Complexity, Parsing, and Factorization of Tree-Local Multi-Component Tree-Adjoining Grammar

Rebecca Nesson, Giorgio Satta
and Stuart Shieber

TR-05-08

Computer Science Group
Harvard University
Cambridge, Massachusetts

Complexity, Parsing, and Factorization of Tree-Local Multi-Component Tree-Adjoining Grammar

Rebecca Nesson*
School of Engineering and Applied Sciences
Harvard University

Giorgio Satta**
Department of Information Engineering
University of Padua

Stuart Shieber†
School of Engineering and Applied Sciences
Harvard University

Tree-Local Multi-Component Tree-Adjoining Grammar (TL-MCTAG) is an appealing formalism for natural language representation because it arguably allows the encapsulation of the appropriate domain of locality within its elementary structures. Although TL-MCTAG was first introduced by Weir (1988) and shown at that time to be equivalent in expressivity to TAG, the complexity of TL-MCTAG is still not well-understood. This paper offers a thorough examination of the problem of TL-MCTAG recognition, showing that even highly restricted forms of TL-MCTAG are NP-complete to recognize. However, in spite of the provable difficulty of the recognition problem, we offer several algorithms that can substantially improve processing efficiency. First, we present a parsing algorithm that improves on the baseline parsing method and runs in polynomial time when both the fan-out and rank of the input grammar are bounded. Second, we offer an optimal, efficient algorithm for factorizing a grammar to produce a strongly-equivalent TL-MCTAG grammar with the rank of the grammar minimized.

1. Introduction

Tree-Local Multi-Component Tree-Adjoining Grammar (TL-MCTAG) is an appealing formalism for natural language representation because it arguably allows the encapsulation of the appropriate domain of locality within its elementary structures (Kallmeyer and Romero 2007). As a result, it has frequently been put to use in a growing body of research into incorporating semantics into the Tree-Adjoining Grammar (TAG) framework (Kallmeyer and Joshi 2003; Han 2006; Nesson and Shieber 2006, 2007). Although TL-MCTAG was first introduced by Weir (1988) and shown at that time to be equivalent in expressivity to TAG, the complexity of TL-MCTAG is still not well-understood. Perhaps because of its equivalence to TAG, questions of processing efficiency have not adequately been addressed. This paper offers a thorough examination of the problem

* Email: nesson@seas.harvard.edu
** Email: satta@dei.unipd.it
† Email: shieber@seas.harvard.edu

of TL-MCTAG recognition, showing that even highly restricted forms of TL-MCTAG are NP-complete to recognize. However, in spite of the provable difficulty of the recognition problem, we offer several algorithms that can substantially improve processing efficiency. First, we present a parsing algorithm that improves on the baseline parsing method and runs in polynomial time when both the fan-out—the maximum number of trees in a tree set—and rank—the maximum number of children of a node in a valid derivation tree—of the input grammar are bounded.[1] Second, we offer an optimal, efficient algorithm for factorizing a grammar to produce a strongly-equivalent TL-MCTAG grammar with the rank of the grammar minimized.

1.1 Summary of Results

Tree-Adjoining Grammar (TAG) is a mildly context-sensitive grammar formalism widely used in natural-language processing. Multicomponent TAG (MCTAG) refers to a group of formalisms that generalize TAG by allowing elementary structures to be sets of TAG trees. One member of the MCTAG formalism group is Tree-Local MCTAG (TL-MCTAG) in which all trees from a single elementary tree set are constrained to adjoin or substitute into a single tree in another elementary tree set. Weir (1988) shows that this constraint is sufficient to guarantee that TL-MCTAG has weak generative capacity equivalent to the polynomially parsable TAG.

Recent work on the complexity of several TAG variants has demonstrated indirectly that the universal recognition problem for TL-MCTAG is NP-hard. This result calls into question the practicality of systems that employ TL-MCTAG as the formalism for expressing a natural language grammar. In this paper we present a more fine-grained analysis of the processing complexity of TL-MCTAG. We demonstrate (Section 2) that even under restricted definitions where either the rank or the fan-out of the grammar is bounded, the universal recognition problem is NP-complete.

We define a novel variant of multi-component TAG formalisms that treats the elementary structures as vectors of trees rather than as unordered sets (Section 3). We demonstrate that this variant of the definition of the formalism (the **vector definition**) is consistent with the linguistic applications of the formalism presented in the literature. Universal recognition of the vector definition of TL-MCTAG is NP-complete when both the rank and fan-out are unbounded. However, when the rank is bounded, the universal recognition problem is polynomial in both the length of the input string and the grammar size.

We present a novel parsing algorithm for TL-MCTAG (Section 4) that accommodates both the set and vector definitions of TL-MCTAG. Although no algorithms for parsing TL-MCTAG have previously been published, the standard method for parsing LCFRS-equivalent formalisms can be applied directly to TL-MCTAG to produce a quite inefficient baseline algorithm in which the polynomial degree of the length of the input string depends on the input grammar. We offer an alternative parser for TL-MCTAG in which the polynomial degree of the length of the input string is constant, though the polynomial degree of the grammar size depends on the input grammar. This alternative parsing algorithm is more appealing than the baseline algorithm because it performs universal recognition of TL-MCTAG (vector definition) with constant polynomial degree in both the length of the input string and the grammar size when rank is bounded.

[1] The terms rank and fan-out are defined in Section 2.

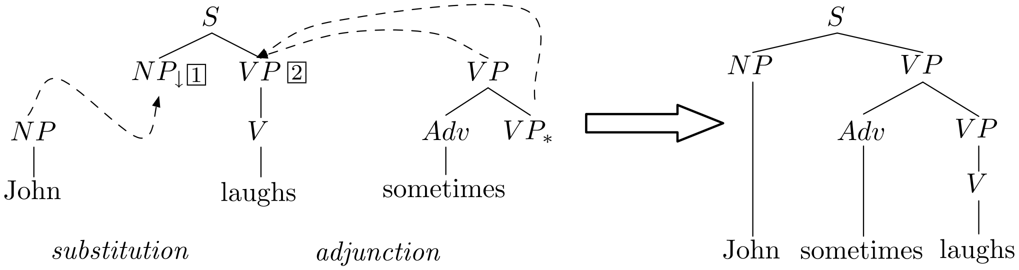


Figure 1
An example of TAG operations **substitution** and **adjunction** used here to model natural language syntax.

It may not be generally desirable to impose an arbitrary rank bound on TL-MCTAGs to be used for linguistic applications. However, it is possible given a TL-MCTAG to optimally minimize the rank of the grammar. In the penultimate section of the paper (Section 5) we offer a novel and efficient algorithm for transforming an arbitrary TL-MCTAG into a strongly equivalent TL-MCTAG where the rank is minimized.

In Section 6, we conclude with a placement of our results in the context of prior literature.

1.2 Technical Background

A tree-adjoining grammar consists of a set of elementary tree structures of arbitrary depth, which are combined by operations of **adjunction** and **substitution**. **Auxiliary trees** are elementary trees in which the root and a frontier node, called the **foot node** and distinguished by the diacritic $*$, are labeled with the same nonterminal A. The adjunction operation entails splicing an auxiliary tree in at an internal node in an elementary tree also labeled with nonterminal A. Trees without a foot node, which serve as a base for derivations and may combine with other trees by substitution, are called **initial trees**. Examples of the adjunction and substitution operations are given in Figure 1. For further background, refer to the survey by Joshi and Schabes (1997).

A TAG derivation can be fully specified by a **derivation tree**, which records how the elementary structures are combined using the TAG operations to form the derived tree. The nodes of the derivation tree are labeled by the names of the elementary trees and the edges are labeled by the addresses at which the child trees substitute or adjoin. In contrast context-free grammars, the derivation and derived trees are distinct.

We depart from the traditional definition in notation only by specifying adjunction sites explicitly with numbered **links** in order to simplify the presentation of the issues raised by multi-component adjunctions. Each link may be used only once in a derivation. Adjunctions may only occur at nodes marked with a link. A numbered link at a single site in a tree specifies that a single adjunction is available at that site. Because we use explicit links, the edges in the derivation tree are labeled with the number of the link used rather than the address at which the operation takes place.

Multiple adjunction refers to permitting an unbounded number of adjunctions to occur at single adjunction site (Vijay-Shanker 1987; Shieber and Schabes 1994). In the standard definition of TAG, multiple adjunction is disallowed in order to avoid spurious ambiguity. In multicomponent TAG, multiple adjunction additionally leads

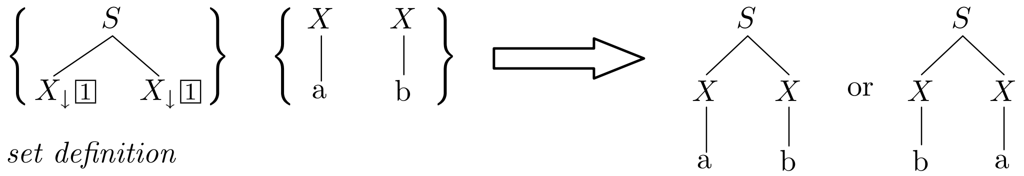


Figure 2
An example of the way in which two tree sets may produce several different derived trees when combined under the standard definition of multicomponent TAG.

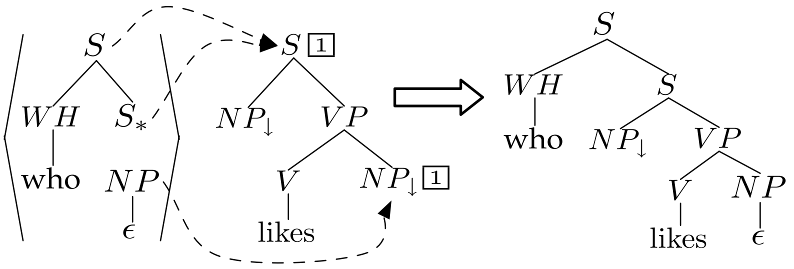


Figure 3
An example TL-MCTAG operation demonstrating the use of TL-MCTAG to model wh-question syntax.

to increased expressivity. Because each available adjunction is explicitly notated with a numbered link, our notation implicitly disallows multiple adjunction but permits a third possibility: bounded multiple adjunction. Bounded multiple adjunction permits the formalism to obtain some of the potential linguistic advantages of allowing multiple adjunction while limiting the increase in expressivity and complexity that results from unbounded multiple adjunction (Nesson and Shieber 2006). The usual constraint of allowing only one adjunction at a given adjunction site may be enforced in our link notation by permitting only one link at a particular link site to be used.

Multicomponent TAG (MCTAG) generalizes TAG by allowing the elementary items to be sets of trees rather than single trees (Joshi and Schabes 1997). The basic operations are the same but all trees in a set must adjoin (or substitute) into another tree set in a single step in the derivation. To allow for multi-component adjunction, a numbered link may appear on two or more nodes in a tree, signifying that the adjoining trees must be members of the same tree set. Any tree in a set may adjoin at any link location if it meets other adjunction or substitution conditions such as a matching node label. Thus a single multicomponent link may give rise to many distinct derived trees even when the link is always used by the same multicomponent tree set. An example is given in Figure 2. This standard definition of multicomponent adjunction we will call the **set definition** for contrast with a variation we introduce in Section 3. A derivation tree for a multicomponent TAG is the same as for TAG except that the nodes are labeled with the names of elementary tree sets.

An MCTAG is **tree-local** if tree sets are required to adjoin within a single elementary tree (Weir 1988). (Using the numbered link notation introduced above for adjunction sites, a tree-local MCTAG (TL-MCTAG) is one in which the domain of locality of the link numbers is a single elementary tree.) An example TL-MCTAG operation is given in Figure 3.

Weir (1988) noted in passing that tree-local MCTAG does not increase the generative capacity of TAG; a combination of well-chosen selective and obligatory adjunction constraints and additions of duplicates of trees to the grammar can produce a weakly equivalent TAG. Alternatively, a feature-based TAG where the features enforce the same constraints may be used. Although the generative capacity of the formalism is not increased, any such conversion from TL-MCTAG to TAG may require an exponential increase in the size of the grammar as we prove in Section 2 below.

2. Complexity

We present several complexity results for TL-MCTAG. Søgaard, Lichte, and Maier (2007) show indirectly that TL-MCTAG membership is NP-hard. For clarity, we present a direct proof here. We then present several novel results demonstrating that the hardness result holds under significant restrictions of the formalism. **Fan-out**, f, measures the number of trees in the largest tree set in the grammar. We show that even when the fan-out is bounded to a maximum of two, the NP-hardness result still holds. The **rank**, r, of a grammar is the largest number of links in any tree in the grammar. We show that when rank is bounded, the NP-hardness result also holds.

A notable aspect of all of the proofs given here is that they do not make use of the additional expressive power provided by the adjunction operation of TAG. Put simply, the trees in the tree sets used in our constructions meet the constraints of Tree Insertion Grammar (TIG), a known context-free–equivalent formalism (Schabes and Waters 1995). As a result, we can conclude that the increase in complexity stems from the multi-component nature of the formalism rather than from the power added by an unconstrained adjunction operation.

2.1 Universal Recognition of TL-MCTAG is NP-Complete

In this section we prove that universal recognition of TL-MCTAG is NP-complete when neither the rank nor the fan-out of the grammar is bounded.

Recall the 3SAT decision problem, which is known to be NP-complete. Let $V = \{v_1, \ldots, v_p\}$ be a set of variables and $C = \{c_1, \ldots, c_n\}$ be a set of clauses; each clause in C is represented by a set of three literals over the alphabet of all literals $L_V = \{v_1, \overline{v}_1, \ldots, v_p, \overline{v}_p\}$. The language 3SAT is defined as the set of all disjunctive formulas over the members of C that are satisfiable.

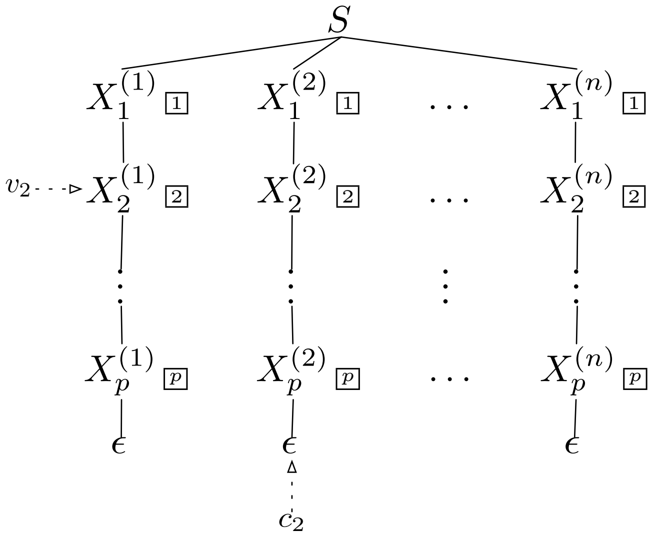
Theorem 1
The universal recognition problem for TL-MCTAG with unbounded rank and fan-out is NP-hard.

