

# Binarization of Synchronous Context-Free Grammars

Liang Huang  
University of Pennsylvania

Daniel Gildea  
University of Rochester

Hao Zhang  
University of Rochester

Kevin Knight  
USC/Information Sciences Institute

*Systems based on synchronous grammars and tree transducers promise to improve the quality of statistical machine translation output, but are often very computationally intensive. The complexity is exponential in the size of individual grammar rules due to arbitrary re-orderings between the two languages. We develop a theory of binarization for synchronous context-free grammars and present a linear-time algorithm for binarizing synchronous rules when possible. In our large-scale experiments, we found that almost all rules are binarizable and the resulting binarized rule set significantly improves the speed and accuracy of a state-of-the-art syntax-based machine translation system. We also discuss the more general, and computationally more difficult, problem of finding good parsing strategies for non-binarizable rules, and present an approximate polynomial-time algorithm for this problem.*

## 1. Introduction

Several recent syntax-based models for machine translation (Chiang, 2005; Galley et al., 2004) can be seen as instances of the general framework of synchronous grammars and tree transducers. In this framework, both alignment (*synchronous parsing*) and decoding can be thought of as parsing problems, whose complexity is in general exponential in the number of nonterminals on the right hand side of a grammar rule. To alleviate this problem, we investigate bilingual binarization to factor the synchronous grammar to a smaller branching factor, although it is not guaranteed to be successful for any synchronous rule with an arbitrary permutation. In particular:

- We develop a technique called *synchronous binarization* and devise a *linear-time* binarization algorithm such that the resulting rule set allows efficient algorithms for both synchronous parsing and decoding with integrated  $n$ -gram language models.
- We examine the effect of this binarization method on end-to-end translation quality on a large-scale syntax-based system, compared to a more typical baseline method, and a state-of-the-art phrase-based system.
- We examine the ratio of *binarizability* in large, empirically-derived rule sets, and show that the vast majority is binarizable. However, we also provide, for the first time, real examples of non-binarizable cases verified by native speakers.
- For these small amount of non-binarizable rules, we investigate the question of finding the most efficient synchronous parsing or decoding strategy. While this

problem is believed to be NP-complete, we prove two results that substantially reduce the search space over strategies. We also present an optimal algorithm that runs tractably in practice and a polynomial-time algorithm that is a good approximation of the former.

Melamed (2003) discusses binarization of multi-text grammars on a theoretical level, showing the importance and difficulty of binarization for efficient synchronous parsing. One way around this difficulty is to stipulate that all rules must be binary from the outset, as in inversion-transduction grammar (ITG) (Wu, 1997) and the binary synchronous context-free grammar (SCFG) employed by the Hiero system (Chiang, 2005) to model the hierarchical phrases. In contrast, the rule extraction method of (Galley et al., 2004) aims to incorporate more syntactic information by providing parse trees for the target language and extracting tree transducer rules that apply to the parses. This approach results in rules with many nonterminals, making good binarization techniques critical.

We explain how synchronous rule binarization interacts with  $n$ -gram language models and affects decoding for machine translation in Section 2. We define binarization formally in Section 3, and present an efficient algorithm for the problem in Section 4. Experiments described in Section 5 show that binarization improves machine translation speed and quality. Some rules cannot be binarized, and we discuss the problem of finding the best parsing strategy for these rules in Section 6.

## 2. Motivation

We will discuss binarization for both SCFG systems and tree transducer systems. Consider the following Chinese sentence and its English translation:

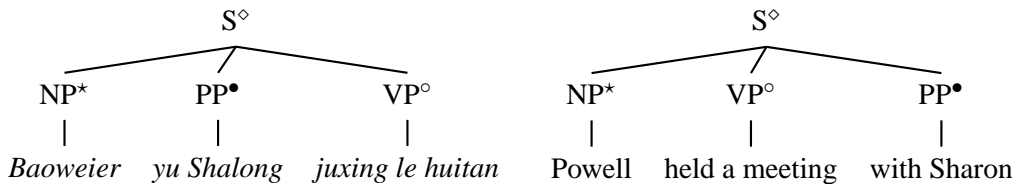
- (1) *Baoweier yu Shalong juxing le huitan*  
 Powell with Sharon hold [*past.*] meeting  
 “Powell held a meeting with Sharon”

Suppose we have the following SCFG, where superscripts indicate reorderings (formal definitions of SCFGs with a more flexible notation can be found in Section 3):

- (2)  $S \rightarrow NP^1 PP^2 VP^3, NP^1 VP^3 PP^2$   
 $NP \rightarrow Baoweier, Powell$   
 $VP \rightarrow juxing le huitan, held a meeting$   
 $PP \rightarrow yu Shalong, with Sharon$

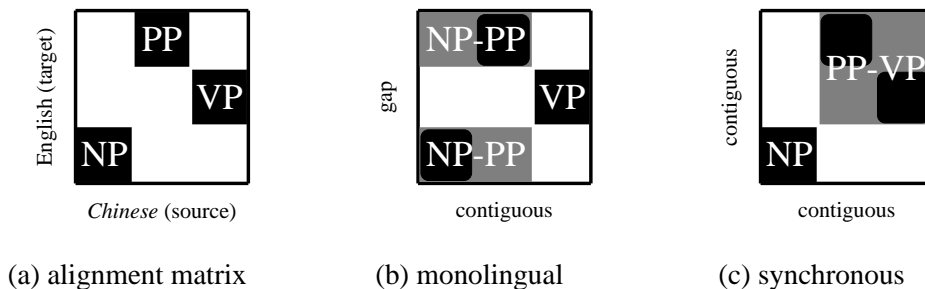
Decoding can be cast as a (monolingual) parsing problem since we only need to parse the source-language side of the SCFG, as if we were constructing a CFG projected on Chinese out of the SCFG:

- (3)  $S \rightarrow NP PP VP$   
 $NP \rightarrow Baoweier$   
 $VP \rightarrow juxing le huitan$   
 $PP \rightarrow yu Shalong$



**Figure 1**

A pair of synchronous parse trees in the SCFG (2). The superscript symbols ( $^{\diamond^* \bullet}$ ) indicate pairs of synchronous nonterminals (and sub trees).



**Figure 2**

The alignment matrix and two binarization schemes, with virtual nonterminals in gray.

The only extra work we need to do for decoding is to build corresponding target-language (English) subtrees in parallel. In other words, we build *synchronous trees* when parsing the source-language input, as shown in Figure 1.

For efficient parsing, we need to binarize the projected CFG either explicitly into Chomsky Normal Form as required by the CKY algorithm, or implicitly into a dotted representation as in the Earley algorithm. To simplify the presentation, we will focus on the former but the discussion below can be easily adapted to the latter. Rules can be binarized in different ways. For example, we could binarize the first rule left to right or right to left (See Figure 2):

$$\begin{array}{l}
 S \rightarrow NP-PP \ VP \\
 NP-PP \rightarrow NP \ PP
 \end{array}
 \quad \text{or} \quad
 \begin{array}{l}
 S \rightarrow NP \ PP-VP \\
 PP-VP \rightarrow PP \ VP
 \end{array}$$

We call those intermediate symbols (e.g. PP-VP) *virtual nonterminals* and corresponding rules *virtual rules*, whose probabilities are all set to 1.

These two binarizations are no different in the translation-model-only decoding described above, just as in monolingual parsing. However, in the source-channel approach to machine translation, we need to combine probabilities from the translation model (TM) (an SCFG) with the language model (an  $n$ -gram), which is shown to be very important for translation quality (Chiang, 2005). To do bigram-integrated decoding, we need to augment each chart item  $(X, i, j)$  with two target-language *boundary words*  $u$  and  $v$  to produce a bigram-item like  $\begin{pmatrix} u & \dots & v \\ i & X & j \end{pmatrix}$ , following the dynamic programming

algorithm of (Wu, 1996).<sup>1</sup>

Now the two binarizations have very different effects. In the first case, we first combine NP with PP:

$$\frac{\left( \begin{array}{ccc} \text{Powell} & \dots & \text{Powell} \\ 1 & \text{NP} & 2 \end{array} \right) : p \quad \left( \begin{array}{ccc} \text{with} & \dots & \text{Sharon} \\ 2 & \text{PP} & 4 \end{array} \right) : q}{\left( \begin{array}{cccc} \text{Powell} & \dots & \text{Powell} & \dots & \text{with} & \dots & \text{Sharon} \\ 1 & & \text{NP-PP} & & 4 & & \end{array} \right) : pq}$$

where  $p$  and  $q$  are the scores of antecedent items.

This situation is unpleasant because in the target-language NP and PP are *not* contiguous so we cannot apply language model scoring when we build the NP-PP item. Instead, we have to maintain all four boundary words (rather than two) and postpone the language model scoring till the next step where NP-PP is combined with  $\left( \begin{array}{ccc} \text{held} & \dots & \text{meeting} \\ 2 & \text{VP} & 4 \end{array} \right)$  to form an S item. We call this binarization method *monolingual binarization* since it works only on the source-language projection of the rule without respecting the constraints from the other side.

This scheme generalizes to the case where we have  $n$  nonterminals in a SCFG rule, and the decoder conservatively assumes nothing can be done on language model scoring (because target-language spans are non-contiguous in general) until the real nonterminal has been recognized. In other words, target-language boundary words from each child nonterminal of the rule will be cached in all virtual nonterminals derived from this rule. In the case of  $m$ -gram integrated decoding, we have to maintain  $2(m - 1)$  boundary words for each child nonterminal, which leads to a prohibitive overall complexity of  $O(|w|^{3+2n(m-1)})$ , which is exponential in rule size (Huang, Zhang, and Gildea, 2005). Aggressive pruning must be used to make it tractable in practice, which in general introduces many search errors and adversely affects translation quality.

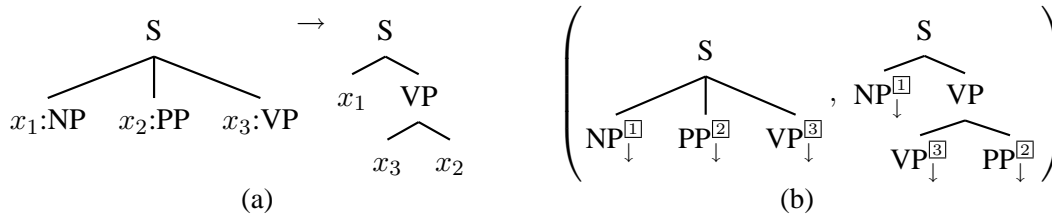
In the second case, however:

$$\frac{\left( \begin{array}{ccc} \text{with} & \dots & \text{Sharon} \\ 2 & \text{PP} & 4 \end{array} \right) : r \quad \left( \begin{array}{ccc} \text{held} & \dots & \text{meeting} \\ 4 & \text{VP} & 7 \end{array} \right) : s}{\left( \begin{array}{ccc} \text{held} & \dots & \text{Sharon} \\ 2 & \text{PP-VP} & 7 \end{array} \right) : rs \cdot \Pr(\text{with} \mid \text{meeting})}$$

Here since PP and VP are contiguous (but swapped) in the target-language, we can include the language model score by adding  $\Pr(\text{with} \mid \text{meeting})$ , and the resulting item again has two boundary words. Later we add  $\Pr(\text{held} \mid \text{Powell})$  when the resulting item is combined with  $\left( \begin{array}{ccc} \text{Powell} & \dots & \text{Powell} \\ 1 & \text{NP} & 2 \end{array} \right)$  to form an S item. As illustrated in Figure 2, PP-VP has contiguous spans on both source- and target-sides, so that we can generate a binary-branching SCFG:

$$(4) \quad \begin{array}{l} \text{S} \rightarrow \text{NP}^{[1]} \text{PP-VP}^{[2]}, \quad \text{NP}^{[1]} \text{PP-VP}^{[2]} \\ \text{PP-VP} \rightarrow \text{VP}^{[1]} \text{PP}^{[2]}, \quad \text{PP}^{[2]} \text{VP}^{[1]} \end{array}$$

<sup>1</sup> An alternative to integrated decoding is *rescoring*, where one first computes the  $k$ -best translations according to the TM only, and then rerank the  $k$ -best list with the language model costs. This method runs very fast in practice (Huang and Chiang, 2005), but often produces a considerable number of search errors since the true best translation is often outside of the  $k$ -best list, especially for longer sentences.



