Optimal Head-Driven Parsing Complexity for Linear Context-Free Rewriting Systems

Pierluigi Crescenzi
Dip. di Sistemi e Informatica
Università di Firenze

Daniel Gildea
Computer Science Dept.
University of Rochester

Andrea Marino
Dip. di Sistemi e Informatica
Università di Firenze

Gianluca Rossi
Dip. di Matematica
Università di Roma *Tor Vergata*

Giorgio Satta
Dip. di Ingegneria dell'Informazione
Università di Padova

Abstract

We study the problem of finding the best *head-driven* parsing strategy for Linear Context-Free Rewriting System productions. A head-driven strategy must begin with a specified righthand-side nonterminal (the head) and add the remaining nonterminals one at a time in any order. We show that it is NP-hard to find the best head-driven strategy in terms of either the time or space complexity of parsing.

1 Introduction

Linear Context-Free Rewriting Systems (LCFRSs) (Vijay-Shankar et al., 1987) constitute a very general grammatical formalism which subsumes context-free grammars (CFGs) and tree adjoining grammars (TAGs), as well as the synchronous context-free grammars (SCFGs) and synchronous tree adjoining grammars (STAGs) used as models in machine translation.[1] LCFRSs retain the fundamental property of CFGs that grammar nonterminals rewrite independently, but allow nonterminals to generate discontinuous phrases, that is, to generate more than one span in the string being produced. This important feature has been recently exploited by Maier and Søgaard (2008) and Kallmeyer and Maier (2010) for modeling phrase structure treebanks with discontinuous constituents, and by Kuhlmann and Satta (2009) for modeling non-projective dependency treebanks.

The rules of a LCFRS can be analyzed in terms of the properties of *rank* and *fan-out*. Rank is the number of nonterminals on the right-hand side (rhs) of a rule, while fan-out is the number of spans of the string generated by the nonterminal in the left-hand side (lhs) of the rule. CFGs are equivalent to LCFRSs with fan-out one, while TAGs are one type of LCFRSs with fan-out two. Rambow and Satta (1999) show that rank and fan-out induce an infinite, two-dimensional hierarchy in terms of generative power; while CFGs can always be reduced to rank two (Chomsky Normal Form), this is not the case for LCFRSs with any fan-out greater than one.

General algorithms for parsing LCFRSs build a dynamic programming chart of recognized nonterminals bottom-up, in a manner analogous to the CKY algorithm for CFGs (Hopcroft and Ullman, 1979), but with time and space complexity that are dependent on the rank and fan-out of the grammar rules. Whenever it is possible, binarization of LCFRS rules, or reduction of rank to two, is therefore important for parsing, as it reduces the time complexity needed for dynamic programming. This has lead to a number of binarization algorithms for LCFRSs, as well as *factorization* algorithms that factor rules into new rules with smaller rank, without necessarily reducing rank all the way to two. Kuhlmann and Satta (2009) present an algorithm for binarizing certain LCFRS rules without increasing their fan-out, and Sagot and Satta (2010) show how to reduce rank to the lowest value possible for LCFRS rules of fan-out two, again without increasing fan-out. Gómez-Rodríguez et al. (2010) show how to factorize *well-nested* LCFRS rules of arbitrary fan-out for efficient parsing.

In general there may be a trade-off required between rank and fan-out, and a few recent papers have investigated this trade-off taking gen-

[1] To be more precise, SCFGs and STAGs generate languages composed by pair of strings, while LCFRSs generate string languages. We can abstract away from this difference by assuming concatenation of components in a string pair.

Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics, pages 450–459,
Portland, Oregon, June 19-24, 2011. ©2011 Association for Computational Linguistics

eral LCFRS rules as input. Gómez-Rodríguez et al. (2009) present an algorithm for binarization of LCFRSs while keeping fan-out as small as possible. The algorithm is exponential in the resulting fan-out, and Gómez-Rodríguez et al. (2009) mention as an important open question whether polynomial-time algorithms to minimize fan-out are possible. Gildea (2010) presents a related method for binarizing rules while keeping the time complexity of parsing as small as possible. Binarization turns out to be possible with no penalty in time complexity, but, again, the factorization algorithm is exponential in the resulting time complexity. Gildea (2011) shows that a polynomial time algorithm for factorizing LCFRSs in order to minimize time complexity would imply an improved approximation algorithm for the well-studied graph-theoretic property known as treewidth. However, whether the problem of factorizing LCFRSs in order to minimize time complexity is NP-hard is still an open question in the above works.

Similar questions have arisen in the context of machine translation, as the SCFGs used to model translation are also instances of LCFRSs, as already mentioned. For SCFG, Satta and Peserico (2005) showed that the exponent in the time complexity of parsing algorithms must grow at least as fast as the square root of the rule rank, and Gildea and Štefankovič (2007) tightened this bound to be linear in the rank. However, neither paper provides an algorithm for finding the best parsing strategy, and Huang et al. (2009) mention that whether finding the optimal parsing strategy for an SCFG rule is NP-hard is an important problem for future work.