Proof
Let $\langle V, C \rangle$ be an arbitrary instance of the 3SAT problem.[2] We use the derivations of the grammar to guess the truth assignments for V and use the tree sets to keep track of the dependencies among different clauses in C. Two tree sets are constructed for each variable, one corresponding to an assignment of **true** to the variable and one

[2] We follow the proof strategy of Satta and Peserico (2005) in this and the following proof.

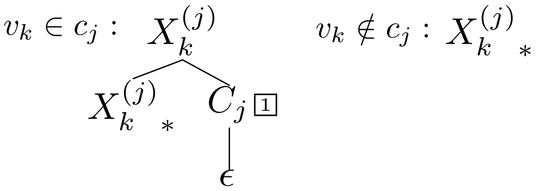
corresponding to an assignment of **false**. The links in the single initial tree permit only one of these two sets to be used. The tree set for a particular truth assignment for a particular variable v_i makes it possible to introduce, by means of another adjunction, terminal symbols taken from the set $\{1, \ldots, n\}$ that correspond to each clause in C that would be satisfied by the given assignment to v_i. In this way, the string $w = 1 \cdots n$ can be generated if and only if all clauses are satisfied by the truth assignment to some variable they contain.

We define a tree-local MCTAG G containing the following tree sets. The initial tree set S contains the single tree:

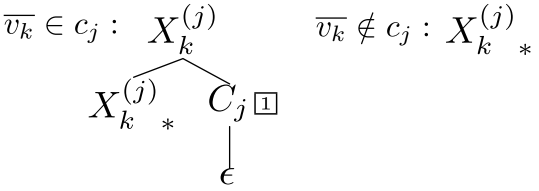


In this tree, the "rows" correspond to the variables and the "columns" to the clauses.

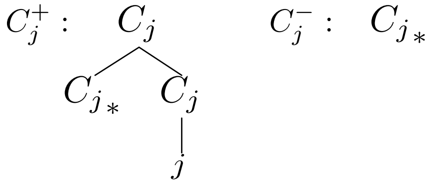
For every variable v_i, $1 \leq i \leq p$, tree set T_i, used when representing an assignment of the value **true** to v_i, contains n trees, one for each clause c_j, $1 \leq j \leq n$, defined as follows:



For every variable v_i, $1 \leq i \leq p$, tree set F_i — used when representing an assignment of the value **false** to v_i — contains n trees, one for each clause c_j, $1 \leq j \leq n$, defined as follows:


$\overline{v_k} \in c_j$:
$X_k^{(j)}$
$X_{k\,*}^{(j)}$
C_j 1
ϵ
$\overline{v_k} \notin c_j$: $X_{k\,*}^{(j)}$

For every clause c_j, $1 \leq j \leq n$, tree sets C_j^+ and C_j^- each contain a single tree as shown below. These trees optionally allow the corresponding clause number terminal symbol to be recognized by an appropriate variable instance.



From the definition of G it directly follows that $w \in L(G)$ implies the existence of a truth-assignment that satisfies C. A satisfying truth assignment can be read directly off of any derivation tree for w. If T_i (resp., F_i) is a child of S in the derivation tree, then v_k is **true** (resp., **false**). The converse can be shown by using a satisfying truth assignment for C to construct a derivation for $w \in G$. $\langle G, w \rangle$ can be constructed in deterministic polynomial time because the number of tree sets in the grammar is $2p + 2n + 1$, the total number of trees in the grammar is bounded by $n(2p + 2n + 1)$, and the length of w is n. All trees in the grammar have constant size except for the initial tree, which has size np. ■

Theorem 2
The universal recognition problem for TL-MCTAG with unbounded rank and fan-out is in NP.

Proof
We show that given an arbitrary TL-MCTAG grammar G and any input string w, the determination of $w \in G$ can be performed in non-deterministic polynomial time.

Note that the collection of elementary tree sets of G that can generate the empty string, $\mathcal{E}$, can be generated in time polynomial in $|G|$ using the standard graph reachability algorithm used for context-free grammars in time polynomial in $|G|$ (Sippu and Soisalon-Soininen 1988).

We begin by showing that given an arbitrary input string w and derivation tree $\mathcal{D}$ for $w \in G$, there must exist a truncated derivation tree for w that has size no larger than $|G| \cdot |w|$. We define a **truncated derivation tree** as a derivation tree in which the children of elementary tree sets in $\mathcal{E}$ are optionally removed.

Consider $\mathcal{D}$. Each node in $\mathcal{D}$ represents an elementary structure of G: a tuple of one or more TAG trees. We call a node n of $\mathcal{D}$ a **non-splitting node** if a single one of its children in the derivation tree, n_i generates the same lexical material from the input string as n itself.[3] We call it a **splitting node** if more than one of its children generate a non-empty part of the portion of the input string generated by n itself or if n itself contributes lexical material. We proceed from the root of $\mathcal{D}$ examining chains of non-splitting nodes. Assume that the root of $\mathcal{D}$ is a non-splitting node. This means that it has a single child node, n_i that generates the lexical material for the entire input string. Its other children all generate the empty string (and therefore must also be members

[3] The child tree tuple n_i may generate the same lexical material in several distinct pieces which are arranged into the string generated by n when the adjunction occurs. Because the adjunction necessarily connects all of these pieces into a single string in a single predetermined way, it does not matter for our proof that the lexical material derived by the child may be in any order before the adjunctions.

of $\mathcal{E}$). We truncate the derivation tree at each child of n other than n_i. We now iterate the process on node n_i. If during the examination of a chain of non-splitting nodes we encounter a node identical to one that we have already seen, we remove the entire cycle from the derivation tree because it is not essential to the derivation. Because all cycles are removed, the longest possible chain of non-splitting nodes we can find before encountering a splitting node or reaching the bottom of the derivation tree is $|G|$.

If a splitting node is encountered, we truncate all child nodes that generate the empty string and then iterate the process of non-splitting node identification on those children that generate lexical material. In the worst case, the process encounters $w - 1$ splitting nodes, each of which may be separated by a chain of non-splitting nodes of maximum length bounded by $|G|$. This process, therefore, produces a truncated derivation tree with size bounded by $|G| \cdot |w|$.

The truncation of the tree at each node that generates the empty string is necessary because the size of the subderivation tree generating the empty string may not be bounded by a polynomial in the size of the grammar. However, the content of the part of the derivation tree used to generate the empty string is not necessary for determining membership of $w \in G$ since we know that each truncated node is a member of $\mathcal{E}$.

To show that TL-MCTAG membership is in NP, we construct a turing machine that will non-deterministically guess a truncated derivation tree of size no larger than $|G| \cdot |w|$. It then checks that the guessed derivation successfully derives w. Because the correctness of the derivation can be checked in linear time, this is sufficient to show that TL-MCTAG membership is in NP. ■

We know from the equivalence of LCFRS and SL-MCTAG (and the rule-to-tree-tuple conversion method used to prove equivalency) (Weir 1988) and the fact that LCFRS membership is PSPACE-complete that SL-MCTAG membership is also PSPACE-complete (Kaji et al. 1992, 1994). Until the above rule it was not known whether TL-MCTAG was in NP. Although the difference in generative capacity between TL-MCTAG and SL-MCTAG is well-known, this proven difference in complexity (assuming NP $\neq$ PSPACE) is novel.

To understand the reason underlying the difference, we note that the bound on the length of non-splitting chains does not hold for set-local MCTAG. In set-local MCTAG a tree tuple may be non-splitting while also performing a permutation of the order of the lexical output generated by its children. Permutation is possible because set-locality allows the tuple of strings generated by a tree tuple to be held separate for an arbitrary number of steps in a derivation. This directly follows the basis of the reasoning of Kaji et al. (1992) in their proof that LCFRS is PSPACE-complete.

2.2 Universal Recognition of TL-MCTAG with Bounded Fan-Out is NP-Complete

The grammar constructed in the proof of Theorem 1 has fan-out n, the number of clauses. However, the hardness result proved above holds even if we restrict tree sets to have at most two elements (TL-MCTAG(2)). We use the postfix (2) to indicate the restriction on the fan-out. The result provided here is as tight as possible. If tree sets are restricted to a maximum size of one (TL-MCTAG(1)), the formalism reduces to TAG and the hardness result does not hold.

Theorem 3
The universal recognition problem for TL-MCTAG(2) with fan-out limited to two and unbounded rank is NP-complete.

Proof
Let $\langle V, C\rangle$ be an arbitrary instance of the 3SAT problem. We define a more complex string $w = w^{(1)}w^{(2)}\cdots w^{(p)}w_c$ where w_c is a representation of C and $w^{(i)}$ controls the truth assignment for the variable v_i, $1 \leq i \leq p$. Then we construct a TL-MCTAG(2) grammar G such that each $w^{(i)}$ can be derived from G in exactly two ways using the left members of tree sets of size 2 that correspond to the variables (and a single initial tree set of size 1). The form of the prefix string enforces the constraint of permitting only two derivations by requiring a strictly alternating string of terminal symbols that can only be generated by the grammar when the truth assignment is stable for a particular variable. The derivation of the prefix string $w^{(1)}w^{(2)}\cdots w^{(p)}$ therefore corresponds to a guess of a truth assignment for V. The right trees from the tree sets derive the components of w_c that are compatible with the guessed truth assignments for $v_1, \ldots, v_p$. Below we explain how $\langle G, w\rangle$ is constructed given an instance of 3SAT $\langle V, C\rangle$.

For every variable v_i, $1 \leq i \leq p$, let $\mathcal{A}_i = \{c_j \mid v_i \in c_j\}$ and $\overline{\mathcal{A}}_i = \{c_j \mid \overline{v}_i \in c_j\}$ be the sets of clauses in which v_i occurs positively and negatively, respectively; let also $m_i = |\mathcal{A}_i| + |\overline{\mathcal{A}}_i|$ be the number of occurrences of the variable v_i. Let $\Sigma' = \{a_i, b_i \mid 1 \leq i \leq p\}$ be an alphabet of not already used symbols; let $w^{(i)}$ (again for $1 \leq i \leq p$) denote a sequence of $m_i + 1$ alternating symbols a_i and b_i such that if m_i is even $w^{(i)} = (a_ib_i)^{m_i/2}a_i$ and if m_i is odd $w^{(i)} = (a_ib_i)^{(m_i+1)/2}$. We define three functions, α, γ, and $\overline{\gamma}$ to aid in the construction. The functions γ and $\overline{\gamma}$ are used to produce pieces of the prefix string and will only produce the correct prefix string for a variable if the truth assignment is consistent within the derivation. The function α is used to produce strings representing the clauses satisfied by a particular truth assignment to a variable. For every variable v_i, $1 \leq i \leq p$, the clauses $\alpha(i,1), \alpha(i,2), \ldots, \alpha(i, |\mathcal{A}_i|)$ are all the clauses in $\mathcal{A}_i$ and the clauses $\alpha(i, |\mathcal{A}_i| + 1), \ldots, \alpha(i, m_i)$ are all the clauses in $\overline{\mathcal{A}}_i$. Further, for every $1 \leq i \leq p$, let $\gamma(i, 1) = a_ib_i$ and let $\gamma(i, h) = a_i$ if h is even and $\gamma(i, h) = b_i$ if h is odd, for $2 \leq h \leq m_i$. For every $1 \leq i \leq p$, let $\overline{\gamma}(i, h) = a_i$ if h is odd, and $\overline{\gamma}(i, h) = b_i$ if h is even for $1 \leq h \leq m_i - 1$ and let $\overline{\gamma}(i, m_i) = a_ib_i$ if m_i is odd and b_ia_i if m_i is even. The crucial property of γ and $\overline{\gamma}$ is that a string $w^{(i)}$ can be parsed either as a sequence of $\gamma(i, \cdot)$ or $\overline{\gamma}(i, \cdot)$ strings, not intermixed elements. The grammar must "commit" to parsing the string one way or the other, corresponding to committing to a value for the variable v_i.

We define a TL-MCTAG(2) G to consist of the tree sets described below. We construct a tree set of length two for each combination of a variable and clause that the variable can satisfy under some truth assignment, filler tree sets of length two that only contribute the string indicating the truth assignment of the variable but no satisfied clause, and a singleton tree set containing only an initial tree rooted in S. The initial tree has $n + 1$ branches with the first branch intended to yield the prefix string $w^{(1)}\cdots w^{(p)}$ and the $(k + 1)$-st branch intended to yield c_k where $1 \leq k \leq n$. Although it is possible to generate strings not of the form of w using this construction, given a pair $\langle G, w\rangle$ where w respects the definition above, we show that $w \in L(G)$ if and only if C is satisfiable.

The initial tree set S contains the single tree pictured in Figure 4.[4] The name of each link in the initial tree set is composed of three indices that indicate the role of the link. The first index, i, corresponds to variable v_i. The second is an index into the series $1 \cdots m_i$ where m_i is defined from v_i as described above. The third index, j, corresponds

[4] Although we permit the presence of multiple links at a single node in the S tree, as noted earlier we follow the usual TAG convention of disallowing multiple adjunction. If one of the links at a node is used, the other links at that node are assumed to be unavailable in the derivation.