**Figure 3**

Two equivalent representations of the first rule in Example 5). (a): tree transducer; (b): STSG. The  $\downarrow$  arrows denote substitution sites, which corresponds to variables in tree transducers.

In this case  $m$ -gram integrated decoding can be done in  $O(|w|^{3+4(m-1)})$  time which is a much lower-order polynomial and no longer depends on rule size (Wu, 1996), allowing the search to be much faster and more accurate, as is evidenced in the Hiero system of Chiang (2005), which restricts the hierarchical phrases to form binary-branching SCFG rules.

The same reasoning applies to tree transducer rules. Suppose we have the following transducer rules (Rounds, 1970; Galley et al., 2004):

$$\begin{array}{ll}
 S(x_1:NP \ x_2:PP \ x_3:VP) & \rightarrow \ S(x_1 \ VP(x_3 \ x_2)) \\
 NP(Baoweier) & \rightarrow \ NP(NNP(Powell)) \\
 VP(juxing \ le \ huitan) & \rightarrow \ VP(VBD(held) \ NP(DT(a) \ NPS(meeting))) \\
 PP(yu \ Shalong) & \rightarrow \ PP(TO(with) \ NP(NNP(Sharon)))
 \end{array}
 \tag{5}$$

where the reorderings of nonterminals are denoted by variables  $x_i$ .

In the tree-transducer formalism of Rounds (1970), the right-hand (target) side subtree can have multiple levels, as in the first rule above. This system can model non-isomorphic transformations on English parse trees to “fit” another language, for example, learning that the  $(V \ S \ O)$  structure in Arabic should be transformed into a  $(S \ (V \ O))$  structure in English, by looking at two-level tree fragments (Knight and Graehl, 2005). From a synchronous rewriting point of view, this is more akin to synchronous tree substitution grammar (STSG) (Eisner, 2003; Shieber, 2004) (see Figure 3). This larger locality captures more linguistic phenomena and leads to a better parameter estimation. By imagining the right-hand-side trees as special nonterminals, we can virtually create an SCFG with the same generative capacity. We can again create a projected CFG which will be exactly the same as in (3), and build English sub-trees while parsing the Chinese input. In this sense we can neglect the tree structures when binarizing a tree-transducer rule, and consider only the alignment (or permutation) of the nonterminal variables. Again, rightmost binarization is preferable for the first rule.

The benefit of binary grammars also applies to synchronous parsing (alignment). Wu (1997) shows that parsing a binary-branching SCFG is in  $O(|w|^6)$  while parsing SCFG with arbitrary rules is NP-hard (Satta and Peserico, 2005). For example, in Figure 2, the complexity of synchronous parsing for the original grammar (a) is  $O(|w|^8)$ , since we have to maintain 4 indices on either side, giving a total of 8; parsing the monolingually binarized grammar (b) involves 7 indices, 3 on the Chinese side and 4 on the English side. In contrast, the synchronously binarized version (c) requires only  $3+3=6$  indices,

which can be thought of as “CKY in two dimensions.” Thus, while binarization is not possible for all rules, an efficient parsing algorithm is guaranteed if a binarization is found.

In general, if we are given an arbitrary synchronous rule with many nonterminals, what are the good decompositions that lead to a binary grammar? Figure 2 suggests that a binarization is good if every virtual nonterminal has contiguous spans on both sides. We formalize this idea in the next section.

### 3. Synchronous Binarization

#### Definition 1

A **synchronous CFG** (SCFG) is a context-free rewriting system for generating string pairs. Each rule (*synchronous production*)

$$A \rightarrow \alpha, B \rightarrow \beta$$

rewrites a pair of *synchronous nonterminals*  $(A, B)$  in two dimensions subject to the constraint that there is a one-to-one mapping between the nonterminal occurrences in  $\alpha$  and the nonterminal occurrences in  $\beta$ . Each co-indexed child nonterminal pair is a pair of synchronous nonterminals and will be further rewritten as a unit.

Note that this notation is more flexible than those in the previous section, in the sense that we can allow different symbols to be synchronized, which is essential to capture the syntactic divergences between languages. For example, the following rule from Chinese to English

$$VP \rightarrow VB^{[1]} NN^{[2]}, \quad VP \rightarrow VBZ^{[1]} NNS^{[2]}$$

illustrates the fact that Chinese does not have a plural noun (NNS) or third-person-singular verb (VBZ), although we can always convert this form back into the other notation by creating a compound nonterminal alphabet:

$$(VP, VP) \rightarrow (VB, VBZ)^{[1]} (NN, NNS)^{[2]}, \quad (VB, VBZ)^{[1]} (NN, NNS)^{[2]}.$$

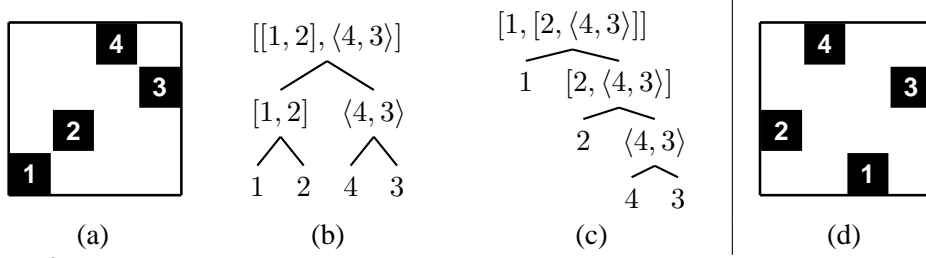
We define the language  $L(G)$  produced by an SCFG  $G$  as the pairs of terminal strings produced by rewriting exhaustively from the start nonterminal pair.

As shown in Section 4.2, terminals do not play an important role in binarization. So we now write rules in the following notation:

$$X \rightarrow X_1^{[1]} \dots X_n^{[n]}, \quad Y_{\pi(1)}^{[\pi(1)]} \dots Y_{\pi(n)}^{[\pi(n)]}$$

where  $X_i$  and  $Y_i$  are variables ranging over nonterminals in the source and target projections of the synchronous grammar, respectively, and  $\pi$  is the **permutation** of the rule. We also define an SCFG rule as  $n$ -ary if its permutation is of  $n$  and call an SCFG  $n$ -ary if its longest rule is  $n$ -ary. Our goal is to produce an equivalent *binary* SCFG for an input  $n$ -ary SCFG.

However, not every SCFG can be binarized. In fact, the binarizability of an  $n$ -ary rule is determined by the structure of its permutation, which can sometimes be resistant to factorization (Aho and Ullman, 1972). We now turn to rigorously defining the binarizability of permutations.



**Figure 4**

(a)-(c): alignment matrix and two binarization trees for  $(1, 2, 4, 3)$ . (d): alignment matrix for the non-binarizable permuted sequence  $(2, 4, 1, 3)$ .

**Definition 2**

A **permuted sequence** is a permutation of consecutive integers. If a permuted sequence  $\mathbf{a}$  can be split into the concatenation of two permuted sequences  $\mathbf{b}$  and  $\mathbf{c}$ , then  $(\mathbf{b}; \mathbf{c})$  is called a **proper split** of  $\mathbf{a}$ . We say  $\mathbf{b} < \mathbf{c}$  if each element in  $\mathbf{b}$  is smaller than any element in  $\mathbf{c}$ .

For example,  $(3, 5, 4)$  is a permuted sequence while  $(2, 5)$  is not. As special cases, single numbers are permuted sequences as well.  $(3; 5, 4)$  is a proper split of  $(3, 5, 4)$  while  $(3, 5; 4)$  is not. A proper split has the following property:

**Lemma 1.** A split  $\mathbf{a} = (\mathbf{b}; \mathbf{c})$  is a proper if and only if  $\mathbf{b} < \mathbf{c}$  or  $\mathbf{c} < \mathbf{b}$ .

*Proof.* The  $\Rightarrow$  direction is trivial by the definition of proper split.

We prove the  $\Leftarrow$  direction by contradiction. If  $\mathbf{b} < \mathbf{c}$  but  $\mathbf{b}$  is not a permuted sequence, i.e., the set of  $\mathbf{b}$ 's elements is not consecutive, then there must be some  $x \in \mathbf{c}$  such that  $\min \mathbf{b} < x < \max \mathbf{b}$ , which contradicts the fact that  $\mathbf{b} < \mathbf{c}$ . We have a similar contradiction if  $\mathbf{c}$  is not a permuted sequence. Now that both  $\mathbf{b}$  and  $\mathbf{c}$  are permuted sequence,  $(\mathbf{b}; \mathbf{c})$  is a proper split. The case when  $\mathbf{b} > \mathbf{c}$  is similar.  $\square$

**Definition 3**

A permuted sequence  $\mathbf{a}$  is said to be **binarizable** if either

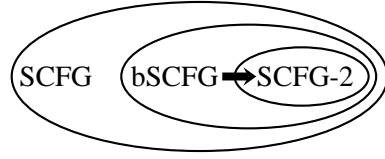
1.  $\mathbf{a}$  is a singleton, i.e.  $\mathbf{a} = (a)$ , or
2. there is a proper split  $\mathbf{a} = (\mathbf{b}; \mathbf{c})$  where  $\mathbf{b}$  and  $\mathbf{c}$  are both binarizable. We call such split a **binarizable split**.

This is a recursive definition, and it implies that there is a hierarchical binarization pattern associated with each binarizable sequence which we rigorously define below:

**Definition 4**

A **binarization tree**  $bi(\mathbf{a})$  of a binarizable sequence  $\mathbf{a}$  is either

1.  $\mathbf{a}$  if  $\mathbf{a} = (a)$ , or
2. either  $[bi(\mathbf{b}), bi(\mathbf{c})]$  if  $\mathbf{b} < \mathbf{c}$ , or  $\langle bi(\mathbf{b}), bi(\mathbf{c}) \rangle$  if otherwise, where  $\mathbf{a} = (\mathbf{b}; \mathbf{c})$  is a binarizable split, and  $bi(\mathbf{b})$  is a binarization tree of  $\mathbf{b}$  and  $bi(\mathbf{c})$  a binarization tree of  $\mathbf{c}$ .



