Optimal k-arization of synchronous tree-adjoining grammar

Rebecca Nesson, Giorgio Satta, and Stuart M. Shieber. Optimal k-arization of synchronous tree-adjoining grammar. In *Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 9)*, Tübingen, Germany, 7-8 June 2008.

Optimal k-arization of Synchronous Tree-Adjoining Grammar

Rebecca Nesson
School of Engineering
and Applied Sciences
Harvard University
Cambridge, MA 02138
nesson@seas.harvard.edu

Giorgio Satta
Department of
Information Engineering
University of Padua
I-35131 Padova, Italy
satta@dei.unipd.it

Stuart M. Shieber
School of Engineering
and Applied Sciences
Harvard University
Cambridge, MA 02138
shieber@seas.harvard.edu

Abstract

Synchronous Tree-Adjoining Grammar (STAG) is a promising formalism for syntax-aware machine translation and simultaneous computation of natural-language syntax and semantics. Current research in both of these areas is actively pursuing its incorporation. However, STAG parsing is known to be NP-hard due to the potential for intertwined correspondences between the linked nonterminal symbols in the elementary structures. Given a particular grammar, the polynomial degree of efficient STAG parsing algorithms depends directly on the *rank* of the grammar: the maximum number of correspondences that appear within a single elementary structure. In this paper we present a compile-time algorithm for transforming a STAG into a strongly-equivalent STAG that optimally minimizes the rank, k, across the grammar. The algorithm performs in $\mathcal{O}(|G| + |Y| \cdot L_G^3)$ time where L_G is the maximum number of links in any single synchronous tree pair in the grammar and Y is the set of synchronous tree pairs of G.

1 Introduction

Tree-adjoining grammar is a widely used formalism in natural-language processing due to its mildly-context-sensitive expressivity, its ability to naturally capture natural-language argument substitution (via its substitution operation) and optional modification (via its adjunction operation), and the existence of efficient algorithms for processing it. Recently, the desire to incorporate syntax-awareness into machine translation systems has generated interest in the application of synchronous tree-adjoining grammar (STAG) to this problem (Nesson, Shieber, and Rush, 2006; Chiang and Rambow, 2006). In a parallel development, interest in incorporating semantic computation into the TAG framework has led to the use of STAG for this purpose (Nesson and Shieber, 2007; Han, 2006b; Han, 2006a; Nesson and Shieber, 2006). Although STAG does not increase the expressivity of the underlying formalisms (Shieber, 1994), STAG parsing is known to be NP-hard due to the potential for intertwined correspondences between the linked nonterminal symbols in the elementary structures (Satta, 1992; Weir, 1988). Without efficient algorithms for processing it, its potential for use in machine translation and TAG semantics systems is limited.

Given a particular grammar, the polynomial degree of efficient STAG parsing algorithms depends directly on the *rank* of the grammar: the maximum number of correspondences that appear within a single elementary structure. This is illustrated by the tree pairs given in Figure 1 in which no two numbered links may be isolated. (By "isolated", we mean that the links can be contained in a fragment of the tree that contains no other links and dominates only one branch not contained in the fragment. A precise definition is given in section 3.)

An analogous problem has long been known to exist for synchronous context-free grammars (SCFG) (Aho and Ullman, 1969). The task of producing efficient parsers for SCFG has recently been addressed by binarization or k-arization of SCFG grammars that produce equivalent grammars in which the rank, k, has been minimized (Zhang

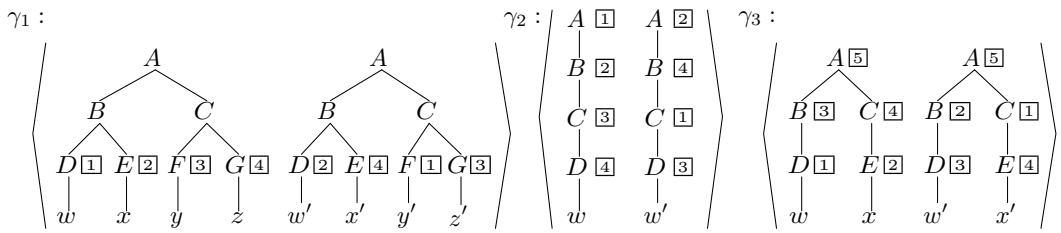

Figure 1: Example of intertwined links that cannot be binarized. No two links can be isolated in both trees in a tree pair. Note that in tree pair γ_1, any set of three links may be isolated while in tree pair γ_2, no group of fewer than four links may be isolated. In γ_3 no group of links smaller than four may be isolated.