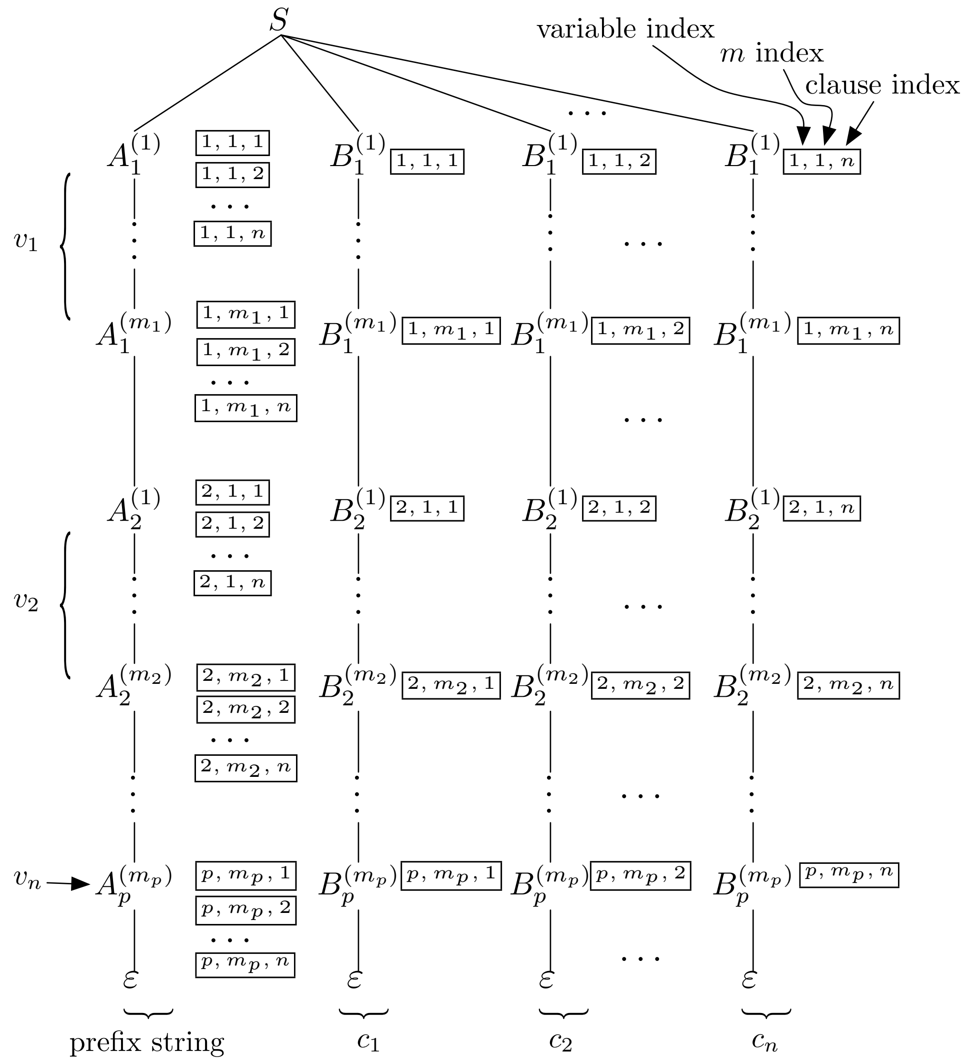
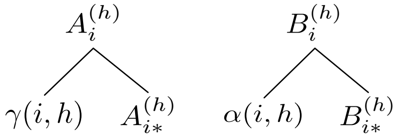


Figure 4
The start tree for TL-MCTAG(2) grammar G. The multiply-indexed link numbers are for clarity only and are treated as simple link names.

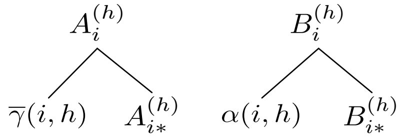
to a clause c_j. The use of multiple indices to name the links is for clarity only. They may be renamed freely.

For every variable v_i, $1 \leq i \leq p$, and index h, $1 \leq h \leq m_i$:

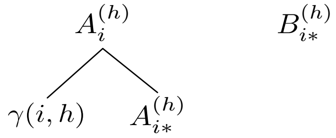
- if $h \leq |\mathcal{A}_i|$, tree set $T_i^{(h)+}$ contains the following two trees:



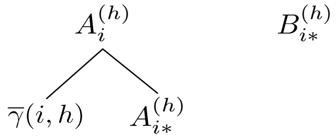
- if $h > |\mathcal{A}_i|$, tree set $F_i^{(h)+}$ contains the the following two trees:



- for all h, tree set $T_i^{(h)-}$ contains the following two trees:



- for all h, tree set $F_i^{(h)-}$ contains the following two trees:



An illustrative example is provided in Figure 5. In this example we demonstrate derivations of two possible satisfying truth assignments for boolean formula $(\overline{x} \vee y \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (y \vee \overline{y} \vee z)$. The truth assignments correspond to whether the T or F tree sets are used in the derivation of the prefix string for a particular variable. As can be seen from the example, the structure of the prefix string enforces that either all T tree sets or all F tree sets are chosen for a particular variable. Each tree set marked with a $+$ is used to satisfy a single clause. Which clause a tree set satisfies can be read off the link number at which it adjoins.

$|G|$ and $|w|$ are polynomially related to p and n. First note that the sum of all the m_i is maximally $3n$ because only $3n$ variables may contribute to the satisfaction of C. There are three tree sets corresponding to each variable and clause that it participates in, for a total of no more than $9pn$ tree sets. With the single initial tree, the total number of tree sets is bounded by $9pn + 1$. Since every tree set other than the initial tree has two trees in it, the total number of trees in the grammar is bounded by $18pn + 1$. The size of all trees other than the initial tree is bounded by a constant. The size of the initial tree is bounded by $3pn$. The length of w is bounded by $4n + p$.

From a derivation of $w \in L(G)$ we can find a truth assignment satisfying C by examining the derivation. If the tree sets $T_i^{(h)+}$ or $T_i^{(h)-}$ are children of S for some i and all h where $1 \leq i \leq p$ and $1 \leq h \leq m_i$, then v_i is true. If the tree sets $F_i^{(h)+}$ or $F_i^{(h)-}$ are children of S for some i and all h where $1 \leq i \leq p$ and $1 \leq h \leq m_i$ then v_i is false. By the construction, if w is of the form described above, for a given variable v_i only two derivations of $w^{(i)}$ will be possible, one in which all tree sets corresponding to that variable are T tree sets and one in which all are F tree sets. Starting from a truth assignment that satisfies C, we can prove $w \in L(G)$ by induction on $|V|$.

That this problem is in NP can be seen from the same reasoning as in the proof of Theorem 2. ■

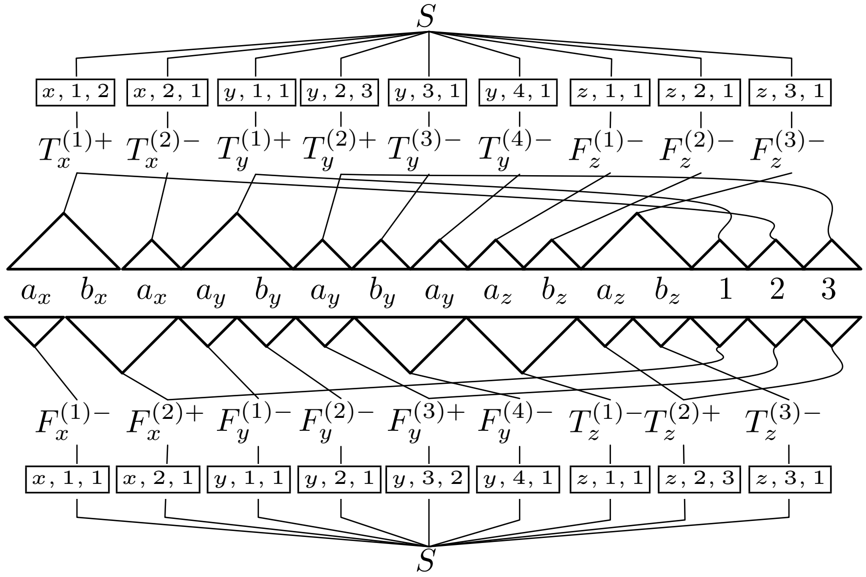


Figure 5
Example derivations of two satisfying assignments for the boolean formula $(\overline{x} \vee y \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (y \vee \overline{y} \vee z)$.

2.3 Universal Recognition of TL-MCTAG with bounded rank is NP-Complete

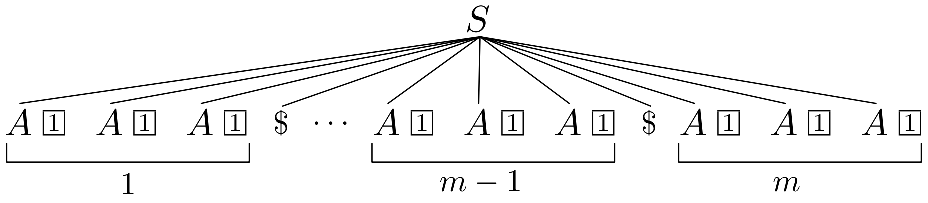
We now show that universal recognition of TL-MCTAG is NP-complete even when the rank is bounded.

We briefly recall here the definition of a decision problem called **3-partition**. Let t and $s_i \leq t$ be positive integers, $1 \leq i \leq 3m$, $m \geq 1$. The language 3PAR is defined as the set of all tuples $\langle s_1, \ldots, s_{3m}, t\rangle$, satisfying the following condition: the multiset $Q = \{s_1, \ldots, s_{3m}\}$ can be partitioned into multisets Q_i, $1 \leq i \leq m$, such that $|Q_i| = 3$ and $\sum_{s \in Q_i} s = t$.

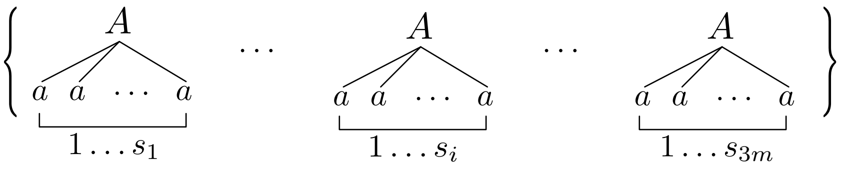
Language 3PAR is strongly NP-complete (Garey and Johnson 1979). This means that 3PAR is NP-complete even in case the integers s_i are all represented in unary notation.

Theorem 4
The universal recognition problem for TL-MCTAG with rank 1 and unbounded fan-out is NP-complete.

Proof
We provide a reduction from 3PAR. Let $\langle s_1, \ldots, s_{3m}, t\rangle$ be an input instance of the 3-partition problem, with all of the integers s_i represented in unary notation. Our target grammar G is defined as follows. We use a set of nonterminal symbols $\{S, A\}$, with S the start symbol. We also use a set of terminal symbols $\{a, \$\}$. G contains two elementary tree sets. The first set has a single elementary tree γ, corresponding to a context-free production of the form $S \rightarrow (AAA\$)^{m-1}AAA$:



Tree γ has a unique link impinging on all of the $3m$ occurrences of nonterminal A. The second (multi)set of G contains elementary trees γ_i , $1 \leq i \leq 3m$. Each γ_i corresponds to a context-free production of the form $A \rightarrow a^{s_i}$:



We also construct a string $w = (a^t\$)^{m-1}a^t$.

If there exists a partition for multiset $Q = \{s_1, \ldots, s_{3m}\}$ satisfying the 3PAR requirement, we can directly construct a derivation for w in G, by sorting the elementary trees in the second set accordingly, and by inserting these trees into the link of the elementary tree γ. Conversely, from any derivation of w in G, we can read off a partition for Q satisfying the requirement for membership in 3PAR for the input instance of the 3-partition problem.

Finally, it is easy to see that G and w can be constructed in linear deterministic time with respect to the size of the input instance of the 3-partition problem.

That this problem is in NP can be seen from the same reasoning as in the proof of Theorem 2. ■

2.4 Universal Recognition of TL-MCTAG with Fixed Input String is NP-Complete

We now show the unusual complexity result that universal recognition of TL-MCTAG is NP-complete even when the input string is fixed. Although it is uncommon to require this result, we rely on it in Section 4 to demonstrate that our parser has better time complexity than the baseline parsing method for TL-MCTAG that we generalize from the standard parsing method for LCFRS-equivalent formalisms.

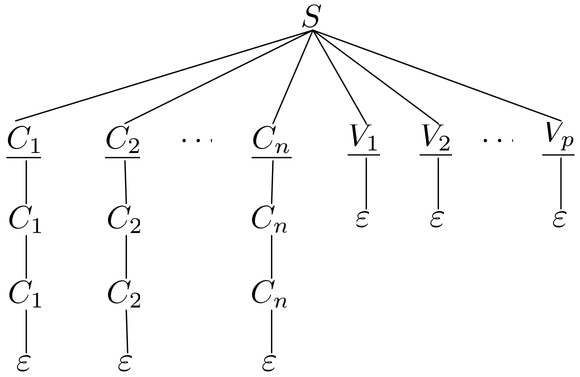
We reduce from a variant of the 3-SAT problem introduced above in which each variable occurs in at most four clauses with no repeats in a clause. This problem was shown to be NP-complete by Tovey (1984).

Theorem 5
Universal recognition of TL-MCTAG is NP-complete when the input string is fixed.

Proof
Let $\langle V, C \rangle$ be an arbitrary instance of the 3SAT problem where each variable occurs in no more than four clauses and does not repeat within a single clause. As in the proof of Theorem 1, we use the derivations of the grammar to guess the truth assignments for V and use the tree sets to keep track of the dependencies among different clauses in C. Two tree sets are constructed for each variable, one corresponding to a true assignment

and one corresponding to a false assignment. The prohibition on multiple adjunction ensures that only one of these two tree sets can be used for each variable. The tree set for a particular truth assignment for a particular variable v_i makes it possible to satisfy obligatory adjunction constraints for nonterminal symbols that represent each of the clauses v_i satisfies in the 3-SAT formula.[5] Additional adjunction sites for each clause provide overflow space in the event that more than one variable satisfies a particular clause. We fix the input string w to be the empty string. None of the trees of the grammar contain any terminal symbols. However, a successful parse of the empty string can only be achieved if all of the obligatory adjunction constraints are satisfied and this occurs if and only if all clauses of the formula are satisfied by the truth assignment to some variable.

We define a tree-local MCTAG G containing the following tree sets. We notate obligatory adjunction constraints by underlining the nodes at which they apply. The initial tree set S contains the single tree:



For every variable v_i, $1 \leq i \leq p$, tree set T_i (resp. F_i), used when representing an assignment of the value **true** (resp. **false**) to v_i, contains at most five trees, one for the variable itself and one for each clause c_j, $1 \leq j \leq n$, such that when v_i is true (resp. false) c_j is satisfied. More formally, tree set T_i contains trees V_{i*} and C_{j*} if and only if $v_i \in c_j$, for $1 \leq j \leq n$. Tree set F_i contains trees V_{i*} and C_{j*} if and only if $\overline{v_i} \in c_j$, for $1 \leq j \leq n$.

Note that the diagram of the initial tree does not show the explicitly notated link locations that we use throughout the paper. We omit the link locations to avoid cluttering the diagram. However, because each variable occurs at most four times in the formula, the total number of links is bounded by pn^{12}.