**Figure 5**

Subclasses of SCFG. The thick arrow denotes the direction of synchronous binarization. For clarity reasons, binary SCFG is coded as SCFG-2.

Here we use  $\llbracket$  and  $\langle \rangle$  for straight ( $\mathbf{b} < \mathbf{c}$ ) and inverted ( $\mathbf{b} > \mathbf{c}$ ) combinations respectively, following the ITG notation of Wu (1997). Note that a binarizable sequence might have multiple binarization trees. See Figure 4 for a binarizable sequence (1, 2, 4, 3) with its two possible binarization trees and a non-binarizable sequence (2, 4, 1, 3).

We are now able to define the binarizability of SCFGs.

**Definition 5**

An SCFG is said to be **binarizable** if the permutation of each synchronous production is binarizable. We denote the class of binarizable SCFGs as **bSCFG**.

This set, bSCFG, represents an important subclass of SCFG that is easy to handle (for example, parsable in  $O(|w|^6)$ ) and covers many interesting longer-than-two rules. The goal of *synchronous binarization* is then to convert a binarizable grammar  $G$  in bSCFG, which might be  $n$ -ary with  $n \geq 2$ , into an equivalent binary grammar  $G'$  that generates the same string pairs (see Figure 5). This is always possible because for each rule in  $G$  with its permutation  $\pi$ , there is a binarization tree  $bi(\pi)$  which essentially decomposes the original permutation into a set of binary ones. All that remains is to decorate the skeleton binarization tree with nonterminal symbols and attach terminals to the skeleton appropriately (see the next section for details). We state this result as the following theorem:

**Theorem 1.** For each grammar  $G$  in bSCFG, there exists a binary SCFG  $G'$ , such that  $L(G') = L(G)$ .

**4. Binarization Algorithms**

We have reduced the problem of binarizing an SCFG rule into the problem of binarizing its permutation. The simplest algorithm for this problem is to try all bracketings of a permutation and pick one that corresponds to a binarization tree. The number of all possible bracketings of a sequence of length  $n$  is known to be the *Catalan Number* (Catalan, 1844)

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

which grows exponentially with  $n$ . A better approach is to reduce this problem to an instance of synchronous ITG parsing (Wu, 1997). Here the parallel string pair that we are parsing is the integer sequence (1... $n$ ) and its permutation ( $\pi(1)\dots\pi(n)$ ). The goal of the ITG parsing is to find a synchronous tree that agrees with the alignment indicated

by the permutation. Synchronous ITG parsing runs in time  $O(n^6)$  but can be improved to  $O(n^4)$  since there is no insertion or deletion in a permutation. But this is still too slow in practice.

Another problem besides efficiency is that there are possibly multiple binarization trees for many permutations whereas we just need one. We would prefer a *consistent* pattern of binarization trees across different permutations so that sub-binarizations (virtual nonterminals) can be shared. For example, permutations (1, 3, 2, 5, 4) and (1, 3, 2, 4) can share the common sub-binarization tree  $[1, \langle 3, 2 \rangle]$ . To this end, we can borrow the non-ambiguous ITG of Wu (1997, Sec. 7) that prefers left-heavy binarization trees so that for each permutation there is a unique synchronous derivation.<sup>2</sup> We now refine the definition of binarization trees accordingly:

### Definition 6

A binarization tree  $bi(\mathbf{a})$  is said to be **canonical** if the split at each non-leaf node of the tree is the *rightmost* binarizable split.

For example, for sequence (1, 2, 4, 3), the binarization tree  $[[1, 2], \langle 4, 3 \rangle]$  is canonical, while  $[1, [2, \langle 4, 3 \rangle]]$  is not, because its top-level split is not at the rightmost binarizable split (1, 2; 4, 3). By definition, there is a unique canonical binarization tree for each binarizable sequence.

We next present an algorithm that is both fast and consistent.

### 4.1 The linear-time skeleton algorithm

Shapiro and Stephens (1991, p. 277) informally present an iterative procedure that, in each pass, scans the permuted sequence from left to right and combines two adjacent subsequences whenever possible. This procedure produces canonical binarization trees and runs in  $O(n^2)$  time since we need  $n$  passes in the worst case. Inspired by the Graham Scan Algorithm Graham (1972) for computing Convex-Hulls from Computational Geometry, we modify this procedure and improve it into a linear-time algorithm that only needs one pass through the sequence.

The skeleton binarization algorithm is an instance of the widely used left-to-right shift-reduce algorithm. It maintains a stack for contiguous subsequences discovered so far, for example: 2-5, 1. In each iteration, it shifts the next number from the input and repeatedly tries to reduce the top two elements on the stack if they are consecutive. See Algorithm 1 for the pseudo-code and Figures 6 and 7 for example runs on binarizable and non-binarizable permutations, respectively.

We need the following lemma to prove the properties of the Algorithm:

**Lemma 2.** If  $\mathbf{c}$  is a permuted sequence (properly) within a binarizable permuted sequence  $\mathbf{a}$ , then  $\mathbf{c}$  is also binarizable.

*Proof.* We prove by induction on the length of  $\mathbf{a}$ . Base case:  $|\mathbf{a}| = 2$ , a (proper) subsequence of  $\mathbf{a}$  has length 1, so it is binarizable. For  $|\mathbf{a}| > 2$ , because  $\mathbf{a}$  has a binarization

<sup>2</sup> We are not aiming at *optimal sharing*, i.e., a strategy that produces the smallest binarized grammar for a given ruleset, which would require a global optimization problem over the whole set. In practice, we can only use online algorithms that binarize rules one by one. The left-heavy (or its symmetric variant, right-heavy) preference we choose here is one of the obvious candidates for consistency.

---

**Algorithm 1** The Linear-time Binarization Algorithm
 

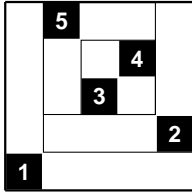
---

```

1: function BINARIZABLE(a)
2:    $top \leftarrow 0$  ▷ stack top pointer
3:   PUSH( $stack, (a_1, a_1)$ ) ▷ initial shift
4:   for  $i \leftarrow 2$  to  $|a|$  do ▷ for each remaining element
5:     PUSH( $stack, (a_i, a_i)$ ) ▷ shift
6:     while  $top > 1$  and CONSECUTIVE( $stack[top], stack[top - 1]$ ) do
7:       ▷ keep reducing if possible
8:        $(p, q) \leftarrow$  COMBINE( $stack[top], stack[top - 1]$ )
9:        $top \leftarrow top - 2$ 
10:      PUSH( $stack, (p, q)$ )
11:   return ( $top = 1$ ) ▷ returns true iff. the input is reduced to a single element
12:
13: function CONSECUTIVE( $(a, b), (c, d)$ )
14:   return ( $b = c - 1$ ) or ( $d = a - 1$ ) ▷ either straight or inverted
15: function COMBINE( $(a, b), (c, d)$ )
16:   return ( $\min(a, c), \max(b, d)$ )
  
```

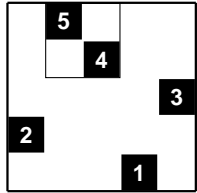
---

iteration	stack	input	action
		1 5 3 4 2	
	1	5 3 4 2	shift
1	1 5	3 4 2	shift
2	1 5 3	4 2	shift
3	1 5 3 4	2	shift
	1 5 <u>3-4</u>	2	reduce [3, 4]
	1 <u>3-5</u>	2	reduce $\langle 5, [3, 4] \rangle$
4	1 3-5 2		shift
	1 <u>2-5</u>		reduce $\langle \langle 5, [3, 4] \rangle, 2 \rangle$
	<u>1-5</u>		reduce $[1, \langle \langle 5, [3, 4] \rangle, 2 \rangle]$

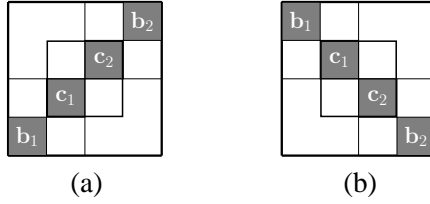

**Figure 6**

Example of Algorithm 1 on the binarizable input (1, 5, 3, 4, 2). The rightmost column shows the binarization-trees generated at each reduction step.

iteration	stack	input	action
		2 5 4 1 3	
	2	5 4 1 3	shift
1	2 5	4 1 3	shift
2	2 5 4	1 3	shift
3	2 <u>4-5</u>	1 3	reduce $\langle 5, 4 \rangle$
4	2 4-5 1	3	shift
5	2 4-5 1 3		shift


**Figure 7**

Example of Algorithm 1 on the non-binarizable input (2, 5, 4, 1, 3).



**Figure 8**

Illustration of the proof of Lemma 2. The arrangement of  $(b_1, c_1; c_2, b_2)$  must be either all straight as in (a) or all inverted as in (b).

tree, there exists a (binarizable) split which is nearest to the root and splits  $c$  into two parts. Let the split be  $(b_1, c_1; c_2, b_2)$ , where  $c = (c_1; c_2)$ , and either  $b_1$  or  $b_2$  can be empty. By Lemma 1, we have  $c_1 < c_2$  or  $c_1 > c_2$ . By Lemma 1 again, we have that  $(c_1; c_2)$  is a proper split of  $c$ , i.e., both  $c_1$  and  $c_2$  are themselves permuted sequences. We also know both  $(b_1, c_1)$  and  $(c_2, b_2)$  are binarizable. By the induction hypothesis,  $c_1$  and  $c_2$  are both binarizable. So we conclude that  $c = (c_1; c_2)$  is binarizable (See Figure 8).  $\square$

We now state the central result of this work.

**Theorem 2.** Algorithm 1 runs in time linear to the length of the input, and succeeds (i.e., it reduces the input into one single element) if and only if the input permuted sequence  $a$  is binarizable, in which case the binarization tree recovered is canonical.

*Proof.* We prove the following three parts of this theorem:

1. If Algorithm 1 succeeds, then  $a$  is binarizable because we can recover a binarization tree from the algorithm.
2. If  $a$  is binarizable, then Algorithm 1 must succeed and the binarization tree recovered must be canonical:

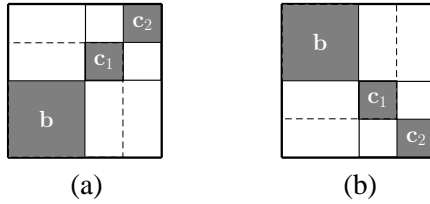
We prove by a complete induction on  $n$ , the length of  $a$ .

Base case:  $n = 1$ , trivial. Assume it holds for all  $n' < n$ .

If  $a$  is binarizable, then let  $a = (b; c)$  be its rightmost binarizable split. By definition, both  $b$  and  $c$  are binarizable. By the induction hypothesis, the algorithm succeeds on the partial input  $b$ , reducing it to the single element  $stack[0]$  on the stack and recovering its canonical binarization tree  $bi(b)$ .

If  $c$  is a singleton, the algorithm will combine it with the element  $stack[0]$  and succeed. The final binarization tree is canonical because the top-level split is at the rightmost binarizable split, and both subtrees are canonical.