In this paper, we investigate the problem of rule binarization for LCFRSs in the context of *head-driven* parsing strategies. Head-driven strategies begin with one rhs symbol, and add one nonterminal at a time. This rules out any factorization in which two subsets of nonterminals of size greater than one are combined in a single step. Head-driven strategies allow for the techniques of lexicalization and Markovization that are widely used in (projective) statistical parsing (Collins, 1997). The statistical LCFRS parser of Kallmeyer and Maier (2010) binarizes rules head-outward, and therefore adopts what we refer to as a head-driven strategy. However, the binarization used by Kallmeyer and Maier (2010) simply proceeds left to right through the rule, without considering the impact of the parsing strategy on either time or space complexity. We examine the question of whether we can efficiently find the strategy that minimizes either the time complexity or the space complexity of parsing. While a naive algorithm can evaluate all $r!$ head-driven strategies in time $O(n \cdot r!)$, where r is the rule's rank and n is the total length of the rule's description, we wish to determine whether a polynomial-time algorithm is possible.

Since parsing problems can be cast in terms of logic programming (Shieber et al., 1995), we note that our problem can be thought of as a type of query optimization for logic programming. Query optimization for logic programming is NP-complete since query optimization for even simple conjunctive database queries is NP-complete (Chandra and Merlin, 1977). However, the fact that variables in queries arising from LCFRS rules correspond to the endpoints of spans in the string to be parsed means that these queries have certain structural properties (Gildea, 2011). We wish to determine whether the structure of LCFRS rules makes efficient factorization algorithms possible.

In the following, we show both the the time- and space-complexity problems to be NP-hard for head-driven strategies. We provide what is to our knowledge the first NP-hardness result for a grammar factorization problem, which we hope will aid in understanding parsing algorithms in general.

2 LCFRSs and parsing complexity

In this section we briefly introduce LCFRSs and define the problem of optimizing head-driven parsing complexity for these formalisms. For a positive integer n, we write $[n]$ to denote the set $\{1, \ldots, n\}$.

As already mentioned in the introduction, LCFRSs generate tuples of strings over some finite alphabet. This is done by associating each production p of a grammar with a function g that takes as input the tuples generated by the nonterminals in p's rhs, and rearranges their string components into a new tuple, possibly adding some alphabet symbols.

Let V be some finite alphabet. We write V^* for the set of all (finite) strings over V. For natural numbers $r \geq 0$ and $f, f_1, \ldots, f_r \geq 1$, consider a function $g : (V^*)^{f_1} \times \cdots \times (V^*)^{f_r} \rightarrow (V^*)^f$ defined by an equation of the form

$$g(\langle x_{1,1}, \ldots, x_{1,f_1} \rangle, \ldots, \langle x_{r,1}, \ldots, x_{r,f_r} \rangle) = \vec{\alpha} \ .$$

Here the $x_{i,j}$'s denote variables over strings in V^*, and $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_f \rangle$ is an f-tuple of strings over g's argument variables and symbols in V. We say that g is **linear, non-erasing** if $\vec{\alpha}$ contains *exactly one* occurrence of each argument variable. We call r and f the **rank** and the **fan-out** of g, respectively, and write $r(g)$ and $f(g)$ to denote these quantities.

Example 1 $g_1(\langle x_{1,1}, x_{1,2} \rangle) = \langle x_{1,1}x_{1,2} \rangle$ takes as input a tuple with two strings and returns a tuple with a single string, obtained by concatenating the components in the input tuple. $g_2(\langle x_{1,1}, x_{1,2} \rangle) = \langle ax_{1,1}b, cx_{1,2}d \rangle$ takes as input a tuple with two strings and wraps around these strings with symbols $a, b, c, d \in V$. Both functions are linear, non-erasing, and we have $r(g_1) = r(g_2) = 1$, $f(g_1) = 1$ and $f(g_2) = 2$. □

A **linear context-free rewriting system** is a tuple $G = (V_N, V_T, P, S)$, where V_N and V_T are finite, disjoint alphabets of nonterminal and terminal symbols, respectively. Each $A \in V_N$ is associated with a value $f(A)$, called its **fan-out**. The nonterminal S is the start symbol, with $f(S) = 1$. Finally, P is a set of productions of the form

$$p : \ \ A \ \ \rightarrow \ \ g(A_1, A_2, \ldots, A_{r(g)}) \ , \qquad (1)$$

where $A, A_1, \ldots, A_{r(g)} \in V_N$, and $g : (V_T^*)^{f(A_1)} \times \cdots \times (V_T^*)^{f(A_{r(g)})} \rightarrow (V_T^*)^{f(A)}$ is a linear, non-erasing function.

Production (1) can be used to transform the $r(g)$ string tuples generated by the nonterminals $A_1, \ldots, A_{r(g)}$ into a tuple of $f(A)$ strings generated by A. The values $r(g)$ and $f(g)$ are called the **rank** and **fan-out** of p, respectively, written $r(p)$ and $f(p)$. Given that $f(S) = 1$, S generates a set of strings, defining the language $L(G)$.