From the definition of G it directly follows that $\varepsilon \in L(G)$ implies the existence of a truth-assignment that satisfies C. A satisfying truth assignment can be read directly off of any derivation tree for w. If T_i (resp., F_i) is a child of S in the derivation tree, then v_k is **true** (resp., **false**). The converse can be shown by using a satisfying truth assignment for C to construct a derivation for $w \in G$.

$\langle G, w \rangle$ can be constructed in deterministic polynomial time because the number of tree sets in the grammar is $2p + 1$, the total number of trees in the grammar is bounded

[5] Obligatory adjunction constraints are standard in the definition of TAG and MCTAG (Weir 1988). However, obligatory adjunction may be avoided in this proof by creating a larger grammar in which a separate tree set is created for each combination of clauses that may be satisfied by a given variable. Because each variable may appear in no more than four clauses, this increases the number of tree sets in the grammar by 2^4. We leave the details of this alternative proof strategy to the reader.

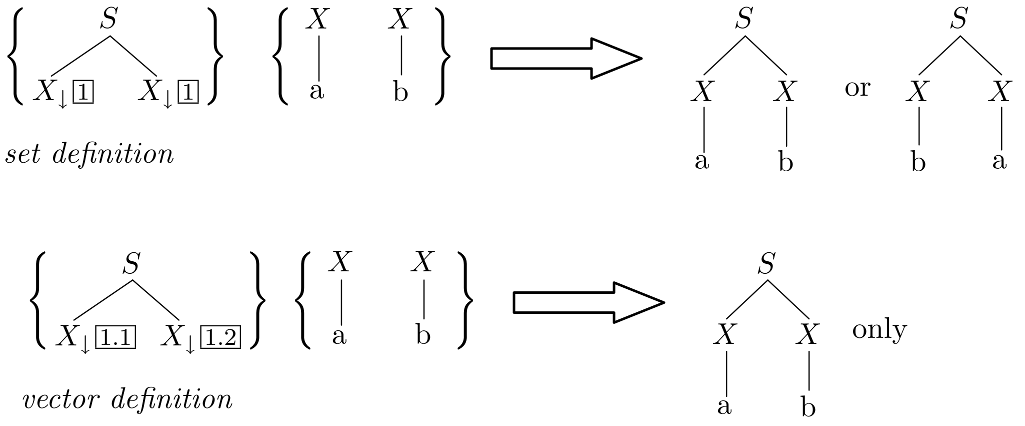


Figure 6
An example contrasting the **set definition** of MCTAG (shown in Figure 2) with the **vector definition**.

by $n(2p + 1)$, and the length of w is 0. All trees in the grammar have constant size except for the initial tree, which has size $3n + p$.

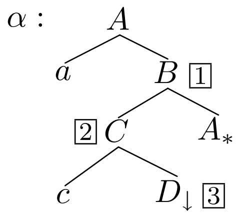
That this problem is in NP can be seen from the same reasoning as in the proof of Theorem 2. ■

3. An Alternative Definition of TL-MCTAG: Tree Vectors

The proof of NP-hardness of TL-MCTAG in the bounded rank case given above (Theorem 4) depends crucially on the treatment of the elementary structures of the TL-MCTAG as unordered sets. In order to produce the satisfying partitions for the 3-partition problem, any tree from the second tree set must be able to adjoin at any location of link $\boxed{1}$ in the first tree set. This is in accordance with the usual definition of multi-component TAG. An alternative definition of multi-component TAG in which the elementary structures are treated as vectors is suggested by the explicit use of numbered links at the available adjunction sites. Under this definition, each location of a link is also given an index and only the tree at that index in a given vector may adjoin at that link location. An example contrasting the two definitions is given in Figure 6.

The dependence of our bounded-rank proof on the set definition of TL-MCTAG does not in itself show that vector definition TL-MCTAG is polynomial in the bounded rank case. We show this constructively in Section 4 by presenting a parser for vector definition TL-MCTAG for which the polynomial degree of both the length of the input string and the grammar size is constant when the rank of the input grammar is bounded.

The difference in complexity between the set and vector definitions of TL-MCTAG makes use of the vector definition an appealing possibility for research using TL-MCTAG for natural language applications. Although all uses of TL-MCTAG in the computational linguistics literature assume the set definition of TL-MCTAG, the linguistic analyses therein do not require the additional flexibility provided by the set definition. In every case of which we are aware, the link locations used by the multicomponent tree sets are completely determined by the constraints imposed by the node labels at the adjunction and substitution sites. As a result, these grammars may be converted



$$\frac{\begin{array}{c}\langle\beta, i+1, m, k, l\rangle\\ \langle\gamma, m, p, q, j\rangle\\ \langle\delta, p+1, \_, \_, q\rangle\end{array}}{\langle\alpha, i, j, k, l\rangle}$$

$$\begin{array}{l} w_{i+1} = a\\ w_{p+1} = c\\ \mathsf{Adj}(\alpha@2 \rhd \boxed{1}, \beta)\\ \mathsf{Adj}(\alpha@2.1 \rhd \boxed{2}, \gamma)\\ \mathsf{Sub}(\alpha@2.12 \rhd \boxed{3}, \delta)\end{array}$$

Figure 7
The deductive rule generated for tree α using the naive TAG parsing method.

to the vector definition without any change in generative capacity or any increase in grammar size but with crucial gains in processing efficiency.

4. Parsing

Although no algorithms for parsing TL-MCTAG have previously been published, the standard method for parsing LCFRS-equivalent formalisms can be applied directly to TL-MCTAG to produce an algorithm for which the polynomial degree of both the length of the input string and the grammar size depends on the input grammar. We offer a novel parser for TL-MCTAG for which the polynomial degree of the length of the input string is constant. For the set definition of TL-MCTAG the polynomial degree of the grammar size depends on both the rank and fan out of the input grammar. For the vector definition of TL-MCTAG the polynomial degree of the grammar size depends on the rank of the input grammar but contains no index of the fan out.

We begin with a brief introduction to TAG parsing before discussing our novel TL-MCTAG parsing algorithm.

4.1 CKY-style TAG Parsing

Following the method of Seki et al. (1991), a naive parser for TAG may be constructed by generating a single inference rule for each tree in the grammar. For a tree containing r links, the rule will have r antecedents with each antecedent item representing a tree that can adjoin at one of the links. Each adjoining tree will cover a span of the input string that can be represented by four indices, indicating the left and right edges of the span and of the subspan that will ultimately be dominated by its foot node. Because the location of the links within the consequent tree is known, the indices in the antecedent items are not entirely independent. An example is given in Figure 7. Observation shows that there will be a worst case of $2(r+1)$ independent indices in a given rule. Since each adjoining tree is independent, there may be $r+1$ different trees represented in a single rule. This results in a time complexity of $O(n^{2(r+1)}|G|^{r+1})$ where n is the length of the input string, $|G|$ is a representation of the grammar size, and r is the rank of the input grammar.

Following Graham, Harrison, and Ruzzo (1980) in their optimization of the Earley parser (Earley 1970), the identifiers of specific trees need not be represented in the items of the parser. Rather the tree identifiers may be replaced by the labels of the root nodes of those trees, effectively bundling items of trees that share a root node label and cover the same span. This modification to the algorithm reduces the time complexity of the parser to $O(n^{2(r+1)}|G|)$.

Item Form:	$\langle \alpha @ a \triangleright \ell, i, j, k, l \rangle$	
Goal:	$\langle \alpha @ \varepsilon \triangleright \_, 0, \_, \_, n \rangle$	$\mathsf{Init}(\alpha)$ $\mathsf{Label}(\alpha @ \varepsilon) = S$
Axioms and Inference Rules:		
Terminal Axiom:	$\langle \alpha @ a \triangleright \_, i, \_, \_, i+1 \rangle$	$\mathsf{Label}(\alpha @ a) = w_{i+1}$
Empty Axiom:	$\langle \alpha @ a \triangleright \_, i, \_, \_, i \rangle$	$\mathsf{Label}(\alpha @ a) = \varepsilon$
Foot Axiom:	$\langle \alpha @ \mathsf{Ft}(\alpha) \triangleright \ell, p, p, q, q \rangle$	$\mathsf{Aux}(\alpha)$ $\mathsf{Link}(\alpha @ \mathsf{Ft}(\alpha)) = \ell$
Unary Complete:	$\dfrac{\langle \alpha @ (a \cdot 1) \triangleright \_, i, j, k, l \rangle}{\langle \alpha @ a \triangleright \ell, i, j, k, l \rangle}$	$\alpha @ (a \cdot 2)$ undefined $\mathsf{Link}(\alpha @ a) = \ell$
Binary Complete:	$\dfrac{\langle \alpha @ (a \cdot 1) \triangleright \_, i, j, k, l \rangle, \langle \alpha @ (a \cdot 2) \triangleright \_, l, j', k', m \rangle}{\langle \alpha @ a \triangleright \ell, i, j \cup j', k \cup k', m \rangle}$	$\mathsf{Link}(\alpha @ a) = \ell$
Adjoin:	$\dfrac{\langle \beta @ \varepsilon \triangleright \_, i, p, q, l \rangle, \langle \alpha @ a \triangleright \boxed{x}, p, j, k, q \rangle}{\langle \alpha @ a \triangleright \_, i, j, k, l \rangle}$	$\mathsf{Adj}(\alpha @ a \triangleright \boxed{x}, \beta)$
No Adjoin:	$\dfrac{\langle \alpha @ a \triangleright \boxed{x}, i, j, k, l \rangle}{\langle \alpha @ a \triangleright \_, i, j, k, l \rangle}$	
Substitute:	$\dfrac{\langle \beta @ \varepsilon \triangleright \_, i, \_, \_, l \rangle}{\langle \alpha @ a \triangleright \_, i, \_, \_, l \rangle}$	$\mathsf{Link}(\alpha @ a) = \ell$ $\mathsf{Subst}(\alpha @ a \triangleright \ell, \beta)$

Figure 8
The CKY algorithm for TAG

We refer to this method of reducing complexity by removing unnecessary information about specific elementary structures from the items of the parser as the GHR optimization. When applied it reduces the time complexity in the grammar size but does not alter the basic form of the time complexity expression. It remains a single term consisting of the product of a polynomial in the input string length and a polynomial in the grammar size. We will return to this observation when examining the complexity of TL-MCTAG parsing.

Shieber, Schabes, and Pereira (1995) and Vijay-Shanker (1987) apply the Cocke-Kasami-Younger (CKY) algorithm first introduced for use with context-free grammars in Chomsky normal form (Kasami 1965; Younger 1967) to the TAG parsing problem to generate parsers with a time complexity of $O(n^6|G|^2)$. The speed up in the parser comes from traversing elementary trees bottom up, handling only one link at a time. As a result, no inference rule needs to maintain information about more than one link at a time. If the GHR optimization is applied, the time complexity is reduced to $O(n^6|G|)$.

In order to clarify the presentation of our TL-MCTAG parser, we briefly review the algorithm of Shieber, Schabes, and Pereira (1995) with minor modification, using the deductive inference rule notation from that paper. As shown in Figure 8, items in CKY-style TAG parsing consist of a node in an elementary tree and the indices that mark the edges of the span dominated by that node. Nodes, notated $\alpha@a \triangleright \ell$, are specified by three pieces of information: the identifier α of the elementary tree the node is in, the Gorn address a of the node in that tree[6], and the link ℓ available at that node if there is one.

Each item has four indices, indicating the left and right edges of the span covered by the node as well as any gap in the node that may be the result of a foot node dominated by the node. Nodes that do not dominate a foot node will have no gap in them, which we indicate by the use of underscores in place of the indices for the gap. To limit the number of inference rules needed, we define the following function $i \cup j$ for combining indices:

$$i \cup j = \begin{cases} i & j = \_ \\ j & i = \_ \\ i & i = j \\ \text{undefined otherwise} \end{cases}$$

The side conditions $\mathsf{Init}(\alpha)$ and $\mathsf{Aux}(\alpha)$ hold if α is an initial tree or an auxiliary tree, respectively. $\mathsf{Label}(\alpha@a)$ specifies the label of the node in tree α at address a. $\mathsf{Ft}(\alpha)$ specifies the address of the foot node of tree α. $\mathsf{Link}(\alpha@a)$ specifies the link available at node $\alpha@a$ if there is one and null (represented as _ in the inference rules) otherwise. $\mathsf{Adj}(\alpha@a \triangleright \ell, \beta)$ holds if ℓ is a link at which tree β may adjoin into tree α at address a. $\mathsf{Subst}(\alpha@a \triangleright \ell, \beta)$ holds if ℓ is a link at which tree β may substitute into tree α at address a. If ℓ is null or the adjunction or substitution is prevented by other constraints such as mismatched node labels, these conditions fail.

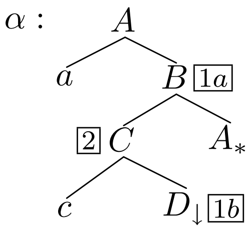
Consistent with the usual definition of TAG, only one link is permitted at a given node. This effectively rules out multiple adjunction. Bounded multiple adjunction may be permitted without affecting the complexity of the parsing algorithm by allowing a list of links at a node. Although it first appears that the introduction of multiple links at a single node could result in an exponential increase in the number of derivations, this is not the case. The link diacritics themselves carry no information about the trees which may adjoin at the associated adjunction site. Any restrictions, such as the requirement of a matching node label, arise from the node itself. As a result, the links are fully interchangeable and serve only as a counter of the number of available adjunctions at a node.[7]

4.2 CKY-Style Tree-Local MCTAG Parsing

As shown in Figure 9, the naive algorithm for parsing TAG may also be applied to TL-MCTAG. The only difference is that each link may have multiple locations within a given tree. Let r and f represent the rank and fan-out of the input grammar, respectively. The time complexity of the naive parser will therefore be $O(n^{2(rf+1)}|G|^{r+1})$. However, the GHR optimization cannot straightforwardly be applied because maintenance of tree locality requires items to carry information about the identities of the specific trees involved rather than just the labels of the root nodes. Theorem 5 addresses the case

[6] A Gorn address uniquely identifies a node within a tree. The Gorn address of the root node is ε. The jth child of the node with address i has address $i \cdot j$.