If  $c$  is not a singleton, we want to show by contradiction that the algorithm will never combine  $b$  with a proper prefix of  $c$ . Since  $a = (b; c)$  is a proper split, we know that either  $b < c$  or  $c < b$ . Now if the algorithm makes a combination of  $(b; c_1)$  for some proper prefix  $c_1$  where  $c = (c_1; c_2)$ , we have either  $c_1 < c_2$  or  $c_1 > c_2$ . By Lemma 1,  $(c_1; c_2)$  is a proper split. Since  $c$  is binarizable,



**Figure 9**  
 Illustration of the proof of Theorem 2. The combination of  $(\mathbf{b}; \mathbf{c}_1)$  (in dashed squares) contradicts the assumption that  $(\mathbf{b}; \mathbf{c})$  is the rightmost binarizable split of  $\mathbf{a}$ .

by Lemma 2,  $\mathbf{c}_2$  is also binarizable. So  $(\mathbf{b}, \mathbf{c}_1; \mathbf{c}_2)$  is a binarizable split to the right of  $(\mathbf{b}; \mathbf{c})$ , which contradicts the assumption that the latter is the right-most binarizable split (see Figure 9).

Therefore, the algorithm will scan through the whole  $\mathbf{c}$  as if from the empty stack. By the induction hypothesis again, it will reduce  $\mathbf{c}$  into  $stack[1]$  on the stack and recover its canonical binarization tree  $bi(\mathbf{c})$ . Since  $\mathbf{b}$  and  $\mathbf{c}$  are combinable, the algorithm reduces  $stack[0]$  and  $stack[1]$  in the last step, forming the canonical binarization tree for  $\mathbf{a}$ , which is either  $[bi(\mathbf{b}), bi(\mathbf{c})]$  or  $\langle bi(\mathbf{b}), bi(\mathbf{c}) \rangle$ .

3. The running time of Algorithm 1 (regardless of success or failure) is linear in  $n$ :  
 By amortized analysis (Cormen, Leiserson, and Rivest, 1990), there are exactly  $n$  shifts and at most  $n - 1$  reductions, and each shift or reduction takes  $O(1)$  time. So the total time complexity is  $O(n)$ .

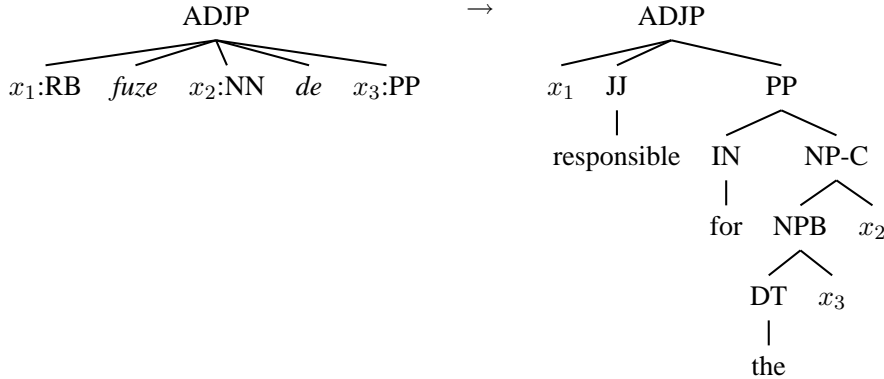
□

## 4.2 Binarizing tree transducers

Without loss of generality, we have discussed how to binarize synchronous productions involving only nonterminals through binarizing the corresponding skeleton permutations. We will now describe how to adapt the skeleton algorithm for the decoding experiments in Section 5.2.

First, we are dealing with tree transducer rules. We view each left-hand side subtree as a monolithic nonterminal symbol and factor each transducer rule into two SCFG rules: one from the root nonterminal to the subtree, and the other from the subtree to the leaves. In this way we can uniquely reconstruct the transducer derivation using the two-step

SCFG derivation. For example, consider the following tree transducer rule:

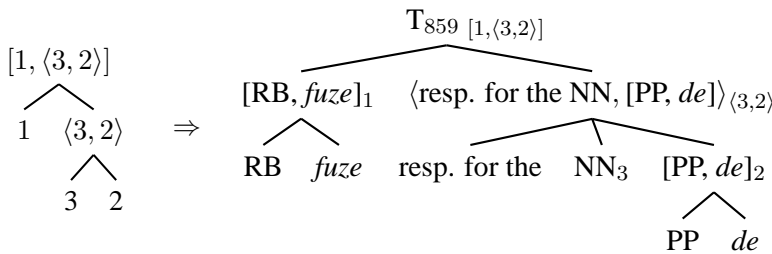


We create a specific nonterminal, say,  $T_{859}$ , which is a unique identifier for the left-hand side subtree and generate the following two SCFG rules:

$$\begin{aligned} \text{ADJP} &\rightarrow T_{859}^{[1]}, & \text{ADJP} &\rightarrow T_{859}^{[1]} \\ T_{859} &\rightarrow \text{RB}^{[1]} \text{ fuze } \text{PP}^{[2]} \text{ de } \text{NN}^{[3]}, & T_{859} &\rightarrow \text{RB}^{[1]} \text{ resp. for the } \text{NN}^{[3]} \text{ PP}^{[2]} \end{aligned}$$

Second, besides synchronous nonterminals, terminals in the two languages can also be present, as in the above example. It turns out we can attach the terminals to the skeleton parse for the synchronous nonterminal strings quite freely as long as we can uniquely reconstruct the original rule from its binary parse tree. In order to do so we need to keep track of sub-alignments including both aligned nonterminals and neighboring terminals.

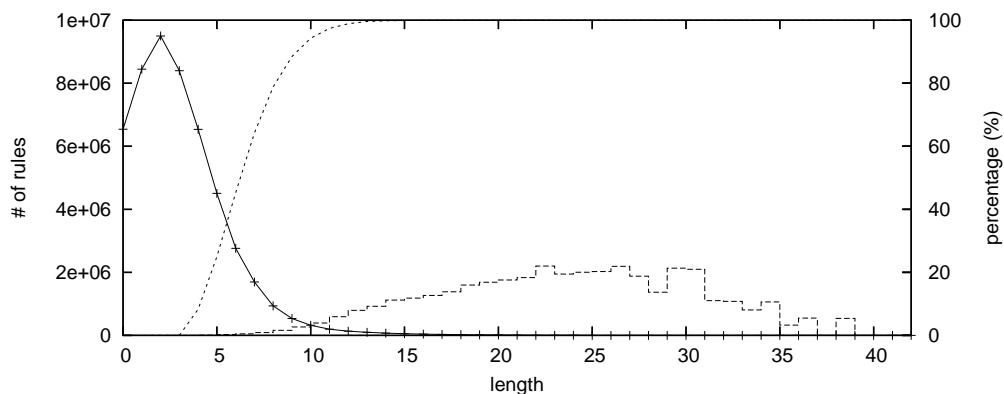
When binarizing the second rule above, we first run the skeleton algorithm to binarize the underlying permutation  $(1, 3, 2)$  to its binarization tree  $[1, \langle 3, 2 \rangle]$ . Then we do a post-order traversal to the skeleton tree, combining Chinese terminals (one at a time) at the leaf nodes and merging English terminals greedily at internal nodes:



A pre-order traversal of the decorated binarization tree gives us the following binary SCFG rules:

$$\begin{aligned} T_{859} &\rightarrow V_1^{[1]} V_2^{[2]}, & T_{859} &\rightarrow V_1^{[1]} V_2^{[2]} \\ V_1 &\rightarrow \text{RB}^{[1]} \text{ fuze}, & V_1 &\rightarrow \text{RB}^{[1]} \\ V_2 &\rightarrow V_3^{[1]} \text{NN}^{[2]}, & V_2 &\rightarrow \text{resp. for the } \text{NN}^{[2]} V_3^{[1]} \\ V_3 &\rightarrow \text{PP}^{[1]} \text{ de}, & V_3 &\rightarrow \text{PP}^{[1]} \end{aligned}$$

where the virtual nonterminals are:



**Figure 10**

The solid-line curve represents the distribution of all rules against permutation lengths. The the dashed-line stairs indicate the percentage of non-binarizable rules in our initial rule set while the dotted-line denotes that percentage among all permutations.

$V_1$ : [RB, *fuze*]

$V_2$ : ⟨resp. for the NN, [PP, *de*]⟩

$V_3$ : [PP, *de*]

Analogous to the “dotted rules” in Earley parsing for monolingual CFGs, the names we create for the virtual nonterminals reflect the underlying subalignments, ensuring intermediate states can be shared across different string-to-tree rules without causing ambiguity.

The whole binarization algorithm still runs in time linear in the number of symbols in the rule (including both terminals and nonterminals).

## 5. Experiments

In this section, we investigate two empirical questions of synchronous binarization.

### 5.1 How many rules are (synchronously) binarizable?

Shapiro and Stephens (1991) and Wu (1997, Sec. 4) show that the percentage of binarizable cases over all permutations of length  $n$  quickly approaches 0 as  $n$  grows (see Figure 10). However, for machine translation, it is more meaningful to compute the *empirical* ratio of binarizable rules extracted from real text. We answer this question in both large-scale automatically aligned data and small-scale hand-aligned data.

*Automatically aligned data* Our rule set here is obtained by first doing word alignment using GIZA++ on a Chinese-English parallel corpus containing 50 million words in English, then parsing the English sentences using a variant of Collins parser, and finally extracting rules using the graph-theoretic algorithm of Galley et al. (2004). We did a “spectrum analysis” on the resulting rule set with 50,879,242 rules. Figure 10 shows how the rules are distributed against their lengths (number of nonterminals). We can see that the percentage of non-binarizable rules in each bucket of the same length does not exceed

25%. Overall, 99.7% of the rules are binarizable. Even for the 0.3% non-binarizable rules, human evaluations show that the majority of them are due to alignment errors. It is also interesting to know that 86.8% of the rules have monotonic permutations, i.e. either taking identical or totally inverted order.

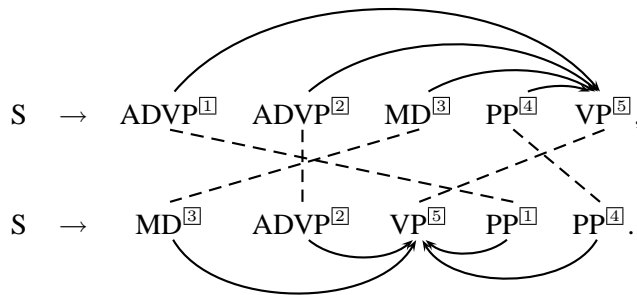
*Hand-aligned data* One might wonder whether automatic alignments computed by GIZA++ are systematically biased toward or against binarizability. If syntactic constraints not taken into account by GIZA++ enforce binarizability, automatic alignments could tend to contain spurious nonbinarizable cases. On the other hand, simply by preferring monotonic alignments, GIZA++ might tend to miss complex nonbinarizable patterns.