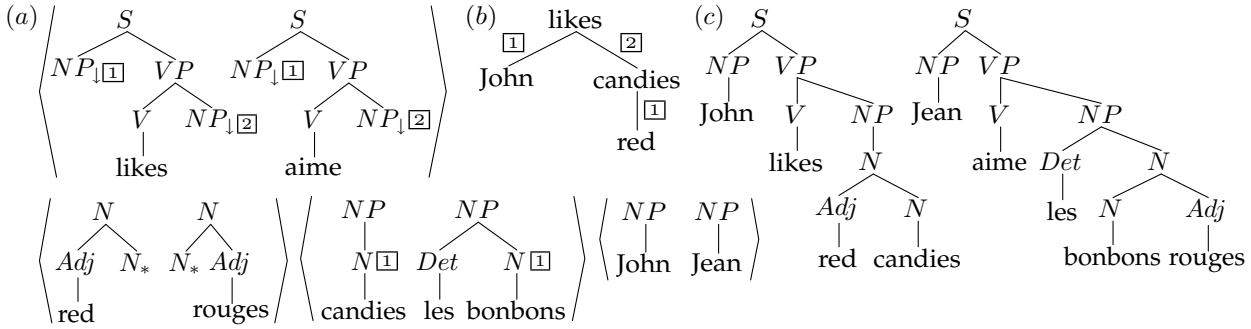

Figure 2: An example STAG derivation of the English/French sentence pair "John likes red candies"/"Jean aime les bonbons rouges". The figure is divided as follows: (a) the STAG grammar, (b) the derivation tree for the sentence pair, and (c) the derived tree pair for the sentences.

and Gildea, 2007; Zhang et al., 2006; Gildea, Satta, and Zhang, 2006). The methods for k-arization of SCFG cannot be directly applied to STAG because of the additional complexity introduced by the expressivity-increasing adjunction operation of TAG. In SCFG, where substitution is the only available operation and the depth of elementary structures is limited to one, the k-arization problem reduces to analysis of permutations of strings of nonterminal symbols. In STAG, however, the arbitrary depth of the elementary structures and the lack of restriction to contiguous strings of nonterminals introduced by adjunction substantially complicate the task.

In this paper we offer the first algorithm addressing this problem for the STAG case. We present a compile-time algorithm for transforming a STAG into a strongly-equivalent STAG that optimally minimizes k across the grammar. This is a critical minimization because k is the feature of the grammar that appears in the exponent of the complexity of parsing algorithms for STAG. Following the method of Seki et al. (1991), an STAG parser can be implemented with complexity $\mathcal{O}(n^{4\cdot(k+1)} \cdot |G|)$. By minimizing k, the worst-case complexity of a parser instantiated for a particular grammar is optimized. The k-arization algorithm performs in $\mathcal{O}(|G| + |Y| \cdot L_G^3)$ time where L_G is the maximum number of links in any single synchronous tree pair in the grammar and Y is the set of synchronous tree pairs of G. By comparison, a baseline algorithm performing exhaustive search requires $\mathcal{O}(|G| + |Y| \cdot L_G^6)$ time.[1]

The remainder of the paper proceeds as follows. In section 2 we provide a brief introduction to the STAG formalism. We present the k-arization algorithm in section 3 and an analysis of its complexity in section 4. We prove the correctness of the algorithm in section 5.

[1] In a synchronous tree pair with L links, there are $\mathcal{O}(L^4)$ pairs of valid fragments. It takes $\mathcal{O}(L)$ time to check if the two components in a pair have the same set of links. Once the synchronous fragment with the smallest number of links is excised, this process iterates at most L times, resulting in time $\mathcal{O}(L_G^6)$.

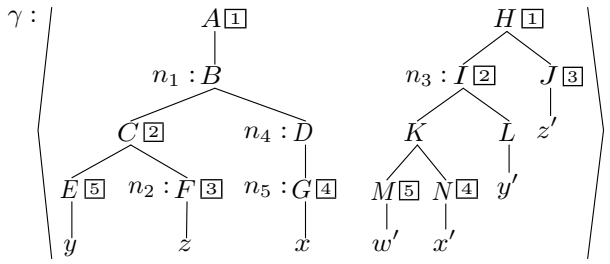

Figure 3: A synchronous tree pair containing fragments $\alpha_L = \gamma_L(n_1, n_2)$ and $\alpha_R = \gamma_R(n_3)$. Since $\mathsf{links}(n_1, n_2) = \mathsf{links}(n_3) = \{\boxed{2}, \boxed{4}, \boxed{5}\}$, we can define synchronous fragment $\alpha = \langle \alpha_L, \alpha_R \rangle$. Note also that node n_3 is a maximal node and node n_5 is not. $\sigma(n_1) = \boxed{2}\boxed{5}\boxed{5}\boxed{3}\boxed{3}\boxed{2}\boxed{4}\boxed{4}$; $\sigma(n_3) = \boxed{2}\boxed{5}\boxed{5}\boxed{4}\boxed{4}\boxed{2}$.