[7] But note that the finite length of the lists of links is necessary for multiple adjunction to remain benign.



$$\frac{\begin{array}{c}\langle \beta_a, i+1, m, k, l\rangle \\ \langle \gamma, m, p, q, j\rangle \\ \langle \beta_b, p+1, \_, \_, q\rangle\end{array}}{\langle \alpha, i, j, k, l\rangle}$$

$w_{i+1} = a$
$w_{p+1} = c$
$\mathsf{Adj}(\alpha@2 \triangleright \boxed{1a}, \beta_a)$
$\mathsf{Adj}(\alpha@2.1 \triangleright \boxed{2}, \gamma)$
$\mathsf{Sub}(\alpha@2.12 \triangleright \boxed{1b}, \beta_b)$

Figure 9
The deductive rule generated for tree α_i using the naive TL-MCTAG parsing method.

Item Form:

$$\langle \alpha_x@a \triangleright \ell, i, j, k, l, \Lambda\rangle$$

Goal Item:

$$\langle \alpha_1@\varepsilon \triangleright \_, 0, \_, \_, n, \emptyset\rangle$$

$\mathsf{Init}(\alpha_1)$
$\mathsf{Label}(\alpha_1@\varepsilon) = S$
$|\alpha| = 1$

Axioms:

Terminal Axiom

$$\langle \alpha_x@a \triangleright \_, i, \_, \_, i+1, \emptyset\rangle$$

$\mathsf{Label}(\alpha_x@a) = w_{i+1}$

Empty Axiom

$$\langle \alpha_x@a \triangleright \_, i, \_, \_, i, \emptyset\rangle$$

$\mathsf{Label}(\alpha_x@a) = \varepsilon$

Foot Axiom

$$\langle \alpha_x@\mathsf{Ft}(\alpha_x) \triangleright \ell, p, p, q, q, \emptyset\rangle$$

$\mathsf{Aux}(\alpha_x)$
$\mathsf{Link}(\alpha_x@\mathsf{Ft}(\alpha_x)) = \ell$

Figure 10
Modified item form, goal, and axioms for the CKY algorithm for tree-local MCTAG. Inference rules of the algorithm are given in Figure 11.

in which the input string length is 0. Therefore, in this case, any factor in the complexity including the input string length cannot contribute to the overall time complexity. By showing that the problem is NP-complete when the input string length is 0, Theorem 5 demonstrates that there must be some exponential factor or term in the time complexity expression other than the input string length factor. Due to the earlier observation that the GHR optimization does not change the form of the time complexity expression, Theorem 5 therefore shows that the GHR optimization cannot reduce the exponent of the grammar size term to a constant unless $P = NP$. This leaves open the possibility of the existence of an algorithm that is polynomial in the grammar size but has an additional exponential term in the time complexity expression. However, such an algorithm, if it exists, cannot be generated by application of the GHR optimization to the baseline parser.

Inference Rules:

Unary Complete

$$\frac{\langle \alpha_x@(a \cdot 1) \triangleright \_, i, j, k, l, \Lambda \rangle}{\langle \alpha_x@a \triangleright \ell, i, j, k, l, \Lambda \rangle} \qquad \begin{array}{l} \alpha_x@(a \cdot 2) \text{ undefined} \\ \mathsf{Link}(\alpha_x@a) = \ell \end{array}$$

Binary Complete

$$\frac{\langle \alpha_x@(a \cdot 1) \triangleright \_, i, j, k, l, \Lambda_1 \rangle \langle \alpha_x@(a \cdot 2) \triangleright \_, l, j', k', m, \Lambda_2 \rangle}{\langle \alpha_x@a \triangleright \ell, i, j \cup j', k \cup k', m, \Lambda \rangle} \qquad \begin{array}{l} \mathsf{Link}(\alpha_x@a) = \ell \\ \mathsf{Valid}(\Lambda_1 \cup \Lambda_2) \\ \mathsf{Filter}(\Lambda_1 \cup \Lambda_2, \\ \qquad \alpha_x@a \triangleright \ell) = \Lambda \end{array}$$

Adjoin (**set definition**):

$$\frac{\langle \beta_y@\varepsilon \triangleright \_, i, p, q, l, \emptyset \rangle \langle \alpha_x@a \triangleright \boxed{\sigma}_z, p, j, k, q, \Lambda_1 \rangle}{\langle \alpha_x@a \triangleright \_, i, j, k, l, \Lambda \rangle} \qquad \begin{array}{l} \mathsf{Adj}(\alpha_x@a \triangleright \boxed{\sigma}_z, \beta_y) \\ \mathsf{Valid}(\Lambda_1 \cup \{\boxed{\sigma}_z \mapsto \beta_y\}) \\ \mathsf{Filter}(\Lambda_1 \cup \{\boxed{\sigma}_z \mapsto \beta_y\}, \\ \qquad \alpha_x@a \triangleright \_) = \Lambda \end{array}$$

Adjoin (**vector definition**):

$$\frac{\langle \beta_y@\varepsilon \triangleright \_, i, p, q, l, \emptyset \rangle \langle \alpha_x@a \triangleright \boxed{\sigma}_y, p, j, k, q, \Lambda_1 \rangle}{\langle \alpha_x@a \triangleright \_, i, j, k, l, \Lambda \rangle} \qquad \begin{array}{l} \mathsf{Adj}(\alpha_x@a \triangleright \boxed{\sigma}_y, \beta_y) \\ \mathsf{Valid}(\Lambda_1 \cup \{\boxed{\sigma} \mapsto \beta\}) \\ \mathsf{Filter}(\Lambda_1 \cup \{\boxed{\sigma} \mapsto \beta\}, \\ \qquad \alpha_x@a \triangleright \_) = \Lambda \end{array}$$

Substitute (**set definition**):

$$\frac{\langle \beta_y@\varepsilon \triangleright \_, i, \_, \_, l, \emptyset \rangle}{\langle \alpha_x@a \triangleright \_, i, \_, \_, l, \Lambda \rangle} \qquad \begin{array}{l} \mathsf{Link}(\alpha_x@a) = \boxed{\sigma}_z \\ \mathsf{Subst}(\alpha_x@a \triangleright \boxed{\sigma}_z, \beta_y) \\ \mathsf{Filter}(\{\boxed{\sigma}_z \mapsto \beta_y\}, \\ \qquad \alpha_x@a \triangleright \_) = \Lambda \end{array}$$

Substitute (**vector definition**):

$$\frac{\langle \beta_y@\varepsilon \triangleright \_, i, \_, \_, l, \emptyset \rangle}{\langle \alpha_x@a \triangleright \_, i, \_, \_, l, \Lambda \rangle} \qquad \begin{array}{l} \mathsf{Link}(\alpha_x@a) = \boxed{\sigma}_y \\ \mathsf{Subst}(\alpha_x@a \triangleright \boxed{\sigma}_y, \beta_y) \\ \mathsf{Filter}(\{\boxed{\sigma} \mapsto \beta\}, \\ \qquad \alpha_x@a \triangleright \_) = \Lambda \end{array}$$

No Adjoin (**set definition**):

$$\frac{\langle \alpha_x@a \triangleright \boxed{\sigma}_y, i, j, k, l, \Lambda_1 \rangle}{\langle \alpha_x@a \triangleright \_, i, j, k, l, \Lambda \rangle} \qquad \begin{array}{l} \mathsf{Valid}(\Lambda_1 \cup \{\boxed{\sigma}_y \mapsto \mathtt{na}_y\}) \\ \mathsf{Filter}(\Lambda_1 \cup \{\boxed{\sigma}_y \mapsto \mathtt{na}_y\}, \\ \qquad \alpha_x@a \triangleright \_) = \Lambda \end{array}$$

No Adjoin (**vector definition**):

$$\frac{\langle \alpha_x@a \triangleright \boxed{\sigma}_y, i, j, k, l, \Lambda_1 \rangle}{\langle \alpha_x@a \triangleright \_, i, j, k, l, \Lambda \rangle} \qquad \begin{array}{l} \mathsf{Valid}(\Lambda_1 \cup \{\boxed{\sigma} \mapsto \mathtt{na}\}) \\ \mathsf{Filter}(\Lambda_1 \cup \{\boxed{\sigma} \mapsto \mathtt{na}\}, \\ \qquad \alpha_x@a \triangleright \_) = \Lambda \end{array}$$

Figure 11
Modified inference rules for the CKY algorithm for tree-local MCTAG. Alternative Adjoin, Substitute, and No Adjoin rules are given for the set and vector definitions of TL-MCTAG. The item form, goal item and axioms are given in Figure 10.

set definition:
$\mathsf{Valid}(\Lambda)$ holds if for all links $\boxed{\sigma}_i$ and $\boxed{\sigma}_j$ in Λ, $\Lambda(\boxed{\sigma}_i) = \mathit{\Gamma}_x$ and $\Lambda(\boxed{\sigma}_j) = \mathit{\Gamma}_y$ and $x \neq y$ for some tree set $\mathit{\Gamma}$.

vector definition:
$\mathsf{Valid}(\Lambda)$ holds if for all links $\boxed{\sigma 1}$ and $\boxed{\sigma 2}$ in Λ, $\boxed{\sigma 1} \neq \boxed{\sigma 2}$.

Figure 12
Definition of the Valid condition, which ensures that all locations of a link are used by unique trees from the same tree set. Under the set definition there is an entry for each link location and both the identity of the tree set and the uniqueness of the tree from that tree set must be checked. Under the vector definition only the link name and the tree vector identifier are stored because the link locations uniquely select trees from within tree vectors.

We can improve on the naive algorithm by generalizing the CKY TAG parsing algorithm presented above to the TL-MCTAG case. The direct specification of a CKY-style tree-local MCTAG parser is given in Figures 10 and 11. For a tree set or vector α from G, we notate the trees in the set or vector using indices that are indicated as subscripts on the tree set identifier. A tree set or vector α from G with length two will therefore contain trees α_1 and α_2. Under the set definition these indices serve only as a way of differentiating the members of the tree set. Under the vector definition, the index must match the index of the link location where the tree will adjoin.

In order to directly parse tree-local MCTAG, items must keep track of the trees that adjoin at each multicomponent link. We handle this by adding a **link history** to each item. Under the set definition, a link history is an associative array of links notated with indices and tree set identifiers notated with indices to identify a unique tree within the set. Note that because under the set definition a tree may adjoin at any location of a link, the indices of the link and tree set need not match. The axioms introduce empty link histories, indicating that no adjunctions have yet occurred. When an adjunction takes place, the tree identifier of the adjoining tree is associated with the link at which it adjoins. In order for an adjunction to take place at a multicomponent link, the adjoining tree's tree set must be the same as that of any tree identifier already stored for that link. This is enforced by the $\mathsf{Valid}(\Lambda)$ condition (Figure 12) defined on link histories. The $\mathsf{Filter}(\Lambda, \alpha@a \triangleright \ell)$ function removes links that are completely used from the argument link history. An empty link history indicates that tree locality has been enforced for the subtree specified by the item; thus no additional information need be maintained or passed on to later stages of the parse.

For the vector definition, the link histories may be simplified because each location of a link fully specifies which tree from within a set may adjoin there. As a result, the link history is an associative array of links (not annotated with indices) and tree vector identifiers. An example contrasting the link histories for the set and vector definitions is given in Figure 13.

The addition of a link history to each item increases the complexity of the algorithm. The maximum link history length is bounded by the rank of the input grammar, r. Under the set definition, the number of possible values for each element of a link history is on the order of the number of tree sets in the grammar multiplied by the power set of the fan-out: $|G| \cdot 2^f$. Thus, for the set definition, the complexity of the algorithm is $O(n^6 |G|^{r+2} 2^{rf})$. Under the vector definition, the number of possible values for each element of a link history is on the order of the number of tree sets in the grammar. Thus, for the vector definition, the complexity of the algorithm is $O(n^6 |G|^{r+2})$. Note that the

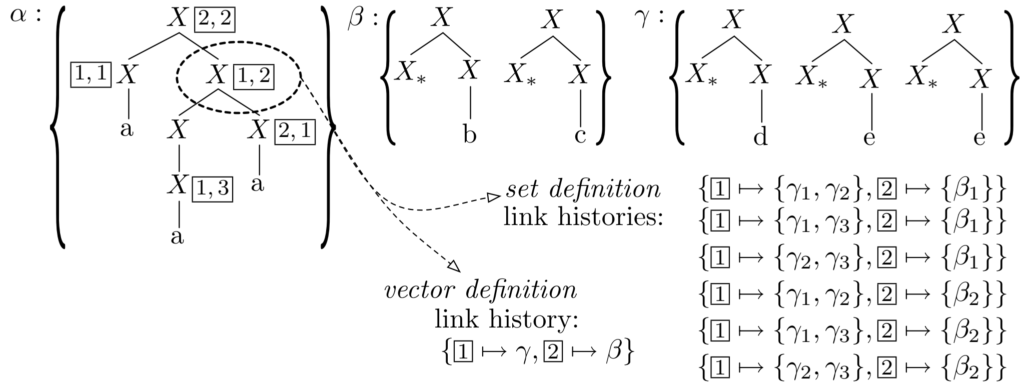

α :
$X\boxed{2,2}$
$\boxed{1,1}X$
$X\boxed{1,2}$
a
X
$X\boxed{2,1}$
$X\boxed{1,3}$
a
a
β :
X X_* X b
X X_* X c
γ :
X X_* X d
X X_* X e
X X_* X e
set definition
link histories:
$\{\boxed{1} \mapsto \{\gamma_1, \gamma_2\}, \boxed{2} \mapsto \{\beta_1\}\}$
$\{\boxed{1} \mapsto \{\gamma_1, \gamma_3\}, \boxed{2} \mapsto \{\beta_1\}\}$
$\{\boxed{1} \mapsto \{\gamma_2, \gamma_3\}, \boxed{2} \mapsto \{\beta_1\}\}$
$\{\boxed{1} \mapsto \{\gamma_1, \gamma_2\}, \boxed{2} \mapsto \{\beta_2\}\}$
$\{\boxed{1} \mapsto \{\gamma_1, \gamma_3\}, \boxed{2} \mapsto \{\beta_2\}\}$
$\{\boxed{1} \mapsto \{\gamma_2, \gamma_3\}, \boxed{2} \mapsto \{\beta_2\}\}$
vector definition
link history:
$\{\boxed{1} \mapsto \gamma, \boxed{2} \mapsto \beta\}$

Figure 13
A sample TL-MCTAG with examples of the possible link histories under the set and vector definitions when the parser reaches the top of the circled node.

variable representing fan-out, f, is present only in the complexity of the set definition. This demonstrates that when rank is bounded, even with unbounded fan-out, parsing the vector definition of TL-MCTAG is polynomial.