Example 2 Let g_1 and g_2 be as in Example 1, and let $g_3() = \langle \varepsilon, \varepsilon \rangle$. Consider the LCFRS G defined by the productions $p_1 : S \rightarrow g_1(A)$, $p_2 : A \rightarrow g_2(A)$ and $p_3 : A \rightarrow g_3()$. We have $f(S) = 1$, $f(A) = f(G) = 2$, $r(p_3) = 0$ and $r(p_1) = r(p_2) = r(G) = 1$. We have $L(G) = \{a^n b^n c^n d^n \mid n \geq 1\}$. For instance, the string $a^3b^3c^3d^3$ is generated by means of the following bottom-up process. First, the tuple $\langle \varepsilon, \varepsilon \rangle$ is generated by A through p_3. We then iterate three times the application of p_2 to $\langle \varepsilon, \varepsilon \rangle$, resulting in the tuple $\langle a^3b^3, c^3d^3 \rangle$. Finally, the tuple (string) $\langle a^3b^3c^3d^3 \rangle$ is generated by S through application of p_1. □

fan-out	*strategy*
4	$((A_1 \oplus A_4) \oplus A_3)^* \oplus A_2$
3	$(A_1 \oplus A_4)^* \oplus (A_2 \oplus A_3)$
3	$((A_1 \oplus A_2)^* \oplus A_4) \oplus A_3$
2	$((A_2^* \oplus A_3) \oplus A_4) \oplus A_1$

Figure 1: Some parsing strategies for production p in Example 3, and the associated maximum value for fan-out. Symbol $\oplus$ denotes the merging operation, and superscript $*$ marks the first step in the strategy in which the highest fan-out is realized.

Existing parsing algorithms for LCFRSs exploit dynamic programming. These algorithms compute partial parses of the input string w, represented by means of specialized data structures called items. Each **item** indexes the boundaries of the segments of w that are spanned by the partial parse. In the special case of parsing based on CFGs, an item consists of two indices, while for TAGs four indices are required.

In the general case of LCFRSs, parsing of a production p as in (1) can be carried out in $r(g) - 1$ steps, collecting already available parses for nonterminals $A_1, \ldots, A_{r(g)}$ one at a time, and 'merging' these into intermediate partial parses. We refer to the order in which nonterminals are merged as a parsing strategy, or, equivalently, a factorization of the original grammar rule. Any parsing strategy results in a complete parse of p, spanning $f(p) = f(A)$ segments of w and represented by some item with $2f(A)$ indices. However, intermediate items obtained in the process might span more than $f(A)$ segments. We illustrate this through an example.

Example 3 Consider a linear non-erasing function $g(\langle x_{1,1}, x_{1,2} \rangle, \langle x_{2,1}, x_{2,2} \rangle, \langle x_{3,1}, x_{3,2} \rangle, \langle x_{4,1}, x_{4,2} \rangle) = \langle x_{1,1}x_{2,1}x_{3,1}x_{4,1}, \ x_{3,2}x_{2,2}x_{4,2}x_{1,2} \rangle$, and a production $p : A \rightarrow g(A_1, A_2, A_3, A_4)$, where all the nonterminals involved have fan-out 2. We could parse p starting from A_1, and then merging with A_4,

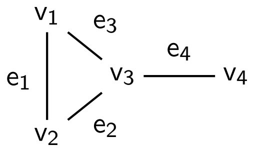

Figure 2: Example input graph for our construction of an LCFRS production.

A_3, and A_2. In this case, after we have collected the first three nonterminals, we have obtained a partial parse having fan-out 4, that is, an item spanning 4 segments of the input string. Alternatively, we could first merge A_1 and A_4, then merge A_2 and A_3, and finally merge the two obtained partial parses. This strategy is slightly better, resulting in a maximum fan-out of 3. Other possible strategies can be explored, displayed in Figure 1. It turns out that the best parsing strategy leads to fan-out 2. □

The maximum fan-out f realized by a parsing strategy determines the space complexity of the parsing algorithm. For an input string w, items will require (in the worst-case) $2f$ indices, each taking $\mathcal{O}(|w|)$ possible values. This results in space complexity of $\mathcal{O}(|w|^{2f})$. In the special cases of parsing based on CFGs and TAGs, this provides the well-known space complexity of $\mathcal{O}(|w|^2)$ and $\mathcal{O}(|w|^4)$, respectively.

It can also be shown that, if a partial parse having fan-out f is obtained by means of the combination of two partial parses with fan-out f_1 and f_2, respectively, the resulting time complexity will be $\mathcal{O}(|w|^{f+f_1+f_2})$ (Seki et al., 1991; Gildea, 2010). As an example, in the case of parsing based on CFGs, nonterminals as well as partial parses all have fan-out one, resulting in the standard time complexity of $\mathcal{O}(|w|^3)$ of dynamic programming methods. When parsing with TAGs, we have to manipulate objects with fan-out two (in the worst case), resulting in time complexity of $\mathcal{O}(|w|^6)$.

We investigate here the case of general LCFRS productions, whose internal structure is considerably more complex than the context-free or the tree adjoining case. Optimizing the parsing complexity for a production means finding a parsing strategy that results in minimum space or time complexity.

We now turn the above optimization problems into decision problems. In the MIN SPACE STRATEGY problem one takes as input an LCFRS production p and an integer k, and must decide whether there exists a parsing strategy for p with maximum fan-out not larger than k. In the MIN TIME STRATEGY problem one is given p and k as above and must decide whether there exists a parsing strategy for p such that, in any of its steps merging two partial parses with fan-out f_1 and f_2 and resulting in a partial parse with fan-out f, the relation $f + f_1 + f_2 \leq k$ holds.

In this paper we investigate the above problems in the context of a specific family of linguistically motivated parsing strategies for LCFRSs, called head-driven. In a **head-driven** strategy, one always starts parsing a production p from a fixed nonterminal in its rhs, called the **head** of p, and merges the remaining nonterminals one at a time with the partial parse containing the head. Thus, under these strategies, the construction of partial parses that do not include the head is forbidden, and each parsing step involves at most one partial parse. In Figure 1, all of the displayed strategies but the one in the second line are head-driven (for different choices of the head).