2 Synchronous Tree-Adjoining Grammar

A tree-adjoining grammar (TAG) consists of a set of elementary tree structures of arbitrary depth, which are combined by substitution, familiar from context-free grammars, or an operation of adjunction that is particular to the TAG formalism. **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 involves 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, are called **initial trees**. For further background, refer to the survey by Joshi and Schabes (1997).

We depart from the traditional definition in notation only by specifying adjunction and substitution sites explicitly with numbered links. Each link may be used only once in a derivation. Operations may only occur at nodes marked with a link. For simplicity of presentation we provisionally assume that only one link is permitted at a node. We later drop this assumption.

In a synchronous TAG (STAG) the elementary structures are ordered pairs of TAG trees, with a linking relation specified over pairs of nonterminal nodes. Each link has two locations, one in the left tree in a pair and the other in the right tree. An example of an STAG derivation including both substitution and adjunction is given in Figure 2. For further background, refer to the work of Shieber and Schabes (1990) and Shieber (1994).

3 k-arization Algorithm

For a synchronous tree pair $\gamma = \langle \gamma_L, \gamma_R \rangle$, a **fragment** of γ_L (or γ_R) is a complete subtree rooted at some node n of γ_L, written $\gamma_L(n)$, or else a subtree rooted at n with a gap at node n', written $\gamma_L(n, n')$; see Figure 3 for an example. We write $\mathsf{links}(n)$ and $\mathsf{links}(n, n')$ to denote the set of links of $\gamma_L(n)$ and $\gamma_L(n, n')$, respectively. When we do not know the root or gap nodes of some fragment α_L, we also write $\mathsf{links}(\alpha_L)$.

We say that a set of links Λ from γ can be **isolated** if there exist fragments α_L and α_R of γ_L and γ_R, respectively, both with links Λ. If this is the case, we can construct a synchronous fragment $\alpha = \langle \alpha_L, \alpha_R \rangle$. The goal of our algorithm is to decompose γ into synchronous fragments such that the maximum number of links of a synchronous fragment is kept to a minimum, and γ can be obtained from the synchronous fragments by means of the usual substitution and adjunction operations. In order to simplify the presentation of our algorithm we assume, without any loss of generality, that all elementary trees of the source STAG have nodes with at most two children.

3.1 Maximal Nodes

A node n of γ_L (or γ_R) is called **maximal** if (i) $\mathsf{links}(n) \neq \emptyset$, and (ii) it is either the root node of γ_L or, for its parent node n', we have $\mathsf{links}(n') \neq \mathsf{links}(n)$. Note that for every node n' of γ_L such that $\mathsf{links}(n') \neq \emptyset$ there is always a unique maximal node n such that $\mathsf{links}(n') = \mathsf{links}(n)$. Thus, for the purpose of our algorithm, we need only look at maximal nodes as places for excising tree fragments. We can show that the number of maximal nodes M_n in a subtree $\gamma_L(n)$ always satisfies $|\mathsf{links}(n)| \leq M_n \leq 2 \times |\mathsf{links}(n)| - 1$.

Let n be some node of γ_L, and let $l(n)$ be the (unique) link impinging on n if such a link exists, and $l(n) = \varepsilon$ otherwise. We associate n with a string $\sigma(n)$, defined by a pre- and post-order traversal of fragment $\gamma_L(n)$. The symbols of $\sigma(n)$ are the links in $\mathsf{links}(n)$, viewed as atomic symbols. Given a node n with p children $n_1, \ldots, n_p$, $0 \leq p \leq 2$, we define $\sigma(n) = l(n)\, \sigma(n_1) \cdots \sigma(n_p)\, l(n)$. See again Figure 3 for an example. Note that $|\sigma(n)| = 2 \times |\mathsf{links}(n)|$.

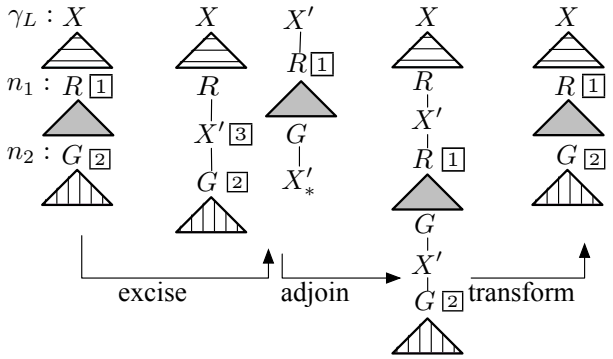

Figure 4: A diagram of the tree transformation performed when fragment $\gamma_L(n_1, n_2)$ is removed. In this and the diagrams that follow, patterned or shaded triangles represent segments of the tree that contain multiple nodes and at least one link. Where the pattern or shading corresponds across trees in a tree pair, the set of links contained within those triangles are equivalent.