Permitting multiple adjunction may be accomplished by a method similar to the one described for the TAG algorithm. Rather than associating each node with at most one link, we permit nodes to be accompanied by a set of links. In contrast to the TAG case, here we must use a set rather than a list in order to allow the expressivity that multiple adjunction can provide. In the TAG case a list is sufficient because the links at a node are fully interchangeable. In the TL-MCTAG case, because the links are defined not just by the node where they appear but by the full set of nodes at which locations of that link appear, the links at a given node are not interchangeable. It must be possible to use them in any order.[8] Because the links can be used in any order, the addition of multiple adjunction adds a factor of 2^r to the time complexity of the parsing algorithm.

5. Link Factorization

The parser presented in the previous section has the advantage of running in polynomial time if the elementary structures of the input TL-MCTAG are defined as vectors and if the rank of the grammar is bounded by some constant. Bounding the rank by a constant might be too strong a limitation in natural language parsing applications, however. Thus, in the general case the running time of our algorithm contains a factor that is an exponential function of the rank of the input grammar. To optimize parsing time, then, we seek a method to "factorize" the elementary trees of the grammar in such a way that the rank is effectively reduced and the set of derived trees is preserved. In this section we present a novel and efficient algorithm for factorizing a TL-MCTAG into a strongly equivalent TL-MCTAG in which rank is minimized across the grammar. Here, strongly equivalent means that the two grammars generate the same set of derived trees

[8] For links that share all locations it is still possible to enforce a strict order over them without compromising expressivity.

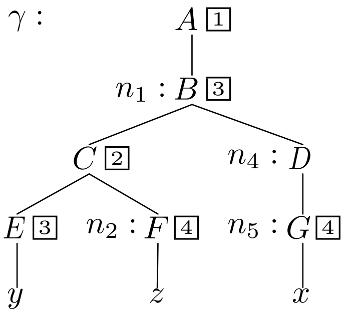


Figure 14
An elementary tree demonstrating fragments, isolation, and maximal nodes. Fragment $\varphi_1 = \alpha(n_1, n_2)$ contains all locations of links $\boxed{2}$ and $\boxed{3}$, because links at the root node of a fragment are contained within that fragment. It does not contain any locations of link $\boxed{4}$, because links at the gap node of a fragment are not contained within that fragment. Because links $\boxed{2}$ and $\boxed{3}$ impinge only on nodes in φ_1 and all other links impinge only on nodes not in φ_1, φ_1 is an isolated fragment. Fragment $\varphi_2 = \alpha(n_4)$ is not an isolated fragment because it contains only one of the link locations of $\boxed{4}$. Note also that n_4 is a maximal node but n_5 is not.

with the exception of permitting a small, reversible transformation that is necessary to ensure that the factorized trees obey the TAG constraint that auxiliary trees must have matching root and foot node labels.

5.1 Preliminaries

Let α be some elementary tree. We write $|\alpha|$ to denote the number of nodes of α. For a link l, we write $|l|$ to denote the number of nodes of l.

For an elementary tree α, we call a **fragment** of α a complete subtree rooted at some node n of α, written $\alpha(n)$, or else a subtree rooted at n with a gap at node n' in its yield, written $\alpha(n, n')$. See Figure 14 for an example. We also use φ to denote a generic fragment with or without a gap node in its yield.

Consider some fragment φ of α. Let N_α be the set of all nodes of α and let N_φ be the set of nodes of φ with the exclusion of the gap node, in case φ has such a node. We say that φ is an **isolated** fragment if φ includes at least one link and no link in α impinges both on nodes in N_φ and on nodes in $N_\alpha - N_\varphi$. See Figure 14 for an example.

Intuitively, we can "excise" an isolated fragment from α without splitting apart the links of α itself, and therefore preserving the tree locality. This operation may also reduce the number of links in α, which is our main goal. The factorization algorithm we present in Subsection 5.2 is based on the detection and factorization of isolated fragments.

Let n be a node from some elementary tree α. We write $\mathsf{lnodes}(n)$ to denote the set of all nodes from fragment $\alpha(n)$ that are part of some link from α. Node n is **maximal** if

- $\mathsf{lnodes}(n) \neq \emptyset$; *and*
- n is either the root node of α or, for its parent node n', we have $\mathsf{lnodes}(n') \neq \mathsf{lnodes}(n)$.

Note that for every node n' of α such that $\mathsf{lnodes}(n') \neq \emptyset$ there is always a unique maximal node n such that $\mathsf{lnodes}(n') = \mathsf{lnodes}(n)$. See Figure 14 for an example. Thus, for the purpose of TL-MCTAG factorization, we can consider only maximal nodes. The first item in the definition of maximal node, stating that a maximal node always dominates (possibly reflexively) some node involved in a link, will often be implicitly used below.

We need to distinguish the nodes in $\mathsf{lnodes}(n)$ depending on their impinging links. Assume that $\{l_1, l_2, \ldots, l_r\}$ is the set of all links occurring in α. For $1 \leq j \leq r$, we write $\mathsf{lnodes}(n, l_j)$ to denote the set of all nodes from fragment $\alpha(n)$ with impinging link l_j. Thus, $\bigcup_{j=1}^{r} \mathsf{lnodes}(n, l_j) = \mathsf{lnodes}(n)$. We associate with each maximal node n of α a **signature** $\sigma(n)$, defined as an integer vector of size r. For each j, $1 \leq j \leq r$, we define

$$\sigma(n)[j] = \begin{cases} \mathsf{lnodes}(n, l_j), & \text{if } 0 < |\mathsf{lnodes}(n, l_j)| < |l_j|; \\ \emptyset, & \text{if } |\mathsf{lnodes}(n, l_j)| = 0 \text{ or} \\ & |\mathsf{lnodes}(n, l_j)| = |l_j|. \end{cases}$$

Observe that, in the above definition, $\sigma(n)[j] = \emptyset$ means that none or all of the nodes of l_j are found within fragment $\alpha(n)$. The **empty** signature, written **0**, is the signature with all of its components set to $\emptyset$.

Consider maximal nodes n_1 and n_2 such that $n_1 \neq n_2$, $\sigma(n_1) \neq \mathbf{0}$ and $\sigma(n_2) \neq \mathbf{0}$. It is not difficult to see that $\sigma(n_1) = \sigma(n_2)$ always implies that one of the two nodes dominates the other. This observation is implicitly used in several places below.

When visiting nodes of α in a path from some leaf node to the root node,[9] one may encounter several maximal nodes having the same non-empty signature. In our factorization algorithm, we need to consider pairs of such nodes that are as close as possible. Consider two maximal nodes n_1 and n_2, $n_1 \neq n_2$, such that n_1 dominates n_2. The ordered pair (n_1, n_2) is called a **minimal pair** if $\sigma(n_1) = \sigma(n_2) \neq \mathbf{0}$ and, for every maximal node n_3 in the path from n_2 to n_1 with $n_3 \neq n_1$ and $n_3 \neq n_2$, we have $\sigma(n_3) \neq \sigma(n_1)$. Consider now a sequence $\langle n_1, n_2, \ldots, n_q \rangle$, $q \geq 2$, of nodes from α. Such a sequence is called a **maximal chain** if each pair (n_{i-1}, n_i) is a minimal pair, $2 \leq i \leq q$, and all nodes n from α with $\sigma(n) = \sigma(n_1)$ are included in the sequence itself.

Notice that two maximal nodes belonging to two different maximal chains must have different signatures, and thus one maximal node cannot belong to more than one maximal chain. We now prove some basic properties of the notions introduced above, that will be used later in the development of our factorization algorithm and in the proof of some of its mathematical properties.

Lemma 1
Let α be an elementary tree and let n, n' be maximal nodes, with n properly dominating n' in (ii) below.

(i) $\sigma(n) = \mathbf{0}$ if and only if $\alpha(n)$ is an isolated fragment;

(ii) $\sigma(n) = \sigma(n')$ if and only if $\alpha(n, n')$ is an isolated fragment.

[9] We view trees as directed graphs with arcs directed from each node to its parent.

Proof
(i). If $\sigma(n) = \mathbf{0}$, then for each link l we have that either all nodes impinged on by l are dominated (possibly reflexively) by n or none of these nodes is dominated by n. Since n is maximal, we further conclude that at least some link l is found within $\alpha(n)$.

Conversely, if $\alpha(n)$ is an isolated fragment then all or none of the nodes impinged on by some link l are dominated by n, and thus $\sigma(n) = \mathbf{0}$.
(ii). Let $\sigma(n) = \sigma(n')$, with n properly dominating n'. For each link l_j, there are two possible cases. First consider the case where $\sigma(n)[j] = \sigma(n')[j] = \emptyset$. In order for this to be true, the link must be in one of three configurations, all of which satisfy the requirement that the locations of l_j must be all inside or all outside of the fragment $\alpha(n_1, n_2)$.

- $\mathsf{lnodes}(n, j) = \emptyset$. In this configuration no one of the nodes on which l_j impinges is dominated by n.
- $|\mathsf{lnodes}(n, j)| = |l_j|$. We distinguish two possible cases.
  - $\mathsf{lnodes}(n', j) = \emptyset$. In this configuration all the nodes on which l_j impinges are within the fragment $\alpha(n_1, n_2)$.
  - $|\mathsf{lnodes}(n', j)| = |l_j|$. In this configuration all the nodes on which l_j impinges are "below" the fragment $\alpha(n, n')$.

Now consider the case where $\sigma(n)[j] = \sigma(n')[j] \neq \emptyset$. The nodes in $\mathsf{lnodes}(n', j)$ are dominated (possibly reflexively) by n' and therefore fall "below" $\alpha(n, n')$. The remaining nodes on which l_j impinges cannot be dominated (possibly reflexively) by n. We thus conclude that no nodes impinged on by l_j occur within the fragment $\alpha(n, n')$.

Assume now that $\alpha(n, n')$ can be isolated. We can use exactly the same arguments as above in the analysis of sets $\mathsf{lnodes}(n, j)$ and $\mathsf{lnodes}(n', j)$, and conclude that $\sigma(n) = \sigma(n')$. ■

The next lemma will be useful later in establishing that the factorization found by our algorithm is optimal.

Lemma 2
Let (n_1, n_2) be some minimal pair. Then

(i) for any node n_3 in the path from n_2 to n_1, $\sigma(n_3) \neq \mathbf{0}$;

(ii) for any minimal pair (n_3, n_4), neither or both of n_3 and n_4 are found in the path from n_2 to n_1.

Proof
(i). Because $\sigma(n_2) \neq \mathbf{0}$, there is some link l_j for which $\sigma(n_2)[j] = \mathsf{lnodes}(n_2, j) \neq \emptyset$. Because n_3 dominates n_2, n_3 dominates the nodes in $\mathsf{lnodes}(n_2, j)$. Therefore, the only way $\sigma(n_3)$ could equal $\mathbf{0}$ is if $|\mathsf{lnodes}(n_3, j)| = |l_j|$. But then $\sigma(n_1)[j] = \emptyset$ because n_1 dominates n_3. This is a contradiction.
(ii). Assume that n_4 is on the path from n_2 to n_1. From the definition of minimal pair, there must exist a link l_k such that $\sigma(n_4)[k] \neq \sigma(n_2)[k]$. By the same reasoning as in the proof of statement (i) above, for any link l_j such that $\sigma(n_2)[j] \neq \emptyset$, we must have $\sigma(n_2)[j] = \sigma(n_4)[j] = \sigma(n_1)[j]$. We thus conclude that $\sigma(n_2)[k] = \emptyset$ and $\sigma(n_4)[k] \neq \emptyset$. Since $\sigma(n_4)[k] = \sigma(n_3)[k] \neq \emptyset$ and $\sigma(n_2)[k] = \sigma(n_1)[k] = \emptyset$, node n_3 must be in the path from n_2 to n_1.

By a similar argument, we can argue that if n_3 is on the path from n_2 to n_1, then node n_4 must be in that path as well. ■

```
Function CHAIN($\alpha$) {$\alpha$ an elementary tree from a TL-MCTAG}
$\mathcal{L} \leftarrow \emptyset$; {associative array mapping signatures into node lists}
for all maximal nodes $n$ from $\alpha$, in top down order do
    if $\sigma(n) \neq \mathbf{0}$ then
        append $n$ to list $\mathcal{L}(\sigma(n))$;
mark as maximal chain each list in $\mathcal{L}$
```

Figure 15
Construction of maximal chains in the factorization algorithm.

5.2 Factorization algorithm

Let G be an input TL-MCTAG grammar. In this subsection we provide a method for the construction of a TL-MCTAG that is strongly equivalent in generative power to G and that has minimal rank. We start with the discussion of some preprocessing of the input.

We annotate each elementary tree α as explained in what follows: We compute sets $\mathsf{lnodes}(n, l_j)$ for all nodes n and all links l_j of α. This can easily be done with a bottom up visit of α, by observing that if an internal node n has children $n_1, n_2, \ldots, n_k$ then $\mathsf{lnodes}(n, l_j) = \bigcup_{i=1}^{k} \mathsf{lnodes}(n_i, l_j) \cup X$, where $X = \emptyset$ if l_j does not impinge on n and $X = \{n\}$ if it does. Using sets $\mathsf{lnodes}(n, l_j)$, we can then mark all nodes n in α that are maximal, and compute the associated signatures $\sigma(n)$.

We also mark all maximal chains within α. This simple procedure is reported in Figure 15. We maintain an associative array with node signatures as entries and node lists as values. We visit all maximal nodes of α in a top down fashion, creating a list for each different signature and appending to such a list all nodes having that signature.

In the algorithm below we excise isolated fragments from each elementary tree α. We now introduce some conventions for doing this. Although it would be possible to excise fragments without the introduction of additional tree structure, we adopt instead two simple tree transformations that preserve auxiliary tree root and foot label matching and result in some simplification of the notation used by the algorithm, particularly in case the root node of a fragment is the same as the gap node of a second fragment within α. A schematic depiction of both transformations is given in Figure 16.

When a fragment $\alpha(n)$ is excised, we leave a copy of the root node n without its impinging links that dominates a fresh node n' with a fresh link indicating obligatory substitution of the excised fragment. The excised fragment consists of $\alpha(n)$ including any links impinging on n, but has a fresh root node immediately dominating n with the same label as n'. After substitution of the excised fragment $\alpha(n)$ back into node n', the original tree can be easily regenerated by removing the fresh node n' and by merging the two copies of node n.

