. 2007. Transforming projective bilexical dependency grammars into efficiently-parsable CFGs with unfold-fold. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 168–175, Prague.

Knuth, D. 1977. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5.

Kschischang, F. R., B. J. Frey, and H. A. Loeliger. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519.

Kuhlmann, Marco 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, Sydney.

Kuhlmann, Marco 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, Athens.

McAllester, David. 2002. On the complexity analysis of static analyses. *Journal of the ACM*, 49(4):512–537.

Melamed, I. Dan, Giorgio Satta, and Ben Wellington. 2004. Generalized multitext grammars. In *Proceedings of the 42nd Annual Conference of the Association for Computational Linguistics (ACL-04)*, pages 661–668, Barcelona.

Nederhof, M.-J. 2003. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–144.

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

Satta, Giorgio. 1992. Recognition of Linear Context-Free Rewriting Systems. In *Proceedings of the 30th Annual Conference of the Association for Computational Linguistics (ACL-92)*, pages 89–95, Newark, DE.

Shafer, G. and P. Shenoy. 1990. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–353.

Shieber, Stuart M., Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. *The Journal of Logic Programming*, 24(1-2):3–36.

Vijay-Shankar, K., 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)*, pages 104–111, Stanford, CA.

Wellington, Benjamin, 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.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403.

Zhang, Hao and Daniel Gildea. 2007. Factorization of synchronous context-free grammars in linear time. In *NAACL Workshop on Syntax and Structure in Statistical Translation (SSST)*, pages 25–32, Rochester, NY.