1	1	0	0	0	0	0	0	0	0	1
2	0	1	0	0	0	0	1	0	0	0
5	0	0	1	1	0	0	0	0	0	0
5	0	0	1	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	1	1	0
3	0	0	0	0	0	0	0	1	1	0
2	0	1	0	0	0	0	1	0	0	0
4	0	0	0	0	1	1	0	0	0	0
4	0	0	0	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	1
	1	2	5	5	4	4	2	3	3	1

Figure 5: Table π with synchronous fragment $\langle \gamma_L(n_1, n_2), \gamma_R(n_3) \rangle$ from Figure 3 highlighted.

3.2 Excision of Synchronous Fragments

Although it would be possible to excise synchronous fragments without creating new nonterminal nodes, for clarity we present a simple tree transformation when a fragment is excised that leaves existing nodes intact. A schematic depiction is given in Figure 4. In the figure, we demonstrate the excision process on one half of a synchronous fragment: $\gamma_L(n_1, n_2)$ is excised to form two new trees. The excised tree is not processed further. In the excision process the root and gap nodes of the original tree are not altered. The material between them is replaced with a single new node with a fresh nonterminal symbol and a fresh link number. This nonterminal node and link form the adjunction or substitution site for the excised tree. Note that any link impinging on the root node of the excised fragment is by our convention included in the fragment and any link impinging on the gap node is not.

To regenerate the original tree, the excised fragment can be adjoined or substituted back into the tree from which it was excised. The new nodes that were 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.

3.3 Method

Let n_L and n_R be the root nodes of trees γ_L and γ_R, respectively. We know that $\mathsf{links}(n_L) = \mathsf{links}(n_R)$, and $|\sigma(n_L)| = |\sigma(n_R)|$, the second string being a rearrangement of the occurrences of symbols in the first one. The main data structure of our algorithm is a Boolean matrix π of size $|\sigma(n_L)| \times |\sigma(n_L)|$, whose rows are addressed by the occurrences of symbols in $\sigma(n_L)$, in the given order, and whose columns are similarly addressed by $\sigma(n_R)$. For occurrences of links $\boxed{x_1}$, $\boxed{x_2}$, the element of π at a row addressed by x_1 and a column addressed by x_2 is 1 if $x_1 = x_2$, and 0 otherwise. Thus, each row and column of π has exactly two non-zero entries. See Figure 5 for an example.

For a maximal node n_1 of γ_L, we let $\pi(n_1)$ denote the stripe of adjacent rows of π addressed by substring $\sigma(n_1)$ of $\sigma(n_L)$. If n_1 dominates n_2 in γ_L, we let $\pi(n_1, n_2)$ denote the rows of π addressed by $\sigma(n_1)$ but not by $\sigma(n_2)$. This forms a pair of horizontal stripes in π. For nodes n_3, n_4 of γ_R, we similarly define $\pi(n_3)$ and $\pi(n_3, n_4)$ as vertical stripes of adjacent columns. See again Figure 5.

Our algorithm is reported in Figure 6. For each synchronous tree pair $\gamma = \langle \gamma_L, \gamma_R \rangle$ from the input grammar, we maintain an agenda B with all candidate fragments α_L from γ_L having at least two links. These fragments are processed greedily in order of increasing number of links. The function ISOLATE(), described in more detail be-

1: **Function** KARIZE(G) {G a binary STAG}
2: $G' \leftarrow$ STAG with empty set of synch trees;
3: **for all** $\gamma = \langle \gamma_L, \gamma_R \rangle$ in G **do**
4: init π and B;
5: **while** $B \neq \emptyset$ **do**
6: $\alpha_L \leftarrow$ next fragment from B;
7: $\alpha_R \leftarrow$ ISOLATE(α_L, π, γ_R);
8: **if** $\alpha_R \neq$ **null then**
9: add $\langle \alpha_L, \alpha_R \rangle$ to G';
10: $\gamma \leftarrow$ excise $\langle \alpha_L, \alpha_R \rangle$ from γ;
11: update π and B;
12: add γ to G';
13: **return** G'

Figure 6: Main algorithm.