A similar transformation is used to excise a fragment $\alpha(n, n')$ that does not collapse the root node n and the gap node n'. Nodes n and n' of the original tree are not altered, and thus they retain their names. The material between them is replaced with a single new node with a fresh nonterminal symbol and a fresh link. This link indicates the obligatory adjunction for a transformation of the excised fragment. The transformation adds to $\alpha(n, n')$ a new root node and a new gap node, both with the same label, in order to make it technically possible to perform adjunction. We remark that any link impinging on the root node of the excised fragment is by our convention included in the excised fragment, and any link impinging on the gap node is not.

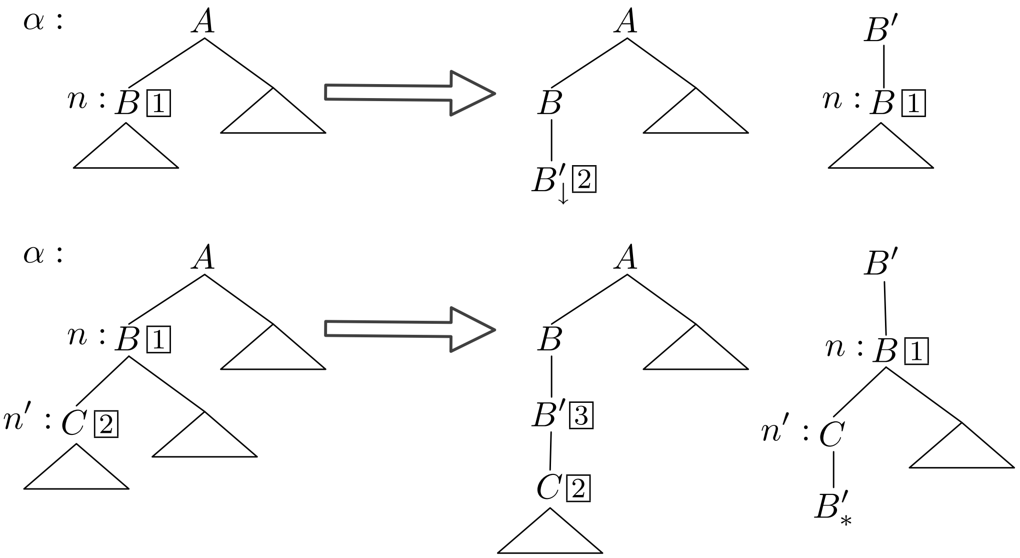


Figure 16
Diagrams of the tree transformations performed when fragments $\alpha(n)$ and $\alpha(n, n')$ are removed.

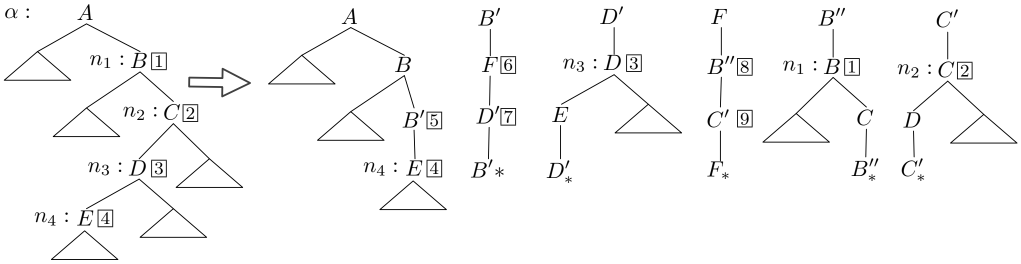


Figure 17
The binarization procedure applied to a maximal chain $c = \langle n_1, n_2, n_3, n_4 \rangle$.

To regenerate the original tree, the excised fragment $\alpha(n, n')$ can be adjoined back into the tree from which it was excised. The new nodes that have been generated in the excision may be removed and the original root and gap nodes may be merged back together retaining any impinging links, respectively. Note that if there was a link on either the root or gap node in the original tree, it is not lost or duplicated in the process.

We need to introduce one more convention for tree excision. Consider a maximal chain $c = \langle n_1, n_2, \ldots, n_q \rangle$ in α, $q \geq 2$. In case $q = 2$, our algorithm processes c by excising a fragment $\alpha(n_1, n_2)$ from α, exactly as explained above. In case $q > 2$, a special processing is required for c. Chain c represents $q - 1$ minimal pairs, corresponding to fragments $\alpha(n_{i-1}, n_i)$, $2 \leq i \leq q$. We do not excise these $q - 1$ fragments one by one, because this would create $q - 1 > 1$ new links within α. We follow instead a procedure that "binarizes" c, as explained below.

Let us recursively define elementary tree α_c as follows.

```
Function FACTORIZE($G$) {$G$ a tree-local MCTAG}
$G' \leftarrow$ tree-local MCTAG with no tree sets;
for all tree sets $\mathit{\Gamma}$ from $G$ do
    for all elementary trees $\alpha$ in $\mathit{\Gamma}$ do
        $\mathcal{A} \leftarrow \emptyset$; {priority queue used as an agenda}
        for all maximal nodes $n$ from $\alpha$ other than the root do
            if $\sigma(n) = \mathbf{0}$ then
                add $n$ to $\mathcal{A}$ with score $\mathsf{links}(n)$;
        for all maximal chains $\langle n_1, \dots, n_q \rangle$ from $\alpha$ do
            add $\langle n_1, \dots, n_q \rangle$ to $\mathcal{A}$ with score $\mathsf{links}(n_1) - \mathsf{links}(n_q)$;
        while $\mathcal{A} \neq \emptyset$ do
            pop from $\mathcal{A}$ item $I$ with smallest score, discarding items with score = 1;
            if $I = n$ then
                $\alpha \leftarrow$ excise $\alpha(n)$ from $\alpha$;
                add to $G'$ tree set $\{\alpha(n)\}$;
            if $I = \langle n_1, n_2 \rangle$ then
                $\alpha \leftarrow$ excise $\alpha(n_1, n_2)$ from $\alpha$;
                add to $G'$ tree set $\{\alpha(n_1, n_2)\}$;
            if $I = \langle n_1, \dots, n_q \rangle$, $q > 2$ then
                $\alpha \leftarrow$ excise $\alpha(n_1, n_q)$ from $\alpha$;
                for all $i$ with $2 \leq i \leq q$ do
                    add to $G'$ tree set $\{\alpha(n_{i-1}, n_i)\}$;
                    add to $G'$ tree set $\{\alpha_c\}$ with $c = \langle n_1, \dots, n_i \rangle$;
    add tree set $\mathit{\Gamma}$ to $G'$
return $G'$
```

Figure 18
The factorization algorithm for tree-local MCTAG.

- In case $q = 3$, α_c is a tree composed of two nodes n and n', with n immediately dominating n'. Node n hosts the (obligatory) adjunction of the fragment $\alpha(n_1, n_2)$ and node n' hosts the (obligatory) adjunction of $\alpha(n_2, n_3)$. Both fragments are transformed as previously discussed.

- In case $q > 3$, α_c is a two node tree specified as above, with n' hosting the (obligatory) adjunction of the transformed fragment $\alpha(n_{q-1}, n_q)$. Node n hosts the adjunction of tree $\alpha_{c'}$, with $c' = \langle n_1, \dots, n_{q-1} \rangle$.

Note that each tree α_c has rank two.

When processing a maximal chain c with $q > 2$, the whole fragment $\alpha(n_1, n_q)$ is excised, using the convention above. This results in a single fresh link added to α. In this case the link refers to the adjunction of a newly created elementary tree α_c, defined as above. An example of the binarization of a maximal chain with $q = 4$ is reported in Figure 17.

We can now discuss the factorization algorithm, reported in Figure 18. For a maximal node n in an elementary tree α, we write $\mathsf{links}(n)$ to denote the number of links from α that are entirely contained in fragment $\alpha(n)$. We process each tree set $\mathit{\Gamma}$ of the source grammar and each elementary tree α in $\mathit{\Gamma}$ as follows.

On a first phase, we add to an agenda $\mathcal{A}$ each maximal node n different from the root of α and such that $\sigma(n) = \mathbf{0}$. We associate this agenda item with the score $\mathsf{links}(n)$. At the same time, each maximal chain $\langle n_1, n_2, \ldots, n_q \rangle$, $q \geq 2$, is added to $\mathcal{A}$, with associated score $\mathsf{links}(n_1) - \mathsf{links}(n_q)$.

On a second phase, we process all items in $\mathcal{A}$, in order of increasing score and ignoring those items that have score of one. If the current item is a maximal node n, we excise the fragment $\alpha(n)$ from α, leaving in place a fresh node with a single node link denoting obligatory substitution. If the current item is a maximal chain of the form $\langle n_1, n_2 \rangle$, we excise from α the fragment $\alpha(n_1, n_2)$, leaving in place a fresh node with a single node link denoting obligatory adjunction of the excised fragment. Finally, if the current item is a maximal chain $c = \langle n_1, \ldots, n_q \rangle$ with $q > 2$, we excise from α the whole fragment $\alpha(n_1, n_q)$, and we apply to the chain the binarization procedure described in this subsection. This results in the addition to the output grammar of fragments $\alpha(n_{i-1}, n_i)$, for $2 \leq i \leq q$, and of newly created elementary tree α_c and elementary trees $\alpha_{c'}$ for each chain c' that is a proper prefix of c. After the processing of all elementary trees in tree set Γ is completed, the resulting version of set Γ is also added to the output grammar.

The discussion of the correctness of the algorithm is reported in the next subsection, along with some other mathematical properties.

5.3 Mathematical properties

We discuss in this subsection some mathematical properties of our factorization algorithm. Let G be the input TL-MCTAG and let G' be the output of the algorithm. We start with the issue of correctness. First, notice that our algorithm stops after a finite number of steps, since the number of possible excisions for G is finite. Assume now that φ and φ' are two isolated fragments within some elementary tree α, and φ' is itself a fragment within φ. It is easy to see that excising φ' from φ results in a new fragment of α that is still an isolated fragment. Using this observation together with Lemma 1, we can then conclude that all fragments that are excised by the algorithm are isolated fragments. This in turn implies that each fragment excision in our algorithm preserves tree locality, and G' is still a TL-MCTAG.

Each fragment that is excised from some source tree must obligatorily be adjoined back into that tree, at the point from which it was removed. Thus, G' is generatively equivalent in the strong sense to G, modulo our trivial tree transformation for the root and the gap nodes. This proves the correctness of our factorization algorithm.

One remark is in order here. Note that we always excise fragments that have at least two links. This can be shown inductively as follows. Consider first the smallest fragments that are excised from some elementary tree α, that is, those fragments that do not contain any other fragment within themselves. These fragments always have at least two links, because of the requirement stated in line 12 in the algorithm. In the inductive case, let φ be some fragment of α from which a second fragment φ' has been already excised in some iteration of the loop at lines from 11 to 23. Fragment φ' is thus replaced by some link l'. Because of the definition of maximal node, φ must contain at least one link l that is not contained in φ'. In case l itself is part of some excised fragment φ'', there will still be some other fresh link replacing φ''. We thus conclude that, when excised, φ always has at least two links. Since excised fragments always have at least two links and since we never consider elementary trees as candidate fragments (line 6), we can conclude that our algorithm always finds a non-trivial factorization of G.

We can now turn to an analysis of the computational complexity of our algorithm. Consider an elementary tree α of G with r links and with a maximum of f

nodes per link. In the preprocessing phase of the algorithm, the computation of sets $\mathsf{lnodes}(n, l_j)$ can be carried out in time $\mathcal{O}(|\alpha| \cdot r \cdot f)$. To see this, notice that there are no more than $|\alpha| \cdot r$ sets $\mathsf{lnodes}(n, l_j)$. Furthermore, we have $|\mathsf{lnodes}(n, l_j)| \leq f$, and each node in $\mathsf{lnodes}(n, l_j)$ is processed in constant time through the union operator, when constructing set $\mathsf{lnodes}(n', l_j)$ for the parent node n' of n. Clearly, $\mathcal{O}(|\alpha| \cdot r \cdot f)$ is also a time upper bound for the computation of quantities $\sigma(n)$ and $\mathsf{links}(n)$ for all nodes in α, and for extracting a list of the maximal nodes therein as well.[10]

In what follows, we will need to compare signatures of different nodes for equality. Despite the fact that each signature has r elements, and each element of a signature is a set with $\mathcal{O}(|f|)$ elements, there are at most $|\alpha|$ different signatures. We can therefore use an atomic symbol to name each signature (perfect hashing). In this way, signatures can be compared in constant time.

The marking of all maximal chains within α, as specified by the algorithm in Figure 15, can be implemented in time $\mathcal{O}(|\alpha|)$. This is done by encoding the associative array $\mathcal{L}$ in the algorithm through a one-dimensional array indexed by signature names. Each element of the array points to a linked list of nodes, representing a maximal chain.

We now analyze the running time of the factorization function in Figure 18. Let us first consider a single elementary tree α. We implement the priority queue $\mathcal{A}$ through a heap data structure. The loops at lines 6 and 9 run in time $\mathcal{O}(|\alpha| \cdot \log(|\alpha|))$: this is the standard result for populating a heap; see for instance (Cormen et al. 2001). At each iteration of the while loop at lines 11 to 23, we extract some fragment $\alpha(n)$ or $\alpha(n_1, n_q)$. The processing of each such fragment φ takes an amount of time $\mathcal{O}(|\varphi|)$, where $|\varphi|$ is the number of nodes of φ. In such an iteration, α needs to be re-edited into a new elementary tree with a number of nodes $|\alpha| - |\varphi| + c$, where $c \leq 3$ is a constant that depends on the specific transformation in Figure 16 that was applied in the excision of the fragment tree. Nonetheless, if a suitable representation is maintained for α, making use of nodes and pointers, the re-editing of α can be done in constant time. Then a single iteration of the while loop takes time $\mathcal{O}(|\varphi|)$, with φ the excised fragment. We can then conclude that all iterations of the while loop take an amount of time $\mathcal{O}(|\alpha| \cdot \log(|\alpha|))$.

Now let α_M be the elementary tree of G with largest size, and let r_G and f_G be the rank and fan-out of G, respectively. Putting everything together, the total running time of the factorization algorithm is $\mathcal{O}(|G| \cdot (r_G \cdot f_G + \log(|\alpha_M|)))$, where $|G|$, the size of the input grammar, is defined as the sum of terms $|\alpha|$ for all elementary trees α of G. Since we always have $f_G \leq |\alpha_M|$, the previous upper bound can be rewritten as $\mathcal{O}(|G| \cdot |\alpha_M| \cdot r_G)$.