3 NP-completeness results

For an LCFRS production p, let H be its head nonterminal, and let $A_1, \ldots, A_n$ be all the non-head nonterminals in p's rhs, with $n + 1 = r(p)$. A head-driven parsing strategy can be represented as a permutation π over the set $[n]$, prescribing that the non-head nonterminals in p's rhs should be merged with H in the order $A_{\pi(1)}, A_{\pi(2)}, \ldots, A_{\pi(n)}$. Note that there are $n!$ possible head-driven parsing strategies.

To show that MIN SPACE STRATEGY is NP-hard under head-driven parsing strategies, we reduce from the MIN CUT LINEAR ARRANGEMENT problem, which is a decision problem over (undirected) graphs. Given a graph $M = (V, E)$ with set of vertices V and set of edges E, a **linear arrangement** of M is a bijective function h from V to $[n]$, where $|V| = n$. The **cutwidth** of M at gap $i \in [n-1]$ and with respect to a linear arrangement h is the number of edges crossing the gap between the i-th vertex and its successor:

$$cw(M, h, i) = |\{(u, v) \in E \mid h(u) \leq i < h(v)\}| \,.$$

$$p: \quad A \rightarrow g(H, A_1, A_2, A_3, A_4)$$

$$\begin{aligned} g(&\langle x_{H,e_1}, x_{H,e_2}, x_{H,e_3}, x_{H,e_4}\rangle, \langle x_{A_1,e_1,l}, x_{A_1,e_1,r}, x_{A_1,e_3,l}, x_{A_1,e_3,r}\rangle, \langle x_{A_2,e_1,l}, x_{A_2,e_1,r}, x_{A_2,e_2,l}, x_{A_2,e_2,r}\rangle, \\ &\langle x_{A_3,e_2,l}, x_{A_3,e_2,r}, x_{A_3,e_3,l}, x_{A_3,e_3,r}, x_{A_3,e_4,l}, x_{A_3,e_4,r}\rangle, \langle x_{A_4,e_4,l}, x_{A_4,e_4,r}\rangle) = \\ &\langle \quad x_{A_1,e_1,l}x_{A_2,e_1,l}x_{H,e_1}x_{A_1,e_1,r}x_{A_2,e_1,r}, \quad x_{A_2,e_2,l}x_{A_3,e_2,l}x_{H,e_2}x_{A_2,e_2,r}x_{A_3,e_2,r}, \\ &\quad\; x_{A_1,e_3,l}x_{A_3,e_3,l}x_{H,e_3}x_{A_1,e_3,r}x_{A_3,e_3,r}, \quad x_{A_3,e_4,l}x_{A_4,e_4,l}x_{H,e_4}x_{A_3,e_4,r}x_{A_4,e_4,r} \quad \rangle \end{aligned}$$

Figure 3: The construction used to prove Theorem 1 builds the LCFRS production p shown, when given as input the graph of Figure 2.

The cutwidth of M is then defined as

$$cw(M) = \min_h \max_{i \in [n-1]} cw(M, h, i) \ .$$

In the MIN CUT LINEAR ARRANGEMENT problem, one is given as input a graph M and an integer k, and must decide whether $cw(M) \leq k$. This problem has been shown to be NP-complete (Gavril, 1977).

Theorem 1 *The* MIN SPACE STRATEGY *problem restricted to head-driven parsing strategies is NP-complete.*

PROOF We start with the NP-hardness part. Let $M = (V, E)$ and k be an input instance for MIN CUT LINEAR ARRANGEMENT, and let $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_q\}$. We assume there are no self loops in M, since these loops do not affect the value of the cutwidth and can therefore be removed. We construct an LCFRS production p and an integer k' as follows.

Production p has a head nonterminal H and a non-head nonterminal A_i for each vertex $v_i \in V$. We let H generate tuples with a string component for each edge $e_i \in E$. Thus, we have $f(H) = q$. Accordingly, we use variables x_{H,e_i}, for each $e_i \in E$, to denote the string components in tuples generated by H.

For each $v_i \in V$, let $E(v_i) \subseteq E$ be the set of edges impinging on v_i; thus $|E(v_i)|$ is the degree of v_i. We let A_i generate a tuple with two string components for each $e_j \in E(v_i)$. Thus, we have $f(A_i) = 2 \cdot |E(v_i)|$. Accordingly, we use variables $x_{A_i,e_j,l}$ and $x_{A_i,e_j,r}$, for each $e_j \in E(v_i)$, to denote the string components in tuples generated by A_i (here subscripts l and r indicate left and right positions, respectively; see below).

We set $r(p) = n + 1$ and $f(p) = q$, and define p by $A \rightarrow g(H, A_1, A_2, \ldots, A_n)$, with $g(t_H, t_{A_1}, \ldots, t_{A_n}) = \langle \alpha_1, \ldots, \alpha_q \rangle$. Here t_H is the tuple of variables for H and each t_{A_i}, $i \in [n]$, is the tuple of variables for A_i. Each string α_i, $i \in [q]$, is specified as follows. Let v_s and v_t be the endpoints of e_i, with $v_s, v_t \in V$ and $s < t$. We define

$$\alpha_i \quad = \quad x_{A_s,e_i,l}x_{A_t,e_i,l}x_{H,e_i}x_{A_s,e_i,r}x_{A_t,e_i,r} \ .$$

Observe that whenever edge e_i impinges on vertex v_j, then the left and right strings generated by A_j and associated with e_i wrap around the string generated by H and associated with the same edge. Finally, we set $k' = q + k$.