1: **Function** ISOLATE(α_L, π, γ_R)
2: select $n \in \gamma_R$ such that $\sigma(n)$ is the shortest string within $\sigma(n_R)$ including left/right boundaries of α_L in π;
3: **if** $|\sigma(n)| = 2 \times |\mathsf{links}(\alpha_L)|$ **then**
4: **return** $\gamma_R(n)$;
5: select $n' \in \gamma_R$ such that $\sigma(n')$ is the gap string within $\sigma(n)$ for which $\mathsf{links}(n) - \mathsf{links}(n') = \mathsf{links}(\alpha_L)$;
6: **if** n' is not defined **then**
7: **return null**; {more than one gap}
8: **return** $\gamma_R(n, n')$;

Figure 7: Find synchronous fragment.

low, looks for a right fragment α_R with the same links as α_L. Upon success, the synchronous fragment $\alpha = \langle \alpha_L, \alpha_R \rangle$ is added to the output grammar. Furthermore, we excise α from γ and update data structures π and B. The above process is iterated until B becomes empty. We show in section 5 that this greedy strategy is sound and complete.

The function ISOLATE() is specified in Figure 7. We take as input a left fragment α_L, which is associated with one or two horizontal stripes in π, depending on whether α_L has a gap node or not. The left boundary of α_L in π is the index x_1 of the column containing the leftmost occurrence of a 1 in the horizontal stripes associated with α_L. Similarly, the right boundary of α_L in π is the index x_2 of the column containing the rightmost occurrence of a 1 in these stripes. We retrieve the shortest substring $\sigma(n)$ of $\sigma(n_R)$ that spans over indices x_1 and x_2. This means that n is the lowest node from γ_R such that the links of α_L are a subset of the links of $\gamma_R(n)$.

If the condition at line 3 is satisfied, all of the matrix entries of value 1 that are found from column x_1 to column x_2 fall within the horizontal stripes associated with α_L. In this case we can report the right fragment $\alpha_R = \gamma_R(n)$. Otherwise, we check whether the entries of value 1 that fall outside of the two horizontal stripes in between columns x_1 and x_2 occur within adjacent columns, say from column $x_3 \geq x_1$ to column $x_4 \leq x_2$. In this case, we check whether there exists some node n' such that the substring of $\sigma(n)$ from position x_3 to x_4 is an occurrence of string $\sigma(n')$. This means that n' is the gap node, and we report the right fragment $\alpha_L = \gamma_R(n, n')$. See again Figure 5.

We now drop the assumption that only one link may impinge on a node. When multiple links impinge on a single node n, $l(n)$ is an arbitrary order over those links. In the execution of the algorithm, any stripe that contains one link in $l(n)$ it must include every link in $l(n)$. This prevents the excision of a proper subset of the links at any node. This preserves correctness because excising any proper subset would impose an order over the links at n that is not enforced in the input grammar. Because the links at a node are treated as a unit, the complexity of the algorithm is not affected.

4 Complexity

We discuss here an implementation of the algorithm of section 3 resulting in time complexity $\mathcal{O}(|G| + |Y| \cdot L_G^3)$, where Y is the set of synchronous tree pairs of G and L_G is the maximum number of links in a synchronous tree pair in Y.

Consider a synchronous tree pair $\gamma = \langle \gamma_L, \gamma_R \rangle$ with L links. If M is the number of maximal nodes in γ_L or γ_R, we have $M = \Theta(L)$ (Section 3.1). We implement the sparse table π in $\mathcal{O}(L)$ space, recording for each row and column the indices of its two non-zero entries. We also assume that we can go back and forth between maximal nodes n and strings $\sigma(n)$ in constant time. Here each $\sigma(n)$ is represented by its boundary positions within $\sigma(n_L)$ or $\sigma(n_R)$, n_L and n_R the root nodes of γ_L and γ_R, respectively.

At line 2 of the function ISOLATE() (Figure 7) we retrieve the left and right boundaries by scanning the rows of π associated with input fragment α_L. We then retrieve node n by visiting all maximal nodes of γ_L spanning these boundaries. Under the above assumptions, this can be done in time $\mathcal{O}(L)$. In a similar way we can implement line 5, resulting in overall run time $\mathcal{O}(L)$ for function ISOLATE().

In the function KARIZE() (Figure 6) we use buckets B_i, $1 \leq i \leq L$, where each B_i stores the candidate fragments α_L with $|\mathsf{links}(\alpha_L)| = i$. To populate these buckets, we first process fragments $\gamma_L(n)$ by visiting bottom up the maximal nodes of γ_L. The quantity $|\mathsf{links}(n)|$ is computed from the quantities $|\mathsf{links}(n_i)|$, where n_i are the highest maximal nodes dominated by n. (There are at most two such nodes.) Fragments $\gamma_L(n, n')$ can then be processed using the relation $|\mathsf{links}(n, n')| = |\mathsf{links}(n)| - |\mathsf{links}(n')|$. In this way each fragment is processed in constant time, and population of all the buckets takes $\mathcal{O}(L^2)$ time.