A special case is worth discussing here. If the maximum number of links impinging on a node of our elementary trees is bounded by some constant, we have $r_G \cdot f_G = \mathcal{O}(|\alpha_M|)$. In this case, the above bound reduces to $\mathcal{O}(|G| \cdot |\alpha_M|)$. The constant bound on the number of links impinging on the nodes of a grammar holds for all of the grammars we have exploited in Section 2.

We now argue that our algorithm provides the factorization G' of G with the smallest possible rank, under the assumption that G and G' are strongly equivalent, that is, they generate the same derived trees.

A factorization f of G is called **maximal** if no one of its fragments has a smaller isolated fragment within itself. We start by observing that the factorization of G found

[10] We remark here that a further improvement in efficiency could be achieved by replacing the sets of nodes in a signature with the single node that is the least common ancestor of the set of nodes. However, using the set of nodes substantially improves the clarity of the presentation of the algorithm, so we do not pursue this optimization here.

by our algorithm is maximal. To see this, consider the excision by our algorithm of a maximal chain $\langle n_1, \ldots, n_q \rangle$ within an elementary tree α. This item is added to the priority heap at line 10, with a score of $\mathsf{links}(n_1) - \mathsf{links}(n_q)$. The score is the number of links found in fragment $\alpha(n_1, n_q)$, with the exclusion of the links at the gap node n_q. The chain is then factorized into fragments $\alpha(n_{i-1}, n_i)$, for each i with $2 \leq i \leq q$. Assume that some fragment $\alpha(n_{i-1}, n_i)$ contains in turn a maximal chain $\langle n'_1, \ldots, n'_{q'} \rangle$ or else an isolated fragment of the form $\alpha(n')$. In the first case we have $\mathsf{links}(n'_1) - \mathsf{links}(n'_{q'}) < \mathsf{links}(n_1) - \mathsf{links}(n_q)$ and in the second case we have $\mathsf{links}(n') < \mathsf{links}(n_1) - \mathsf{links}(n_q)$. Thus the smaller chain or fragment is processed earlier than our maximal chain, and by the time our maximal chain is processed, the smaller chain or fragment has already been excised. A similar argument shows that the excision by our algorithm of an isolated fragment of the form $\alpha(n)$ happens after the excision of any maximal chain or fragment included within $\alpha(n)$ itself.

We now show that the maximal factorization of G is unique. Let φ and φ' be two isolated fragments of some elementary tree α. We say that φ and φ' **partially overlap** if the set of nodes shared by φ and φ' is not empty and is a proper subset of the nodes of both fragments. It is not difficult to see that if φ and φ' partially overlap, then at least one among φ and φ' must have the form $\alpha(n_1, n_2)$.

Without any loss of generality, we assume that the elementary trees of G are always factorized at their maximal nodes, as discussed in Subsection 5.1. Let us assume that f and f' are two distinguishable maximal factorizations of G. Since no fragment of one factorization can be a sub-fragment of some fragment of the other factorization, there must be some fragment φ of f and some fragment φ' of f' such that f and f' partially overlap.

Assume that φ has the form $\alpha(n_1)$. Then φ' must have the form $\alpha(n_2, n_3)$, and n_1 must be in the path from n_3 to n_2. Since φ' is as small as possible, (n_2, n_3) must be a minimal pair. We have then established a violation of Lemma 2(i). Assume now that φ has the form $\alpha(n_1, n_2)$. Again, (n_1, n_2) must be a minimal pair. If φ' has the form $\alpha(n_3)$, the above argument applies again, resulting in a violation of Lemma 2(i). If φ' has the form $\alpha(n_3, n_4)$, then (n_3, n_4) must be a minimal pair. Furthermore, n_1, n_2, n_3 and n_4 must all be on the same path within α, with n_1, n_2 in alternation with n_3, n_4. This establishes a violation of Lemma 2(ii). The assumption that f and f' partially overlap then leads to a contradiction, and we must conclude that the maximal factorization of G is unique.

We can also use the above argument against the existence of overlapping fragments to show that any factorization f of G other than the unique maximal factorization f_M must be coarser than f_M, meaning that each fragment φ of f is a also a fragment of f_M, or else φ can be represented as a combination of the fragments of f_M (through substitution and adjunction). This means that no factorization of G can have rank smaller than the rank of the maximal factorization f_M. We conclude that our algorithm is optimal.

The above discussion on the optimality of the factorization algorithm crucially assumes strong equivalence with the source TL-MCTAG G. Of course there might be TL-MCTAGs that are weakly equivalent to G, that is, they generate the same language, and have rank strictly smaller than the rank of G'. However, finding such structurally different grammars is a task that seems to require techniques quite different from the factorization techniques we have developed in this section. Furthermore, the task might be computationally unfeasible, considering the fact that the weak equivalence problem for TL-MCTAG is undecidable. (Such a problem is undecidable even for context-free grammars).

We remark here that if we are allowed to change G by recasting its elementary trees in some suitable way, we might still be able to further reduce the rank with respect to the algorithm we have presented in this section. In this case the output grammar would not preserve the derived trees, that is, we lose the strong equivalence, but still retain the derivation trees unaltered. One such case arises when the input TL-MCTAG is not in binary form, that is, some nodes have more than two children. If we allow binarization of the elementary trees of the source grammar, then we might be able to isolate sets of links that could not be factorized in the source grammar itself. Currently, the definition of fragment does not allow splitting apart a subset of the children of a given node from the remaining ones. However, the number of binarizations of an elementary tree is not bounded by a polynomial function of the size of the tree itself, and choosing the binarization that leads to optimal rank reduction is a challenging problem. This new optimization task has the flavor of other unsupervised learning problems from the literature on machine learning and grammar induction that are computationally hard. We leave this problem for future work.

A second case arises when multiple links impinge on the same node of an elementary tree. As presented, the factorization algorithm is designed to handle grammars in which multiple adjunction is permitted. However, if multiple adjunction is disallowed and the grammar contains trees in which multiple links impinge on the same node, the use of one link at a node will disqualify any other impinging links from use. This opens up the possibility of further reducing the rank of the grammar by producing tree sets that do not contain any nodes on which multiple links impinge. This can be accomplished by performing a first pass grammar transformation in which a copy of each elementary tree set is added to the grammar for each distinct, maximal, non-conflicting set of links appearing in the tree set. This transformation in itself may result in a reduction of the rank of the source grammar. The factorization algorithm can then be applied on the new grammar. However, if the elementary trees in the source grammar contain clusters of links that are mutually overlapping, the suggested transformation may blow up the size of the input grammar in a way that is not bounded by any polynomial function.

6. Conclusion

This paper explores the complexity of TL-MCTAG, showing that recognition is NP-complete under a range of interesting restrictions. It then provides a parsing algorithm that performs better than the extrapolation of the standard multiple CFG parsing method to TL-MCTAG. As shown by our proofs, the difficulty in parsing TL-MCTAG stems from the rank of the input grammar. We offer a novel and efficient algorithm for minimizing the rank of the input grammar while preserving its strong generative capacity.

Our work on TL-MCTAG complexity bears comparison to that of several others. Kallmeyer (2007) provides a clear and insightful breakdown of the different characteristics of MCTAG-variant grammars and the effect of these characteristics on expressivity and complexity. That work clarifies the definitions of MCTAG variants and the relationship between them rather than presenting new complexity results. However, it suggests the possibility of proving results such as ours in its assertion that after a standard TAG parse a check of whether particular trees belong to the same tree set cannot be performed in polynomial time. Kallmeyer (2007) also addresses the problem of parsing MCTAG, although not specifically for TL-MCTAG. The method proposed differs from ours in that MCTAGs are parsed first as a standard TAG with any conditions on tree or set locality checked on the derivation forest as a second step. No specific algorithm is presented for performing the check of tree-locality on a TAG derivation forest, so it is difficult to directly compare the methods. However, that method cannot take advantage of the gains in efficiency produced by discarding inappropriate partial parses at the time that they are first considered. Aside from Kallmeyer's work, little attention has been paid to the problem of directly parsing TL-MCTAG.

Søgaard, Lichte, and Maier (2007) present several proofs regarding the complexity of the recognition problem for some linguistically motivated extensions of TAG that are similar to TL-MCTAG. Their work shows NP-hardness of the recognition problem for these variants and, as an indirect result, also demonstrates the NP-hardness of TL-MCTAG recognition. This work differs from ours in that it does not directly show the NP-hardness of TL-MCTAG recognition and does not further locate and constrain the source of the NP-hardness of the problem to the rank of the input grammar nor does it provide mitigation through rank reduction of the grammar.

Our work on TL-MCTAG factorization is thematically though not formally related to the body of work on induction of TAG grammars from a treebank exemplified by Chen and Shanker (2004). The factorization performed in their work is done on the basis of syntactic constraints rather than with the goal of reducing complexity. Working from a treebank of actual natural language sentences, their work does not have the benefit of explicitly labeled adjunction sites but rather must attempt to reconstruct a derivation from complete derived trees.

The factorization problem we address is more closely related to work on factorizing synchronous CFG (Gildea, Satta, and Zhang 2006; Zhang and Gildea 2007) and on factorizing synchronous TAG (Nesson, Satta, and Shieber 2008). Synchronous grammars are a special case of multicomponent grammars, so the problems are quite similar to the TL-MCTAG factorization problem. However, synchronous grammars are fundamentally set-local rather than tree-local formalisms, which in some cases simplifies their analysis. In the case of CFG, the problem reduces to one of identifying problematic permutations of non-terminals (Zhang and Gildea 2007) and can be done efficiently by using a sorting algorithm to binarize any non-problematic permutations until only the intractable correspondences remain (Gildea, Satta, and Zhang 2006). This method is unavailable in the TAG case because the elementary structures may have depth greater than one and therefore the concept of adjacency relied upon in their work is inapplicable. The factorization algorithm of Nesson, Satta, and Shieber (2008) is the most closely related to this one but is not directly applicable to TL-MCTAG because each link is presumed to have exactly two locations and all adjunctions occur in a set-local rather than tree-local manner.

In conclusion, this work falls in an active line of research into efficient processing of multicomponent and synchronous formalisms that appear computationally intractable but have desirable characteristics for meeting the expressive needs of natural language. It presents novel complexity results and algorithms for TL-MCTAG, a widely known and used formalism in computational linguistics that may be applied more effectively in natural-language processing with the introduction of algorithms to process it as efficiently as possible.

References

Chen, John and Vijay K. Shanker. 2004. Automated extraction of TAGs from the penn treebank. *New developments in parsing technology*, pages 73–89.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms*. The MIT Press, 2nd edition.

Earley, J. 1970. *An efficient context-free parsing algorithm*. Ph.D. thesis, University of California, Berkeley, California.

Garey, M. R. and D. S. Johnson. 1979. *Computers and Intractability*. Freeman and Co., New York, NY.

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

Graham, S.L., M.A. Harrison, and W.L. Ruzzo. 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2:415–462.

Han, Chung-Hye. 2006. Pied-piping in relative clauses: Syntax and compositional semantics based on synchronous tree adjoining grammar. In *Proceedings of the 8th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 8)*, pages 41–48, Sydney, Australia.

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

Kaji, Yuichi, Ryuchi Nakanishi, Hiroyuki Seki, and Tadao Kasami. 1992. The universal recognition problems for multiple context-free grammars and for linear context-free rewriting systems. *IEICE Transactions on Information and Systems*, E75-D(1):78–88.

Kaji, Yuichi, Ryuchi Nakanishi, Hiroyuki Seki, and Tadao Kasami. 1994. The computational complexity of the universal recognition problem for parallel multiple context-free grammars. *Computational Intelligence*, 10(4):440–452.

Kallmeyer, Laura. 2007. A declarative characterization of different types of multicomponent tree adjoining grammars. In Andreas Witt Georg Rehm and Lothar Lemnitzer, editors, *Datenstrukturen für linguistische Ressourcen und ihre Anwendungen*, pages 111–120.

Kallmeyer, Laura and Aravind K. Joshi. 2003. Factoring predicate argument and scope semantics: Underspecified semantics with LTAG. *Research on Language and Computation*, 1:3–58.

Kallmeyer, Laura and Maribel Romero. 2007. Reflexives and reciprocals in ltag. In Harry Bunt Jeroen Geertzen, Elias Thijsse and Amanda Schiffrin, editors, *Proceedings of the Seventh International Workshop on Computational Semantics ICWS-7*, pages 271–282, Tilburg, January.

Kasami, T. 1965. An efficient recognition and syntax algorithm for context-free languages. Technical Report AF-CRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.

Nesson, Rebecca, Giorgio Satta, and Stuart Shieber. 2008. Optimal k-arization of synchronous tree-adjoining grammar. In *the Association for Computational Linguistics (ACL-2008)*, Columbus, OH, June.

Nesson, Rebecca and Stuart M. Shieber. 2006. Simpler TAG semantics through synchronization. In *Proceedings of the 11th Conference on Formal Grammar*, Malaga, Spain, 29–30 July.

Nesson, Rebecca and Stuart M. Shieber. 2007. Extraction phenomena in synchronous TAG syntax and semantics. In *Proceedings of Syntax and Structure in Statistical Translation (SSST)*, Rochester, NY, April.

Satta, Giorgio 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 (HLT05/EMNLP05)*, pages 803–810, Vancouver, British Columbia.

Schabes, Yves and Richard C. Waters. 1995. Tree insertion grammar: A cubic-time parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–513, December.

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

Shieber, Stuart M. and Yves Schabes. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124.

Shieber, Stuart M., Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36, July–August. Also available as cmp-lg/9404008.

Sippu, S. and E. Soisalon-Soininen. 1988. *Parsing Theory: Languages and Parsing*. Springer-Verlag, Berlin, Germany.

Søgaard, Anders, Timm Lichte, and Wolfgang Maier. 2007. On the complexity of linguistically motivated extensions of tree-adjoining grammar. In *Recent Advances in Natural Language Processing 2007*.

Tovey, C. A. 1984. A simplified NP-complete satisfiablity problem. *Discrete Applied Mathematics*, 8(1):85–90.

Vijay-Shanker, K. 1987. A study of tree-adjoining grammars. PhD Thesis, Department of Computer and Information Science, University of Pennsylvania.

Weir, David. 1988. Characterizing mildly context-sensitive grammar formalisms. PhD Thesis, Department of Computer and Information Science, University of Pennsylvania.

Younger, D.H. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

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)*.