Example 4 Given the input graph of Figure 2, our reduction constructs the LCFRS production shown in Figure 3. Figure 4 gives a visualization of how the spans in this production fit together. For each edge in the graph of Figure 2, we have a group of five spans in the production: one for the head nonterminal, and two spans for each of the two nonterminals corresponding to the edge's endpoints. □

Assume now some head-driven parsing strategy π for p. For each $i \in [n]$, we define D_i^π to be the partial parse obtained after step i in π, consisting of the merge of nonterminals $H, A_{\pi(1)}, \ldots, A_{\pi(i)}$. Consider some edge $e_j = (v_s, v_t)$. We observe that for any D_i^π that includes or excludes both nonterminals A_s and A_t, the α_j component in the definition of p is associated with a single string, and therefore contributes with a single unit to the fan-out of the partial parse. On the other hand, if D_i^π includes only one nonterminal between A_s and A_t, the α_j component is associated with two strings and contributes with two units to the fan-out of the partial parse.

We can associate with π a linear arrangement h_π of M by letting $h_\pi(v_{\pi(i)}) = i$, for each $v_i \in V$. From the above observation on the fan-out of D_i^π,

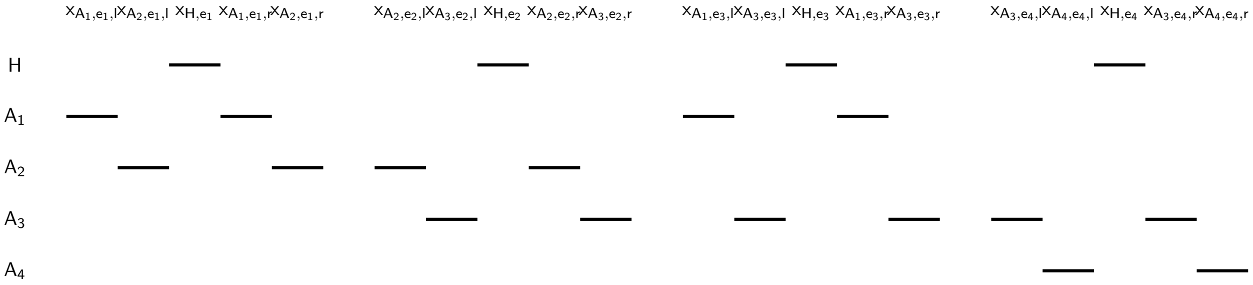

Figure 4: A visualization of how the spans for each nonterminal fit together in the left-to-right order defined by the production of Figure 3.

we have the following relation, for every $i \in [n-1]$:

$$f(D_i^\pi) \quad = \quad q + cw(M, h_\pi, i) \ .$$

We can then conclude that M, k is a positive instance of MIN CUT LINEAR ARRANGEMENT if and only if p, k' is a positive instance of MIN SPACE STRATEGY. This proves that MIN SPACE STRATEGY is NP-hard.

To show that MIN SPACE STRATEGY is in NP, consider a nondeterministic algorithm that, given an LCFRS production p and an integer k, guesses a parsing strategy π for p, and tests whether $f(D_i^\pi) \leq k$ for each $i \in [n]$. The algorithm accepts or rejects accordingly. Such an algorithm can clearly be implemented to run in polynomial time. ■

We now turn to the MIN TIME STRATEGY problem, restricted to head-driven parsing strategies. Recall that we are now concerned with the quantity $f_1 + f_2 + f$, where f_1 is the fan-out of some partial parse D, f_2 is the fan-out of a nonterminal A, and f is the fan out of the partial parse resulting from the merge of the two previous analyses.

We need to introduce the MODIFIED CUTWIDTH problem, which is a variant of the MIN CUT LINEAR ARRANGEMENT problem. Let $M = (V, E)$ be some graph with $|V| = n$, and let h be a linear arrangement for M. The **modified cutwidth** of M at position $i \in [n]$ and with respect to h is the number of edges crossing over the i-th vertex:

$$mcw(M, h, i) = |\{(u, v) \in E \mid h(u) < i < h(v)\}| \ .$$

The modified cutwidth of M is defined as

$$mcw(M) = \min_h \max_{i \in [n]} mcw(M, h, i) \ .$$

In the MODIFIED CUTWIDTH problem one is given as input a graph M and an integer k, and must decide whether $mcw(M) \leq k$. The MODIFIED CUTWIDTH problem has been shown to be NP-complete by Lengauer (1981). We strengthen this result below; recall that a cubic graph is a graph without self loops where each vertex has degree three.

Lemma 1 *The* MODIFIED CUTWIDTH *problem restricted to cubic graphs is NP-complete.*

PROOF The MODIFIED CUTWIDTH problem has been shown to be NP-complete when restricted to graphs of maximum degree three by Makedon et al. (1985), reducing from a graph problem known as bisection width (see also Monien and Sudborough (1988)). Specifically, the authors construct a graph G' of maximum degree three and an integer k' from an input graph $G = (V, E)$ with an even number n of vertices and an integer k, such that $mcw(G') \leq k'$ if and only if the **bisection width** $bw(G)$ of G is not greater than k, where

$$bw(G) = \min_{A,B \subseteq V} |\{(u, v) \in E \mid u \in A \wedge v \in B\}|$$

with $A \cap B = \emptyset$, $A \cup B = V$, and $|A| = |B|$.