We now consider the while loop at lines 5 to 11 in function KARIZE(). For a synchronous tree pair γ, the loop iterates once for each candidate fragment α_L in some bucket. We have a total of $\mathcal{O}(L^2)$ iterations, since the initial number of candidates in the buckets is $\mathcal{O}(L^2)$, and the possible updating of the buckets after a synchronous fragment is removed does not increase the total size of all the buckets. If the links in α_L cannot be isolated, one iteration takes time $\mathcal{O}(L)$ (the call to function ISOLATE()). If the links in α_L can be isolated, then we need to restructure π and to repopulate the buckets. The former can be done in time $\mathcal{O}(L)$ and the latter takes time $\mathcal{O}(L^2)$, as already discussed. Crucially, the updating of π and the buckets takes place no more than $L - 1$ times. This is because each time we excise a synchronous fragment, the number of links in γ is reduced by at least one.

We conclude that function KARIZE() takes time $\mathcal{O}(L^3)$ for each synchronous tree γ, and the total running time is $\mathcal{O}(|G| + |Y| \cdot L_G^3)$, where Y is the set of synchronous tree pairs of G. The term $|G|$ accounts for the reading of the input, and dominates the complexity of the algorithm only in case there are very few links in each synchronous tree pair.

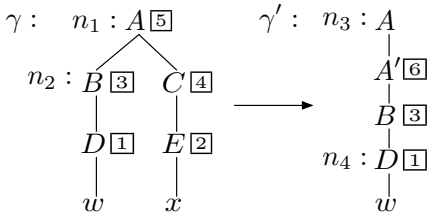

Figure 8: In γ links 3 and 5 cannot be isolated because the fragment would have to contain two gaps. However, after the removal of fragment $\gamma(n_1, n_2)$, an analogous fragment $\gamma'(n_3, n_4)$ may be removed.

5 Proof of Correctness

The algorithm presented in the previous sections produces an optimal k-arization for the input grammar. In this section we sketch a proof of correctness of the strategy employed by the algorithm.[2]

The k-arization strategy presented above is greedy in that it always chooses the excisable fragment with the smallest number of links at each step and does not perform any backtracking. We must therefore show that this process cannot result in a non-optimal solution. If fragments could not overlap each other, this would be trivial to show because the excision process would be confluent. If all overlapping fragments were cases of complete containment of one fragment within another, the proof would also be trivial because the smallest-to-largest excision order would guarantee optimality. However, it is possible for fragments to partially overlap each other, meaning that the intersection of the set of links contained in the two fragments is non-empty and the difference between the set of links in one fragment and the other is also non-empty. Overlapping fragment configurations are given in Figure 9 and discussed in detail below.

The existence of partially overlapping fragments complicates the proof of optimality for two reasons. First, the excision of a fragment α that is partially overlapped with another fragment β necessarily precludes the excision of β at a later stage in the ex-

[2]Note that the soundness of the algorithm can be easily verified from the fact that the removal of fragments can be reversed by performing standard STAG adjunction and substitution operations until a single STAG tree pair is produced. This tree pair is trivially homomorphic to the original tree pair and can easily be mapped to the original tree pair.

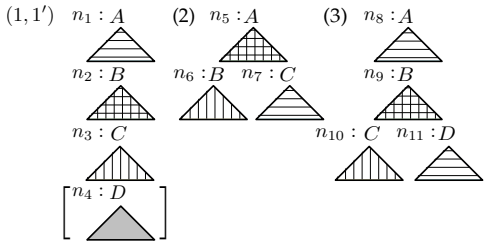

Figure 9: The four possible configurations of overlapped fragments within a single tree. For type 1, let $\alpha = \gamma(n_1, n_3)$ and $\beta = \gamma(n_2, n_4)$. The roots and gaps of the fragments are interleaved. For type $1'$, let $\alpha = \gamma(n_1, n_3)$ and $\beta = \gamma(n_2)$. The root of β dominates the gap of α. For type 2, let $\alpha = \gamma(n_5, n_6)$ and $\beta = \gamma(n_5, n_7)$. The fragments share a root and have gap nodes that do not dominate each other. For type 3 let $\alpha = \gamma(n_8, n_{10})$ and $\beta = \gamma(n_9, n_{11})$. The root of α dominates the root of β, both roots dominate both gaps, but neither gap dominates the other.