To test this, we carried out a similar experiment on the hand-aligned Chinese-English data of Liu, Liu, and Lin (2005) which contains 935 pairs of parallel sentences. Of the 13713 rules extracted using the same method as above, 0.3% (44) are non-binarizable, which is exactly the same ratio as the GIZA-aligned data. The following is an interesting example of non-binarizable rules:

(6) *dangtian*<sub>1</sub> *hai*<sub>2</sub> *jiang*<sub>3</sub> [*yu ... Mishira*]<sub>4</sub> *huiwu*<sub>5</sub>  
 that-day also will with ... Mishira meet

‘will<sub>3</sub> also<sub>2</sub> meet<sub>5</sub> [on the same day]<sub>1</sub> [with Mishira, (chief secretary to the Indian prime minister and national security advisor) ]<sub>4</sub>’

where ... is the long phrase in the parenthesis modifying “Mishira”. Here the non-binarizable permutation is (3, 2, 5, 1, 4) which is reduceable to (2, 4, 1, 3). The SCFG version of the tree-transducer rule (with dependency links in solid arcs and permutation in dashed lines) is:



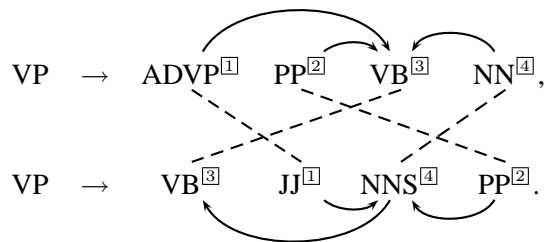
Note that the Chinese ADVP<sup>1</sup> *dangtian* (lit., “that day”) is translated into an English PP<sup>1</sup> (on the same day), but the dependency structures on both sides are isomorphic.

A simpler but slightly non-literal example is the following:

(7) *jinyibu*<sub>1</sub> [*jiu zhongdong weiji*]<sub>2</sub> *juxing*<sub>3</sub> *huitan*<sub>4</sub>  
 further on Mideast crisis hold talk

‘hold<sub>3</sub> further<sub>1</sub> talks<sub>4</sub> [on the Mideast crisis]<sub>2</sub>’

where the SCFG version of the tree-transducer rule (in the same format as the previous example) is:



Note that the Chinese ADVP<sup>1</sup> (*jinyibu*) modifying the verb VB<sup>3</sup> becomes a JJ<sup>1</sup> (further) in the English translation modifying the object of the verb, NNS<sup>4</sup>, and this change also happens to PP<sup>2</sup>. This is an example of syntactic divergence, where the dependency structures are not isomorphic between the two languages (Eisner, 2003).

Wu (1997, p. 158) has “been unable to find real examples” of non-binarizable cases, at least in “fixed-word-order languages that are lightly inflected, such as English and Chinese.” Our empirical results not only confirm that this is largely (99.7% as in our datasets) correct, but also provide, for the first time, “real examples” between English and Chinese, verified by native speakers. Wellington, Waxmonsky, and Melamed (2006) used a different measure of non-binarizability, which is on the sentence-level permutations, as opposed to rule-level permutation in our case, and reported 5% non-binarizable cases for a different hand-aligned English-Chinese dataset, but they did not provide real examples.

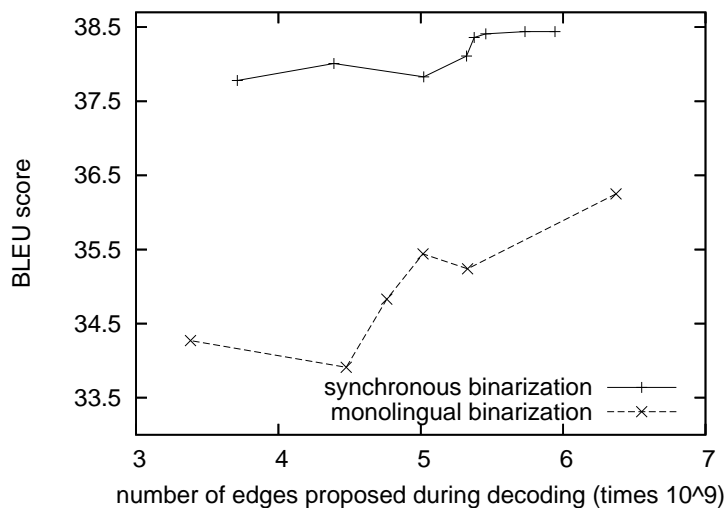
## 5.2 Does synchronous binarizer help decoding?

We did experiments on our CKY-based decoder with two binarization methods. It is the responsibility of the binarizer to instruct the decoder how to compute the language model scores from children nonterminals in each rule. The baseline method is monolingual left-to-right binarization. As shown in Section 2, decoding complexity with this method is exponential in the size of the longest rule and since we postpone all the language model scorings, pruning in this case is also biased.

To move on to synchronous binarization, we first did an experiment using the above baseline system without the 0.3% of rules that are non-binarizable and did not observe any difference in BLEU scores. This indicates that we can safely focus on the binarizable rules, discarding the rest.

The decoder now works on the binary translation rules supplied by an external synchronous binarizer. As shown in Section 1, this results in a simplified decoder with a polynomial time complexity, allowing less aggressive and more effective pruning based on both translation model and language model scores.

We compare the two binarization schemes in terms of translation quality with various pruning thresholds. The rule set is that of the previous section. The test set has 116 Chinese sentences of no longer than 15 words. Both systems use trigram as the integrated language model. Figure 11 demonstrates that decoding accuracy is significantly improved after synchronous binarization. The number of edges (or *items*, in the deductive parsing terminology) proposed during decoding is used as a measure of the size of search space, or time efficiency. Our system is consistently faster and more accurate than the baseline system.



**Figure 11**

Comparing the two binarization methods in terms of translation quality against search effort.

system	BLEU
monolingual binarization	36.25
synchronous binarization	38.44
alignment-template system	37.00

**Table 1**

Syntax-based systems vs. ATS

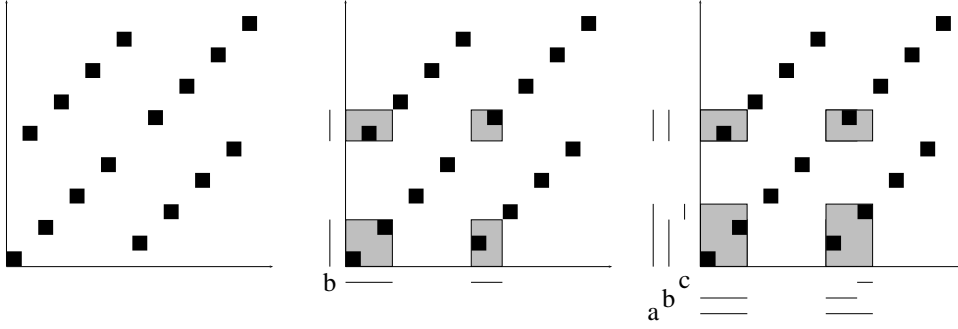
We also compare the top result of our synchronous binarization system with the state-of-the-art alignment-template approach (ATS) (Och and Ney, 2004). The results are shown in Table 1. Our system has a promising improvement over the ATS system which is trained on a larger data-set but tuned independently. A larger-scale system based on our best result performs very well in the 2006 NIST MT Evaluation (ISI Machine Translation Team, 2006), achieving the best overall BLEU scores in the Chinese-to-English track among all participants.<sup>3</sup>

## 6. Factorizations for Non-Binarizable Grammars

As shown in Section 5.1, there are indeed a small amount of rules that are non-binarizable. This section investigates strategies for handling those rules efficiently, both for synchronous parsing (alignment) and for decoding with an  $m$ -gram language model. We show how to analyze parsing and decoding strategies for a given SCFG rule in Sec-

<sup>3</sup> NIST 2006 Machine Translation Evaluation Official Results.

<http://www.nist.gov/speech/tests/mt/mt06eval.official.results.html>



**Figure 12**

Left, a general pattern of non-binarizable permutations. Center, a partially completed chart item with two spans in each dimension; the intersection of the completed spans is shaded. Right, the combination of the item from the center panel with a singleton item. The two subsets of nonterminals in the inner marked spans are combined into a new chart item with the outer spans.

tion 6.1, and then present an exponential-time dynamic programming algorithm for finding the best strategy in Section 6.2. We prove that factorizing an SCFG rule into smaller SCFG rules is a safe preprocessing step for finding the best strategy in Section 6.3, which leads to much faster computation in many cases. In Section 6.4 we describe an  $O(n^3)$  algorithm that finds good approximations to the optimal strategy in practice.

### 6.1 Formalizing the Problem

The complexity for parsing given a grammar factorization can be expressed in terms of the number of spans of the items being combined at each step. Each span of the items being combined creates two indices into the string being parsed, for the span's beginning and end. For example, in the combination shown in Figure 12, there are two spans for one of the items being combined (four indices in each dimension), while the singleton item has one span (two indices in each dimension). However, some of the indices are tied together: if we are joining two spans into one span in the new item, the old items' beginning and end point must be equal. If we are combining  $b$  spans for item  $B$  with  $c$  spans for item  $C$  to produce  $a$  spans for a new item  $A$ , the number of linked indices is  $b + c - a$ . The total number of indices is

$$2(b + c) - (b + c - a) = a + b + c \quad (8)$$

where  $2(b + c)$  represents the original beginning and end points, and  $b + c - a$  the number of shared indices. More generally, if we are combining  $k$  items  $B_i$  ( $i = 1, \dots, k, k \geq 2$ ) together to produce a new item  $A$ , the generalized formula for counting the total number of indices is

$$a + \sum_{i=1}^k b_i \quad (9)$$

where  $b_i$  is the number of spans for  $B_i$  and  $a$  is the number of spans for the resulting item  $A$ . In the next section, we will prove we never need  $k > 2$  if we allow discontinuous spans.

Exactly the same analysis applies in both of our parsing dimensions. If we are combining item  $B$  with  $b_e$  target spans and  $b_f$  source spans with item  $C$  having  $c_e$  target spans and  $c_f$  source spans to form new item  $A$  having  $a_e$  target spans and  $a_f$  source spans, the complexity of the operation is  $|w|^{a_e+b_e+c_e+a_f+b_f+c_f}$ . In Figure 12,  $a_e = b_e = a_f = b_f = 2$  and  $c_e = c_f = 1$ , giving overall complexity of  $O(|w|^{10})$ .

The “a-plus-b-plus-c” formula can also be generalized for decoding with an integrated  $m$ -gram language model. At first glance, since we need to maintain  $(m - 1)$  boundary words at both the left and right edges of each target span, the total number of interacting variables is:

$$2(m - 1)(b_e + c_e) + a_f + b_f + c_f$$

However, by using the “hook trick” suggested by Huang, Zhang, and Gildea (2005), we can optimize the decoding algorithm into one where boundary  $m$ -gram words have to match to validate combinations. In this version, each left boundary for a substring of an output language hypothesis contains  $m - 1$  words of language model state, and each right boundary contains a “hook” specifying what the next  $m - 1$  words must be. This yields a complexity analysis similar to that for synchronous parsing, based on the total number of boundaries, but now multiplied by a factor of  $m - 1$ :

$$(m - 1)(a_e + b_e + c_e) + a_f + b_f + c_f \quad (10)$$

for translation from source to target.