The graph G' has vertices of degree two and three only, and it is based on a grid-like gadget $R(r, c)$; see Figure 5. For each vertex of G, G' includes a component $R(2n^4, 8n^4 + 8)$. Moreover, G' has a component called an H-shaped graph, containing left and right columns $R(3n^4, 12n^4 + 12)$ connected by a middle bar $R(2n^4, 12n^4 + 9)$; see Figure 6. From each of the n vertex components there is a sheaf of $2n^2$ edges connecting distinct degree 2 vertices in the component to $2n^2$ distinct degree 2 vertices in

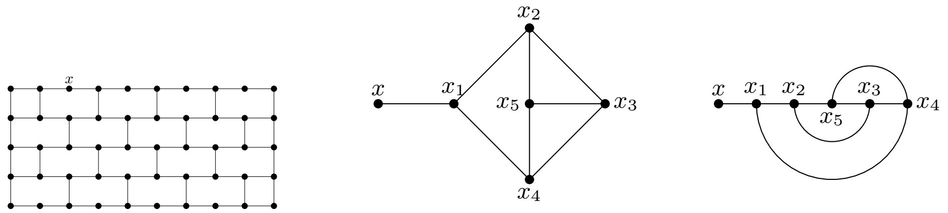

Figure 5: The $R(5, 10)$ component (left), the modification of its degree 2 vertex x (middle), and the corresponding arrangement (right).

the middle bar of the H-shaped graph. Finally, for each edge (v_i, v_j) of G there is an edge in G' connecting a degree 2 vertex in the component corresponding to the vertex v_i with a degree 2 vertex in the component corresponding to the vertex v_j. The integer k' is set to $3n^4 + n^3 + k - 1$.

Makedon et al. (1985) show that the modified cutwidth of $R(r, c)$ is $r - 1$ whenever $r \geq 3$ and $c \geq 4r + 8$. They also show that an optimal linear arrangement for G' has the form depicted in Figure 6, where half of the vertex components are to the left of the H-shaped graph and all the other vertex components are to the right. In this arrangement, the modified cutwidth is attested by the number of edges crossing over the vertices in the left and right columns of the H-shaped graph, which is equal to

$$3n^4 - 1 + \frac{n}{2}2n^2 + \gamma = 3n^4 + n^3 + \gamma - 1 \quad (2)$$

where γ denotes the number of edges connecting vertices to the left with vertices to the right of the H-shaped graph. Thus, $bw(G) \leq k$ if and only if $mcw(G') \leq k'$.

All we need to show now is how to modify the components of G' in order to make it cubic.

Modifying the vertex components All vertices x of degree 2 of the components corresponding to a vertex in G can be transformed into a vertex of degree 3 by adding five vertices $x_1, \ldots, x_5$ connected as shown in the middle bar of Figure 5. Observe that these five vertices can be positioned in the arrangement immediately after x in the order x_1, x_2, x_5, x_3, x_4 (see the right part of the figure). The resulting maximum modified cutwidth can increase by 2 in correspondence of vertex x_5. Since the vertices of these components, in the optimal arrangement, have modified cutwidth smaller than $2n^4 + n^3 + n^2$, an increase by 2 is still smaller than the maximum modified cutwidth of the entire graph, which is $3n^4 + \mathcal{O}(n^3)$.

Modifying the middle bar of the H-shaped graph The vertices of degree 2 of this part of the graph can be modified as in the previous paragraph. Indeed, in the optimal arrangement, these vertices have modified cutwidth smaller than $2n^4 + 2n^3 + n^2$, and an increase by 2 is still smaller than the maximum cutwidth of the entire graph.

Modifying the left/right columns of the H-shaped graph We replace the two copies of component $R(3n^4, 12n^4 + 12)$ with two copies of the new component $D(3n^4, 24n^4 + 16)$ shown in Figure 7, which is a cubic graph. In order to prove that relation (2) still holds, it suffices to show that the modified cutwidth of the component $D(r, c)$ is still $r - 1$ whenever $r \geq 3$ and $c = 8r + 16$.

We first observe that the linear arrangement obtained by visiting the vertices of $D(r, c)$ from top to bottom and from left to right has modified cutwidth $r - 1$. Let us now prove that, for any partition of the vertices into two subsets V_1 and V_2 with $|V_1|, |V_2| \geq 4r^2$, there exist at least r disjoint paths between vertices of V_1 and vertices of V_2. To this aim, we distinguish the following three cases.

- Any row has (at least) one vertex in V_1 and one vertex in V_2: in this case, it is easy to see there exist at least r disjoint paths between vertices of V_1 and vertices of V_2.

- There exist at least $3r$ 'mixed' columns, that is, columns with (at least) one vertex in V_1 and one vertex in V_2. Again, it is easy to see that there exist at least r disjoint paths between vertices

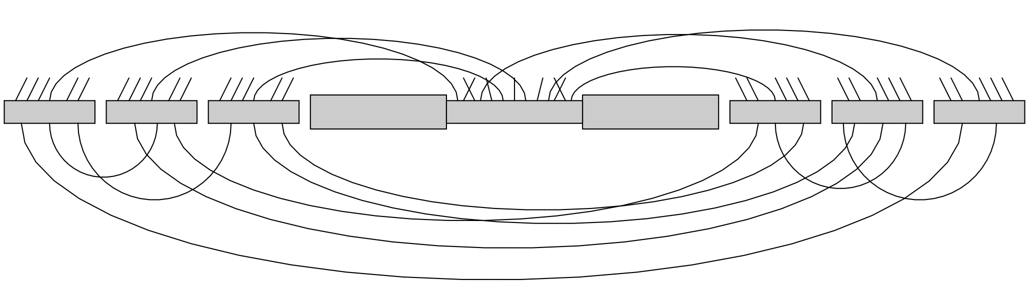

Figure 6: The optimal arrangement of G'.

of V_1 and vertices of V_2 (at least one path every three columns).