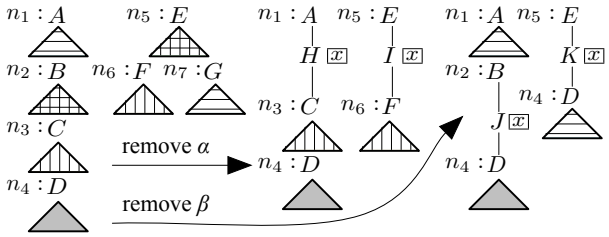

Figure 10: Removal from a tree pair γ containing type 1–type 2 fragment overlap. The fragment α is represented by the horizonal-lined pieces of the tree pair. The fragment β is represented by the vertical-lined pieces of the tree pair. Cross-hatching indicates the overlapping portion of the two fragments.

cision process. Second, the removal of a fragment may cause a previously non-isolatable set of links to become isolatable, effectively creating a new fragment that may be advantageous to remove. This is demonstrated in Figure 8. These possibilities raise the question of whether the choice between removing fragments α and β may have consequences at a later stage in the excision process. We demonstrate that this choice cannot affect the k found for a given grammar.

We begin by sketching the proof of a lemma that shows that removal of a fragment β that partially overlaps another fragment α always leaves an analogous fragment that may be removed.

5.1 Validity Preservation

Consider a STAG tree pair γ containing the set of links Λ and two synchronous fragments α and β with α containing links $\mathsf{links}(\alpha)$ and β containing $\mathsf{links}(\beta)$ ($\mathsf{links}(\alpha), \mathsf{links}(\beta) \subsetneq \Lambda$).

If α and β do not overlap, the removal of β is defined as validity preserving with respect to α.

If α and β overlap, removal of β from γ is **validity preserving** with respect to α if after the removal there exists a valid synchronous fragment (containing at most one gap on each side) that contains all and only the links $(\mathsf{links}(\alpha) - \mathsf{links}(\beta)) \cup \{\boxed{x}\}$ where $\boxed{x}$ is the new link added to γ.

We prove a lemma that removal of any synchronous fragment from an STAG tree pair is validity preserving with respect to all of the other synchronous fragments in the tree pair.

It suffices to show that for two arbitrary synchronous fragments α and β, the removal of β is validity preserving with respect to α. We show this by examination of the possible configurations of α and β.

Consider the case in which β is fully contained within α. In this case $\mathsf{links}(\beta) \subsetneq \mathsf{links}(\alpha)$. The removal of β leaves the root and gap of α intact in both trees in the pair, so it remains a valid fragment. The new link is added at the new node inserted where β was removed. Since β is fully contained within α, this node is below the root of α but not below its gap. Thus, the removal process leaves α with the links $(\mathsf{links}(\alpha) - \mathsf{links}(\beta)) \cup \{\boxed{x}\}$, where $\boxed{x}$ is the link added in the removal process; the removal is validity preserving.

Synchronous fragments may partially overlap in several different ways. There are four possible configurations for an overlapped fragment within a single tree, depicted in Figure 9. These different single-tree overlap types can be combined in any way to form valid synchronous fragments. Due to space constraints, we consider two illustrative cases and leave the remainder as an exercise to the reader.

An example of removing fragments from a tree set containing type 1–type 2 overlapped fragments is given in Figure 10. Let $\alpha = \langle \gamma_L(n_1, n_3), \gamma_R(n_5, n_6) \rangle$. Let

$\beta = \langle \gamma_L(n_2, n_4), \gamma_R(n_5, n_7) \rangle$. If α is removed, the validity preserving fragment for β is $\langle \gamma'_L(n_1, n_4), \gamma'_R(n_5) \rangle$. It contains the links in the vertical-lined part of the tree and the new link $\boxed{x}$. This forms a valid fragment because both sides contain at most one gap and both contain the same set of links. In addition, it is validity preserving for β because it contains exactly the set of links that were in $\mathsf{links}(\beta)$ and not in $\mathsf{links}(\alpha)$ plus the new link $\boxed{x}$. If we instead choose to remove β, the validity preserving fragment for α is $\langle \gamma'_L(n_1, n_4), \gamma'_R(n_5) \rangle$. The links in each side of this fragment are the same, each side contains at most one gap, and the set of links is exactly the set left over from $\mathsf{links}(\alpha)$ once $\mathsf{links}(\beta)$ is removed plus the newly generated link $\boxed{x}$.