### Definition 7

The **number of  $m$ -gram weighted spans** of a constituent, denoted  $a_m$ , is defined as the number of source spans plus the number target spans weighted by the language model factor  $(m - 1)$ :

$$a_m = a_f + (m - 1)a_e \quad (11)$$

Using this notation, we can rewrite the expression for the complexity of decoding in Equation 10 as a simple sum of the numbers of weighted spans of constituents  $A$ ,  $B$ , and  $C$ :

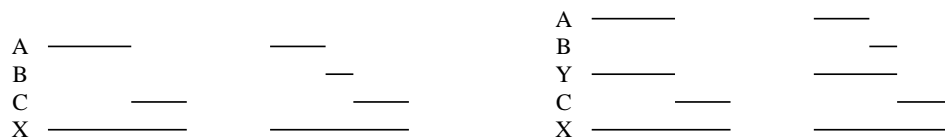
$$a_m + b_m + c_m \quad (12)$$

and more generally when there are  $k \geq 2$  constituents combine together:

$$\begin{aligned} (a_f + (m - 1)a_e) + \sum_{i=1}^k (b_{fi} + (m - 1)b_{ei}) \\ = a_m + \sum_{i=1}^k b_{mi} \end{aligned} \quad (13)$$

It can be seen that as  $m$  grows, the parsing/decoding strategies that favor contiguity on the output side will prevail. Later on, we will use experimental results to demonstrate the bias on parsing strategy introduced by language model integration.

The analysis above applies to one combination of two subsets of a rule’s children during parsing or decoding. A strategy for parsing (or decoding) the entire rule must



**Figure 13**

Left, example spans for a ternary rule decomposition  $X \rightarrow A B C$ . Line segments represent the projection of each set of child nonterminals into a single language, as in Figure 12. Right, factorization into  $X \rightarrow Y C$  and  $Y \rightarrow A B$ .

build up the complete set of the rule’s children through a sequence of such combinations. Thus a parsing strategy corresponds to a recursive partitioning of the rule’s children, that is, an unordered rooted tree having the child nonterminals as leaves. Each node in the partition tree represents a subset of nonterminals used as a partial result in the chart for parsing, built by combining the subsets corresponding to the node’s children. This combination step at each node has complexity determined by the number of spans, and the overall complexity of a parsing strategy is the complexity of the strategy’s worst combination step. We wish to find the recursive partition with the lowest overall complexity. Unfortunately, the number of recursive partitions of  $n$  items grows superexponentially, as  $0.175n!n^{-3/2}2.59^n = \Theta(\Gamma(n+1)2.59^n)$  (Schröder, 1870, Problem IV). More formally, the optimization over the space of all recursive partitions is expressed as:

$$best(A) = \min_{B:A=\bigcup_{i=1}^k B_i} \max\{compl(A \rightarrow B_1\dots B_k), \max_{i=1}^k best(B_i)\} \quad (14)$$

where  $compl(A \rightarrow B_1\dots B_k)$  is given by Equation 13. The expression implies that we can solve the optimization problem using dynamic programming techniques.

## 6.2 Combining Two Subsets at a Time is Optimal

In this section we show that a branching factor of more than two is not necessary in our recursive partitions, by showing that any ternary combination can be factored into two binary combinations with no increase in complexity. This fact leads to a more efficient, but still exponential, algorithm for finding the best parsing strategy for a given SCFG rule.

**Theorem 3.** For any SCFG rule, if there exists a recursive partition of child nonterminals which enables tabular parsing of an input sentence  $w$  in time  $O(|w|^k)$ , then there exists a recursive binary partition whose corresponding parser is also  $O(|w|^k)$ .

*Proof.* We use the notion of number of weighted spans (Equation 13) to concisely analyze the complexity of synchronous parsing/decoding. For any fixed  $m$ , we count a constituent’s number of spans using the weighted span value from Equation 11, and we drop both the adjective “weighted” and the  $m$  subscript from this point forward. Consider a ternary rule  $X \rightarrow A B C$  where  $X$  has  $x$  spans,  $A$  has  $a$  spans,  $B$  has  $b$  spans, and  $C$  has  $c$  spans. In the example shown in Figure 13,  $x = 2$ ,  $a = 2$ ,  $b = 1$ , and  $c = 2$ .

The complexity for parsing this is  $O(|w|^{a+b+c+x})$ . Now factor this rule into two rules:

$$X \rightarrow Y C$$

$$Y \rightarrow A B$$

and refer to the number of spans in partial constituent  $Y$  as  $y$ . Parsing  $Y \rightarrow A B$  takes time  $O(|w|^{y+a+b})$ , so we need to show that:

$$y + a + b \leq a + b + c + x$$

to show that we can parse this new rule in no more time than the original ternary rule. Subtracting  $a + b$  from both sides, we need to prove that:

$$y \leq c + x$$

Each of the  $y$  spans in  $Y$  corresponds to a left edge. (In the case of decoding, each edge has multiplicity of  $(m - 1)$  on the output language side.) The left edge in each span of  $Y$  corresponds to the left edge of a span in  $X$  or to the right edge of a span in  $C$ . Therefore,  $Y$  has at most one span for each span in  $C \cup X$ , so  $y \leq c + x$ .

Returning to the first rule in our factorization, the time to parse  $X \rightarrow Y C$  is  $O(|w|^{x+y+c})$ . We know that:

$$y \leq a + b$$

since  $Y$  was formed from  $A$  and  $B$ . Therefore

$$x + y + c \leq x + (a + b) + c$$

so parsing  $X \rightarrow Y C$  also takes no more time than the original rule  $X \rightarrow A B C$ . By induction over the number of subsets, a rule having any number of subsets on the righthand side can be converted into a series of binary rules.  $\square$

Our finding that combining no more than two subsets of children at a time is optimal implies that we need consider only *binary* recursive partitions, which correspond to unordered binary rooted trees having the SCFG rule's child nonterminals as leaves. The total number of binary recursive partitions of  $n$  nodes is  $\frac{(2n-3)!}{2^{n-2}(n-2)!} = \Theta(\Gamma(n - \frac{1}{2})2^{n-1})$  (Schröder, 1870, Problem III). Note that this number grows much faster than the Catalan Number which characterizes the number of *bracketings* representing the search space of synchronous binarization (Section 4).

While the total number of binary recursive partitions is still superexponential, the binary branching property also enables a straightforward dynamic programming algorithm shown in Algorithm 2. The same algorithm can be used to find optimal strategies for synchronous parsing or for  $m$ -gram decoding: for parsing, the variables  $a$ ,  $b$ , and  $c$  in Line 7 refer to the total number of spans of  $A$ ,  $B$ , and  $C$  (Equation 8), while for decoding,  $a$ ,  $b$ , and  $c$  refer to weighted spans (Equation 12). The dynamic programming states correspond to subsets of the input rule's children, for which an optimal strategy has already been computed. In each iteration of the algorithm's inner loop, each of the child nonterminals is identified as belonging to  $B$ ,  $C$ , or neither  $B$  nor  $C$ , making the total

---

**Algorithm 2** An  $O(3^n)$  search algorithm for the optimal parsing strategy that may contain discontinuous spans

---

```

1: function BESTDISCONTINUOUSPARSER( $S \rightarrow X_1 \dots X_n$ )
2:   for  $i \leftarrow 2$  to  $n$  do
3:     for  $A \subset \{X_1 \dots X_n\}$  s.t.  $|A| = i$  do
4:        $best(A) \leftarrow \infty$ 
5:       for  $B, C$  s.t.  $A = B \cup C \wedge B \cap C = \emptyset$  do
6:         Let  $a, b$ , and  $c$  denote the number of  $A, B$ , and  $C$ 's spans
7:          $compl(A \rightarrow BC) = \max\{a + b + c, best(B), best(C)\}$ 
8:         if  $compl(A \rightarrow BC) < best(A)$  then
9:            $best(A) \leftarrow compl(A \rightarrow BC)$ 
10:           $rule(A) \leftarrow A \rightarrow BC$ 
11:   return  $best(\{X_1 \dots X_n\})$ 

```

---

running time of the algorithm is  $O(3^n)$ . While this is exponential in  $n$ , it is a significant improvement over considering all recursive partitions.

The algorithm can be improved by adopting a best-first exploration strategy (Knuth, 1977), in which dynamic programming items are placed on a priority queue sorted according to their complexity, and only used to build further items after all items of lower complexity have been exhausted. This technique, shown in Algorithm 3, guarantees polynomial-time processing on input permutations of bounded complexity. To see why this is, observe that each rule of the form  $A \rightarrow BC$  that has complexity no greater than  $k$  can be written using a string of  $k_e < k$  indices into the target nonterminal string to represent the spans' boundaries. For each index we must specify whether the corresponding nonterminal either: starts a span of subset  $B$ , starts a span of subset  $C$ , or ends a span of  $B \cup C$ . Therefore there are  $O((3n)^k)$  rules of complexity no greater than  $k$ . If there exists a parsing strategy for the entire rule with complexity  $k$ , the best-first algorithm will find it after, in the worst case, popping all  $O((3n)^k)$  rules of complexity less than or equal to  $k$  off of the heap in the outer loop, and combining each one with all other  $O((3n)^k)$  such rules in the inner loop, for a total running time of  $O(9^k n^{2k})$ . While the algorithm is still exponential in the rule length  $n$  in the worst case (when  $k$  is linearly correlated to  $n$ ), the best-first behavior makes it much more practical for our empirically observed rules.

*Adding One Nonterminal at a Time is Not Optimal* One might wonder whether it is necessary to consider all combinations of all subsets of nonterminals, or whether an optimal parsing strategy can be found by adding one nonterminal at a time to an existing subset of nonterminals until the entire permutation has been covered. Were such an assumption warranted, this would enable an  $O(n2^n)$  dynamic programming algorithm. It turns out that one-at-a-time parsing strategies are sometimes not optimal. For example, the permutation (4, 7, 3, 8, 1, 6, 2, 5), shown in Figure 14, can be parsed in time  $O(|w|^8)$  using unconstrained subsets, but only in time  $O(|w|^{10})$  by adding one nonterminal at a time. All permutations of less than eight elements can be optimally parsed by adding one element at a time.