- The previous two cases do not apply. Hence, there exists a row entirely formed by vertices of V_1 (or, equivalently, of V_2). The worst case is when this row is the smallest one, that is, the one with $\frac{(c-3-1)}{2}+1 = 4r+7$ vertices. Since at most $3r-1$ columns are mixed, we have that at most $(3r-1)(r-2) = 3r^2-7r+2$ vertices of V_2 are on these mixed columns. Since $|V_2| \geq 4r^2$, this implies that at least r columns are fully contained in V_2. On the other hand, at least $4r+7-(3r-1) = r+8$ columns are fully contained in V_1. If the V_1-columns interleave with the V_2-columns, then there exist at least $2(r-1)$ disjoint paths between vertices of V_1 and vertices of V_2. Otherwise, all the V_1-columns precede or follow all the V_2-columns (this corresponds to the optimal arrangement): in this case, there are r disjoint paths between vertices of V_1 and vertices of V_2.

Observe now that any linear arrangement partitions the set of vertices in $D(r,c)$ into the sets V_1, consisting of the first $4r^2$ vertices in the arrangement, and V_2, consisting of all the remaining vertices. Since there are r disjoint paths connecting V_1 and V_2, there must be at least $r-1$ edges passing over every vertex in the arrangement which is assigned to a position between the $(4r^2+1)$-th and the position $4r^2+1$ from the right end of the arrangement: thus, the modified cutwidth of any linear arrangement of the vertices of $D(r,c)$ is at least $r-1$.

We can then conclude that the original proof of Makedon et al. (1985) still applies, according to relation (2). ■

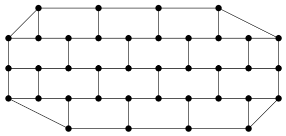

Figure 7: The $D(5,10)$ component.

We can now reduce from the MODIFIED CUTWIDTH problem for cubic graphs to the MIN TIME STRATEGY problem restricted to head-driven parsing strategies.

Theorem 2 *The* MIN TIME STRATEGY *problem restricted to head-driven parsing strategies is NP-complete.*

PROOF We consider hardness first. Let M and k be an input instance of the MODIFIED CUTWIDTH problem restricted to cubic graphs, where $M = (V,E)$ and $V = \{v_1,\ldots,v_n\}$. We construct an LCFRS production p exactly as in the proof of Theorem 1, with rhs nonterminals $H, A_1, \ldots, A_n$. We also set $k' = 2\cdot k + 2\cdot |E| + 9$.

Assume now some head-driven parsing strategy π for p. After parsing step $i \in [n]$, we have a partial parse D_i^π consisting of the merge of nonterminals $H, A_{\pi(1)}, \ldots, A_{\pi(i)}$. We write $tc(p,\pi,i)$ to denote the exponent of the time complexity due to step i. As already mentioned, this quantity is defined as the sum of the fan-out of the two antecedents involved in the parsing step and the fan-out of its result:

$$tc(p,\pi,i) \quad = \quad f(D_{i-1}^\pi) + f(A_{\pi(i)}) + f(D_i^\pi) \ .$$

Again, we associate with π a linear arrangement h_π of M by letting $h_\pi(v_{\pi(i)}) = i$, for each $v_i \in V$. As in the proof of Theorem 1, the fan-out of D_i^π is then related to the cutwidth of the linear arrangement h_π of M at position i by

$$f(D_i^\pi) \quad = \quad |E| + cw(M, h_\pi, i) \ .$$

From the proof of Theorem 1, the fan-out of nonterminal $A_{\pi(i)}$ is twice the degree of vertex $v_{\pi(i)}$, denoted by $|E(v_{\pi(i)})|$. We can then rewrite the above equation in terms of our graph M:

$$\begin{aligned} tc(p, \pi, i) = 2 \cdot |E| + cw(M, h_\pi, i-1) + \\ + 2 \cdot |E(v_{\pi(i)})| + cw(M, h_\pi, i) \ . \end{aligned}$$

The following general relation between cutwidth and modified cutwidth is rather intuitive:

$$\begin{aligned} mcw(M, h_\pi, i) = \frac{1}{2} \cdot [cw(M, h_\pi, i-1) + \\ - |E(v_{\pi(i)})| + cw(M, h_\pi, i)] \ . \end{aligned}$$

Combining the two equations above we obtain:

$$\begin{aligned} tc(p, \pi, i) \quad = \quad 2 \cdot |E| + 3 \cdot |E(v_{\pi(i)})| + \\ + 2 \cdot mcw(M, h_\pi, i) \ . \end{aligned}$$

Because we are restricting M to the class of cubic graphs, we can write:

$$tc(p, \pi, i) \quad = \quad 2 \cdot |E| + 9 + 2 \cdot mcw(M, h_\pi, i) \ .$$

We can thus conclude that there exists a head-driven parsing strategy for p with time complexity not greater than $2 \cdot |E| + 9 + 2 \cdot k = k'$ if and only if $mcw(M) \leq k$.