An example of removing fragments from a tree set containing type $1'$–type 3 (reversed) overlapped fragments is given in Figure 11. If α is removed, the validity preserving fragment for β is $\langle \gamma'_L(n_1), \gamma'_R(n_4) \rangle$. If β is removed, the validity preserving fragment for α is $\langle \gamma'_L(n_1, n_8), \gamma'_R(n_4) \rangle$.

Similar reasoning follows for all remaining types of overlapped fragments.

5.2 Proof Sketch

We show that smallest-first removal of fragments is optimal. Consider a decision point at which a choice is made about which fragment to remove. Call the size of the smallest fragments at this point m, and let the set of fragments of size m be X with $\alpha, \beta \in X$.

There are two cases to consider. First, consider two partially overlapped fragments $\alpha \in X$ and $\delta \notin X$. Note that $|\mathsf{links}(\alpha)| < |\mathsf{links}(\delta)|$. Validity preservation of α with respect to δ guarantees that δ or its validity preserving analog will still be available for excision after α is removed. Excising δ increases k more than excising α or any fragment that removal of α will lead to before δ is considered. Thus, removal of δ cannot result in a smaller value for k if it is removed before α rather than after α.

Second, consider two partially overlapped fragments $\alpha, \beta \in X$. Due to the validity preservation lemma, we may choose arbitrarily between the fragments in X without jeopardizing our ability to later remove other fragments (or their validity preserving analogs) in that set. Removal of fragment α cannot increase the size of any remaining fragment.

Removal of α or β may generate new fragments that were not previously valid and may reduce the size of existing fragments that it overlaps. In addition, removal of α may lead to availability of smaller fragments at the next removal step than removal of β (and vice versa). However, since removal of either α or β produces a k of size at least m, the later removal of fragments of size less than m cannot affect the k found by the algorithm. Due to validity preservation, removal of any of these smaller fragments will still permit removal of all currently existing fragments or their analogs at a later step in the removal process.

If the removal of α generates a new fragment δ of size larger than m all remaining fragments in X (and all others smaller than δ) will be removed before δ is considered. Therefore, if removal of β generates a new fragment smaller than δ, the smallest-first strategy will properly guarantee its removal before δ.

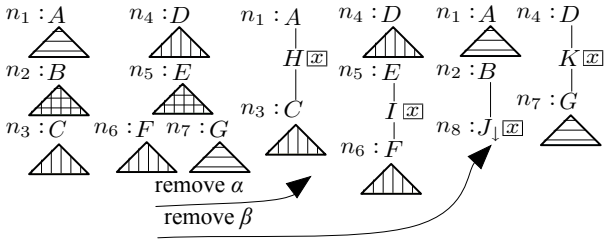

Figure 11: Removal from a tree pair γ containing a type $1'$–type 3 (reversed) fragment overlap. The fragment α is represented by the horizontal lined pieces of the tree pair. The fragment β is represented by the vertical-lined pieces of the tree pair. Cross-hatching indicates the overlapping portion of the two fragments.

6 Conclusion

In order for STAG to be used in machine translation and other natural-language processing tasks it must be possible to process it efficiently. The difficulty in parsing STAG stems directly from the factor k that indicates the degree to which the correspondences are intertwined within the elementary structures of the grammar. The algorithm presented in this paper is the first method available for k-arizing a synchronous TAG grammar into an equivalent grammar with an optimal value for k. The algorithm operates offline and requires only $\mathcal{O}(|G| + |Y| \cdot L_G^3)$ time. Both the derivation trees and derived trees produced are trivially homomorphic to those that are produced by the original grammar.

References

Aho, Alfred V. and Jeffrey D. Ullman. 1969. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3(1):37–56.

Chiang, David and Owen Rambow. 2006. The hidden TAG model: synchronous grammars for parsing resource-poor languages. In *Proceedings of the 8th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 8)*, pages 1–8.

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

Han, Chung-Hye. 2006a. 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.

Han, Chung-Hye. 2006b. A tree adjoining grammar analysis of the syntax and semantics of it-clefts. In *Proceedings of the 8th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 8)*, pages 33–40, 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.

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.

Nesson, Rebecca, Stuart M. Shieber, and Alexander Rush. 2006. Induction of probabilistic synchronous tree-insertion grammars for machine translation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA 2006)*, Boston, Massachusetts, 8-12 August.

Satta, Giorgio. 1992. Recognition of linear context-free rewriting systems. In *Proceedings of the 10th Meeting of the Association for Computational Linguistics (ACL92)*, pages 89–95, Newark, Delaware.

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

Shieber, Stuart M. 1994. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385, November.

Shieber, Stuart M. and Yves Schabes. 1990. Synchronous tree adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING '90)*, Helsinki, August.

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

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)*, April.

Zhang, Hao, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proceedings of the Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*.