---

**Algorithm 3** Best-first search for the optimal parsing strategy
 

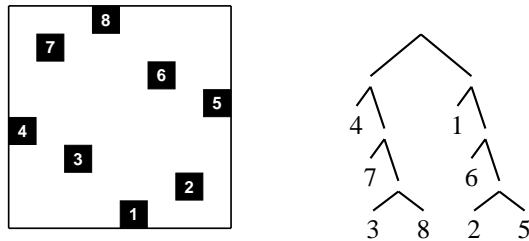
---

```

1: function BESTDISCONTINUOUSPARSER( $S \rightarrow X_1 \dots X_n$ )
2:   for  $A \subset \{X_1 \dots X_n\}$  do
3:      $chart(A) = \infty$ 
4:   for  $i \leftarrow 1 \dots n$  do
5:      $push(heap, 0, \{X_i\})$  ▷ Priority queue of good subsets
6:   while  $chart(\{X_1, \dots, X_n\}) = \infty$  do
7:      $B \leftarrow pop(heap)$ 
8:      $chart(B) \leftarrow best(B)$  ▷ guaranteed to have found optimal analysis for subset  $B$ 
9:     for  $C$  s.t.  $B \cap C = \emptyset \wedge chart(C) < \infty$  do
10:       $A \leftarrow B \cup C$ 
11:      Let  $a, b,$  and  $c$  denote the number of  $A, B,$  and  $C$ 's spans
12:       $compl(A \rightarrow BC) = \max \{a + b + c, best(B), best(C)\}$ 
13:      if  $compl(A \rightarrow BC) < best(A)$  then
14:         $best(A) \leftarrow compl(A \rightarrow BC)$ 
15:         $rule(A) \leftarrow A \rightarrow BC$ 
16:         $push(heap, best(A), A)$ 
17:   return  $best(\{X_1 \dots X_n\})$ 

```

---



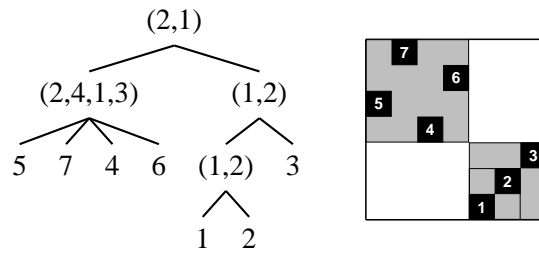
**Figure 14**

The permutation (4, 7, 3, 8, 1, 6, 2, 5) cannot be efficiently parsed by adding one nonterminal at a time. The optimal grouping of nonterminals is shown on the right.

### 6.3 Discontinuous Parsing is Necessary Only for Non-decomposable Permutations

In this subsection, we show that an optimal parsing strategy can be found by first factoring an SCFG rule into a sequence of shorter SCFG rules, if possible, and then considering each of the new rules independently. The first step can be done efficiently using the algorithms of Zhang and Gildea (2007). The second step can be done in time  $O(9^{k_c} \cdot n^{2k_c})$  using Algorithm 3, where  $k_c$  is the complexity of the longest SCFG rule after factorizations, implying that  $k_c \leq (n + 4)$ . Below we show that this two-step process is optimal, by proving that the optimal parsing strategy for the initial rule will not need to build subsets of children that cross the boundaries of the factorization into shorter SCFG rules.

Figure 15 shows a permutation that contains permutations of fewer numbers within itself so that the entire permutation can be decomposed hierarchically. We prove that if there is a contiguous block of numbers that are permuted within a permutation, the optimal parsing strategy for the entire permutation does not have to involve interactions



**Figure 15**  
A permutation that be decomposed into smaller permutations hierarchically.

between subsets of numbers inside and outside the block. We call filled entries in the permutation matrix *pebbles*; the contiguous blocks are shaded in Figure 15, and form submatrices with a pebble in each row and column. We can first decompose a given permutation into a hierarchy of smaller permutations as the tree shown in Figure 15 and then apply the discontinuous strategy to the non-decomposable permutations in the tree. So, in this example, we just need to focus on the optimal parsing strategy for  $(2, 4, 1, 3)$ , which is applied to permute  $(4, 5, 6, 7)$  into  $(5, 7, 4, 6)$ . By doing this kind of minimization, we can effectively reduce the search space without losing optimality of the parsing strategy for the original permutation.

**Theorem 4.** For any SCFG rule, if there exists a recursive partition of child nonterminals which enables tabular parsing of an input sentence  $w$  in time  $O(|w|^k)$ , and if  $S$  is subset of child nonterminals forming a single continuous span in each language, then there exists a recursive partition containing  $S$  as a member whose corresponding parser is also  $O(|w|^k)$ .

See Appendix for the proof.

### 6.4 A CKY-style Algorithm for Parsing Strategies

Because Algorithm 3 is exponential in  $n$  in the worst case, it is not practical for very large rules. However, the  $O(n^3)$  algorithm we present in this section can find good factorizations for most non-binarizable rules. This algorithm, shown in Algorithm 4, considers only factorizations that have only one span in on of the two languages, and efficiently searches over all such factorizations by combining adjacent spans with CKY-style parsing.<sup>4</sup>

While this CKY-style algorithm finds the best grammar factorization maintaining continuous spans in one of the two dimensions, in general the best factorization may require discontinuous spans in both dimensions. As an example, the following pattern

<sup>4</sup> A special case of this algorithm, the *target-side binarization*, is discussed in (Huang, 2007). It binarizes left-to-right on the target-side while leaving gaps on the source-side, and is shown to be preferable to source-side monolingual binarization in  $m$ -gram integrated decoding.

---

**Algorithm 4** An  $O(n^3)$  CKY-style algorithm for parsing strategies, keeping continuous spans in one language

---

```

1: function BESTCONTINUOUSPARSER( $S \rightarrow X_1 \dots X_n$ )
2:   for  $span \leftarrow 1$  to  $n - 1$  do
3:     for  $i \leftarrow 1$  to  $n - span$  do
4:        $A = \{X_i \dots X_{i+span}\}$ 
5:        $best(A) = \infty$ 
6:       for  $j \leftarrow i + 1$  to  $i + span$  do
7:          $B = \{X_i \dots X_{j-1}\}$ 
8:          $C = \{X_j \dots X_{i+span}\}$ 
9:          $compl(A \rightarrow BC) = \max\{a_f + b_f + c_f, best(B), best(C)\}$ 
10:        if  $compl(A \rightarrow BC) < best(A)$  then
11:           $best(A) = compl(A \rightarrow BC)$ 
12:           $rule(A) = A \rightarrow BC$ 
13:   return  $best(\{X_1 \dots X_n\})$ 

```

---

Algorithm	Assumptions of the Strategy Space	Complexity
Algorithm 1 (synchronous)	Contiguous on both sides	$O(n)$
Algorithm 2 (optimal)	No assumptions	$O(3^n)$
Algorithm 3 (best-first)		$O(9^k n^{2k})$
Algorithm 4 (CKY)	Contiguous on one side	$O(n^3)$

---

**Table 2**

A summary of the four binarization algorithms. Algorithms 2 and 3 can be improved by first factorizing the permutation into smaller permutations (Section 6.3).

---

causes problems for the algorithm regardless of which of the dimensions it parses across:

$$\begin{array}{llll}
1 & \leftrightarrow & 1 & \quad \quad \quad n/2 + 1 & \leftrightarrow & 2 \\
2 & \leftrightarrow & n/2 + 1 & \quad \quad \quad n/2 + 2 & \leftrightarrow & n/2 + 2 \\
3 & \leftrightarrow & 3 & \quad \quad \quad n/2 + 3 & \leftrightarrow & 4 \\
4 & \leftrightarrow & n/2 + 3 & \quad \quad \quad n/2 + 4 & \leftrightarrow & n/2 + 4 \\
\dots & & & & & \dots
\end{array}$$

This pattern, shown graphically in Figure 12 for  $n = 16$ , can be parsed in time  $O(|w|^{10})$  by maintaining a partially completed item with two spans in each dimension, one beginning a position 1 and one beginning at position  $\frac{n}{2} + 1$ , and adding one nonterminal at a time to the partially completed item, as shown in Figure 12(right). However, our CKY factoring algorithm will give a factoring with  $n/2$  discontinuous spans in one dimension. Thus in the worst case, the number of spans found by the cubic-time algorithm grows with  $n$ , even when a constant number of spans is possible, implying that there is no approximation ratio on how close the algorithm will get to the optimal solution.

Table 2 summarizes the four binarization algorithms presented in this paper.

Complexities	Synchronous Parsing /Bigram Decoding		Trigram Decoding (Chinese to English)		Trigram Decoding (English to Chinese)	
	<i>opt</i>	<i>cky-min</i>	<i>opt</i>	<i>cky-min</i>	<i>opt</i>	<i>cky-min</i>
19						1
18				1		
17			1		1	
16						
15		1	7	10	3	4
14			4	3	6	6
13			2240	2238	1080	1079
12	1	1	610	610	548	548
11			154,350	154,350	155,574	155,574
10	68	156				
9	101	307				
8	157,042	156,747				

**Table 3**

The distribution of parsing complexities of non-binarizable rules extracted from the GIZA-aligned Chinese-English data in Section 5. The first column denotes the exponent of the time complexity, e.g., 10 means  $O(|w|^{10})$ . *opt* denotes the optimal parsing strategy and *cky-min* denotes the approximation strategy that takes the better of CKY results on both sides.

## 6.5 Experiments

The combination of minimizing SCFG rule length as a preprocessing step and then applying the best-first version of Algorithm 2 makes it possible to find optimal parsing strategies for all of the rules in the large Chinese-English rule set used for our decoding experiments. For the 157,212 nonbinarizable rules, the complexity of the optimal parsing strategies is shown in Table 3. While the worst parsing complexity is  $O(|w|^{12})$ , this is only achieved by a single rule. The best-first analyzer takes approximately five minutes of CPU time to analyze this single rule, but processes all others in less than one second.

We tested the CKY-based factorization algorithm on our set of nonbinarizable rules extracted from Chinese-English data. The CKY-on-English method found an optimal parsing strategy for 98% of the rules, and its worst-case complexity over the entire rule-set was  $O(|w|^{15})$ , rather than the optimal  $O(|w|^{12})$ . If we run CKY factorization from two directions (one for the permutation  $\pi$  and the other for the permutation  $\pi^{-1}$ ) and take the minimum of both, we can get an even better approximation. In Table 3, we compare the approximate strategy which takes the minimum of CKY runs for two languages with the optimal strategy. For synchronous parsing, for 99.77% of the rules, the CKY-min method found an optimal strategy. When generalized for  $m$ -gram integrated decoding, CKY maintains continuous spans on the output language and allows for discontinuous parsing on the input sentence. The difference between CKY-on-output and the optimal

decoding strategy was negligible in the situation of trigram-integrated decoding for the given rules. The worst-case complexity for decoding into English by CKY-on-English was  $O(|w|^{18})$ , versus  $O(|w|^{17})$  from the optimal strategy. The CKY-on-English approach found an optimal decoding strategy for 99.97% of the nonbinarizable rules. The CKY-min strategy was even better, only found sub-optimal results for 6 rules out of all rules, which translates to 99.996%. In Table 3, we also included the comparison for translating into Chinese, in which case, the inverted permutations are used and the language model weight is put on the Chinese side. A similar approximation accuracy was achieved.

### 6.6 Bounds on Complexity of Factorization

Given that our algorithms for optimal factorization are exponential, it is natural to ask whether the problem is provably NP-complete. Gildea and Štefanković (2007) relate the problem of finding the optimal parsing strategy for a rule to computing the *treewidth* of a graph derived from the rule's permutation. Computing treewidth of arbitrary graphs is NP-complete (Arnborg, Corneil, and Proskurowski, 1987), but the graphs derived from SCFG permutations have a restricted structure that it might be possible to exploit. In particular, the graphs have degree no greater than six. While computing treewidth for graphs of bounded degree nine was shown to be NP-complete by Bodlaender and Thilikos (1997), whether the treewidth problem for graphs of degree between three and eight is NP-complete is not known. Thus, whether computing the optimal parsing strategy for an SCFG rule is NP-complete remains an interesting open problem.