The membership of MODIFIED CUTWIDTH in NP follows from an argument similar to the one in the proof of Theorem 1. ■

We have established the NP-completeness of both the MIN SPACE STRATEGY and the MIN TIME STRATEGY decision problems. It is now easy to see that the problem of finding a space- or time-optimal parsing strategy for a LCFRS production is NP-hard as well, and thus cannot be solved in polynomial (deterministic) time unless P = NP.

4 Concluding remarks

Head-driven strategies are important in parsing based on LCFRSs, both in order to allow statistical modeling of head-modifier dependencies and in order to generalize the Markovization of CFG parsers to parsers with discontinuous spans. However, there are $n!$ possible head-driven strategies for an LCFRS production with a head and n modifiers. Choosing among these possible strategies affects both the time and the space complexity of parsing. In this paper we have shown that optimizing the choice according to either metric is NP-hard. To our knowledge, our results are the first NP-hardness results for a grammar factorization problem.

SCFGs and STAGs are specific instances of LCFRSs. Grammar factorization for synchronous models is an important component of current machine translation systems (Zhang et al., 2006), and algorithms for factorization have been studied by Gildea et al. (2006) for SCFGs and by Nesson et al. (2008) for STAGs. These algorithms do not result in what we refer as head-driven strategies, although, as machine translation systems improve, lexicalized rules may become important in this setting as well. However, the results we have presented in this paper do not carry over to the above mentioned synchronous models, since the fan-out of these models is bounded by two, while in our reductions in Section 3 we freely use unbounded values for this parameter. Thus the computational complexity of optimizing the choice of the parsing strategy for SCFGs is still an open problem.

Finally, our results for LCFRSs only apply when we restrict ourselves to head-driven strategies. This is in contrast to the findings of Gildea (2011), which show that, for unrestricted parsing strategies, a polynomial time algorithm for minimizing parsing complexity would imply an improved approximation algorithm for finding the treewidth of general graphs. Our result is stronger, in that it shows strict NP-hardness, but also weaker, in that it applies only to head-driven strategies. Whether NP-hardness can be shown for unrestricted parsing strategies is an important question for future work.

Acknowledgments

The first and third authors are partially supported from the Italian PRIN project DISCO. The second author is partially supported by NSF grants IIS-0546554 and IIS-0910611.

References

Ashok K. Chandra and Philip M. Merlin. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proc. ninth annual ACM symposium on Theory of computing*, STOC '77, pages 77–90.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proc. 35th Annual Conference of the Association for Computational Linguistics (ACL-97)*, pages 16–23.

F. Gavril. 1977. Some NP-complete problems on graphs. In *Proc. 11th Conf. on Information Sciences and Systems*, pages 91–95.

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

Daniel Gildea, Giorgio Satta, and Hao Zhang. 2006. Factoring synchronous grammars by sorting. In *Proc. International Conference on Computational Linguistics/Association for Computational Linguistics (COLING/ACL-06) Poster Session*, pages 279–286.

Daniel Gildea. 2010. Optimal parsing strategies for Linear Context-Free Rewriting Systems. In *Proc. 2010 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-10)*, pages 769–776.

Daniel Gildea. 2011. Grammar factorization by tree decomposition. *Computational Linguistics*, 37(1):231–248.

Carlos Gómez-Rodríguez, Marco Kuhlmann, Giorgio Satta, and David Weir. 2009. Optimal reduction of rule length in Linear Context-Free Rewriting Systems. In *Proc. 2009 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-09)*, pages 539–547.

Carlos Gómez-Rodríguez, Marco Kuhlmann, and Giorgio Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In *Proc. 2010 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-10)*, pages 276–284, Los Angeles, California.

John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

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

Laura Kallmeyer and Wolfgang Maier. 2010. Data-driven parsing with probabilistic linear context-free rewriting systems. In *Proc. 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 537–545.

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

Thomas Lengauer. 1981. Black-white pebbles and graph separation. *Acta Informatica*, 16:465–475.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In Philippe de Groote, editor, *Proc. 13th Conference on Formal Grammar (FG-2008)*, pages 61–76, Hamburg, Germany. CSLI Publications.

F. S. Makedon, C. H. Papadimitriou, and I. H. Sudborough. 1985. Topological bandwidth. *SIAM J. Alg. Disc. Meth.*, 6(3):418–444.

B. Monien and I.H. Sudborough. 1988. Min cut is NP-complete for edge weighted trees. *Theor. Comput. Sci.*, 58:209–229.

Rebecca Nesson, Giorgio Satta, and Stuart M. Shieber. 2008. Optimal k-arization of synchronous tree adjoining grammar. In *Proc. 46th Annual Meeting of the Association for Computational Linguistics (ACL-08)*, pages 604–612.

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

Benoît Sagot and Giorgio Satta. 2010. Optimal rank reduction for linear context-free rewriting systems with fan-out two. In *Proc. 48th Annual Meeting of the Association for Computational Linguistics*, pages 525–533, Uppsala, Sweden.

Giorgio Satta and Enoch Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 803–810, Vancouver, Canada.

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

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

K. Vijay-Shankar, D. L. Weir, and A. K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proc. 25th Annual Conference of the Association for Computational Linguistics (ACL-87)*, pages 104–111.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proc. 2006 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-06)*, pages 256–263.