## 7. Conclusion

This work develops a theory of binarization for synchronous context-free grammars. We presents a technique called *synchronous binarization* along with an efficient binarization algorithm. Empirical study shows that the vast majority of syntactic reorderings, at least between languages like English and Chinese, can be efficiently decomposed into hierarchical binary reorderings. As a result, decoding with  $n$ -gram models can be fast and accurate, making it possible for our syntax-based system to overtake a comparable phrase-based system in BLEU score.

There are, however, some interesting rules that are not binarizable and we provide, for the first time, real examples verified by native speakers. For these remaining rules, we have shown an exponential time algorithm for finding optimal parsing strategies, which runs quite fast with the help of two optimality-maintaining operations and the A\* search strategy. We also provide an efficient approximation, which usually finds optimal parsing strategies in practice.

As for future work, we think it is interesting to explore the extensions of our techniques to more powerful models such as synchronous tree-adjointing grammars (Shieber and Schabes, 1990).

## References

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

- Arnborg, Stefen, Derek G. Corneil, and Andrzej Proskurowski. 1987. Complexity of finding embeddings in a  $k$ -tree. *SIAM Journal of Algebraic and Discrete Methods*, 8:277–284, April.
- Bodlaender, H. L. and D. M. Thilikos. 1997. Treewidth for graphs with small chordality. *Discrete Applied Mathematics*, pages 45–61.
- Catalan, Eugène. 1844. Note extraite d’une lettre adressée. *J. Reine Angew. Math.*, 27:192.
- Chiang, David. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Conference of the Association for Computational Linguistics (ACL-05)*, pages 263–270.
- Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. 1990. *Introduction to algorithms*. MIT Press, Cambridge, MA.
- Eisner, Jason. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics, companion volume*, Sapporo, Japan.
- Galley, Michel, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What’s in a translation rule? In *Proceedings of the Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*.
- Gildea, Daniel and Daniel Štefanković. 2007. Worst-case synchronous grammar rules. In *Proceedings of the 2007 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-07)*.
- Graham, Ronald. 1972. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133.
- Huang, Liang. 2007. Binarization, synchronous binarization, and target-side binarization. In *Proceedings of the NAACL/AMTA Workshop on Syntax and Structure in Statistical Translation (SSST)*, Rochester, NY.
- Huang, Liang and David Chiang. 2005. Better  $k$ -best parsing. In *International Workshop on Parsing Technologies (IWPT05)*, Vancouver, BC.
- Huang, Liang, Hao Zhang, and Daniel Gildea. 2005. Machine translation as lexicalized parsing with hooks. In *International Workshop on Parsing Technologies (IWPT05)*, Vancouver, BC.
- ISI Machine Translation Team. 2006. ISI at NIST-06. Working Notes of the NIST MT Evaluation Workshop, September. Washington, D.C.
- Knight, Kevin and Jonathan Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In *Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*. Lecture Notes in Computer Science (Springer-Verlag).
- Knuth, D. 1977. A generalization of Dijkstra’s algorithm. *Info. Proc. Letters*, 6(1).
- Liu, Yang, Qun Liu, and Shouxun Lin. 2005. Log-linear models for word alignment. In *Proceedings of the 43rd Annual Conference of the Association for Computational Linguistics (ACL-05)*, Ann Arbor, Michigan.
- Melamed, I. Dan. 2003. Multitext grammars and synchronous parsers. In *Proceedings of the 2003 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-03)*, Edmonton.
- Och, Franz Josef and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4).
- Rounds, William C. 1970. Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287.
- 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 (HLT/EMNLP)*, pages 803–810, Vancouver, Canada, October.
- Schröder, E. 1870. Vier combinatorische Probleme. *Zeitschrift für Mathematik und Physik*, 15:361–376.
- Shapiro, L. and A. B. Stephens. 1991. Bootstrap percolation, the Schröder numbers, and the  $n$ -kings problem. *SIAM Journal on Discrete Mathematics*, 4(2):275–280.
- Shieber, Stuart and Yves Schabes. 1990. Synchronous tree-adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*, volume III, pages 253–258.
- Shieber, Stuart M. 2004. Synchronous grammars as tree transducers. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*.
- Wellington, Benjamin, Sonjia Waxmonsky, and I. Dan Melamed. 2006. Empirical lower bounds on the complexity of translational equivalence. In *Proceedings of the International Conference on Computational Linguistics / Association for Computational Linguistics (COLING/ACL-06)*.
- Wu, Dekai. 1996. A polynomial-time algorithm for statistical machine translation. In *34th Annual Meeting of the Association for Computational Linguistics*.
- Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403.
- Zhang, Hao and Daniel Gildea. 2007. Factorization of synchronous context-free grammars in linear time. In *NAACL Workshop on Syntax and Structure in Statistical Translation (SSST)*.

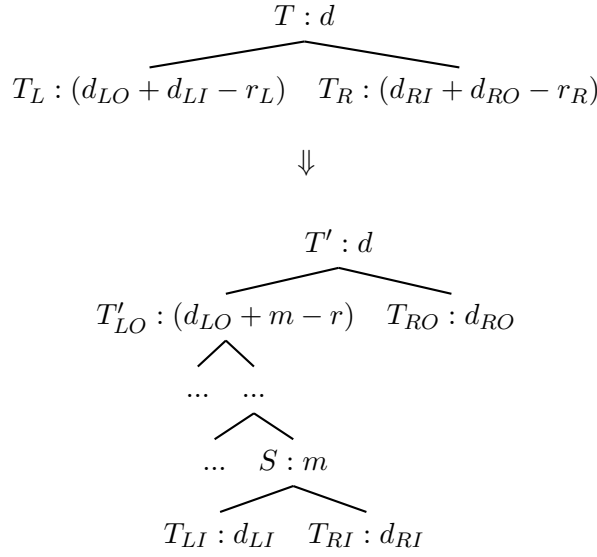
#### A. Proof of Theorem 4

*Proof.* We prove by contradiction. Let us suppose that the optimal parsing strategy for a permutation  $P$  splits a contiguous block  $S$  of  $P$  into two subtrees  $T_L$  and  $T_R$ , as shown on the top of Figure 16, in either or both of which there are some pebbles from outside  $S$ . As in the previous section, we count spans in this section using the weighted span value of Equation 11 to account for  $m$ -gram language model state. We use  $d_{LO}$  to denote the number of spans of the pebbles outside of  $S$  in  $T_L$ .  $d_{LI}$  is the number of spans of the pebbles inside  $S$  for  $T_L$ . We use  $r_L$  to denote the reduction in the number of spans achieved by merging the pebbles inside and outside of  $S$  for  $T_L$ . So, the number of spans of the root of  $T_L$  is  $d_{LO} + d_{LI} - r_L$ . We have symmetric notions for  $T_R$ . We use  $d$  to denote the number of spans of the root of the subtree  $T$  after merging  $T_L$  and  $T_R$ . The number of spans is annotated for each node in Figure 16.

Notice that  $(r_R + r_L) \leq 2m$  because there are at most two boundaries shared by inside pebbles and outside pebbles in each language. Each boundary in source corresponds to reduction of one span. Each boundary in target corresponds to reduction of one weighted span of  $(m - 1)$ . In total, we can reduce the number of (weighted) spans by no more than  $2(m - 1) + 2 = 2m$ . This inequality implies either  $r_R \leq m$  or  $r_L \leq m$ . Without loss of generality, we assume

$$r_R \leq m \tag{15}$$

Given the decomposition into  $T_L$  and  $T_R$ , the yield of the best strategy throughout



**Figure 16**  
Reorganization of a parsing strategy to build a continuous span  $S$  first.

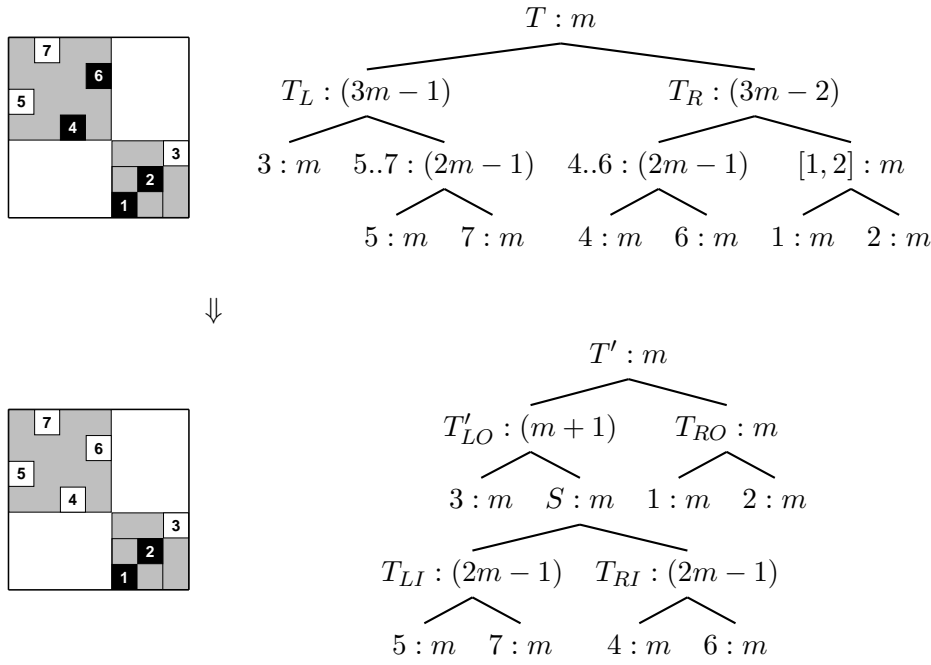
$T$  is:

$$best(T) = \max \{ best(T_L), best(T_R), ((d_{LO} + d_{LI} - r_L) + (d_{RI} + d_{RO} - r_R) + d) \} \tag{16}$$

Figure 17 gives a concrete example. The permutation is (5, 7, 4, 6, 1, 2, 3). The block  $S$  we focus on is (5, 7, 4, 6). The original strategy on the top of the figure splits the block into  $T_L$  and  $T_R$ . The improved strategy on the bottom merges the pebbles inside  $S$  together before making combinations with pebbles outside  $S$ .

In general, we argue that we can have an equally good or better strategy by separating each of  $T_L$  and  $T_R$  into two trees involving pebbles purely inside or outside of  $S$ , as shown on the bottom of Figure 16. The separation works by simply ignoring the pebbles that are not inside when creating the inside half of the tree or outside when doing the outside half throughout  $T_L$  and  $T_R$ . Then we have four elementary subtrees  $T_{LO}$ ,  $T_{LI}$ ,  $T_{RI}$ , and  $T_{RO}$ . In our new strategy, we recombine the four elementary trees by merging  $T_{LI}$  and  $T_{RI}$  to create a pebble first and merging the resulting pebble back into  $T_{LO}$  to make a  $T'_{LO}$ , and finally merging  $T'_{LO}$  with  $T_{RO}$ .

The elementary trees yield better strategies because the number of spans of each node in these trees is reduced or not changed as compared to that before separation. Using the “a-plus-b-plus-c” formula with reduced  $a$ ,  $b$ , and  $c$  will produce lower complexity. Roughly speaking, the reason is the inside pebbles and outside pebbles are positioned side by side instead of mixed together. Mathematically, the reduction of spans by combining both sides is upper-bounded by  $2m$ , considering there are two boundaries in each language. At the same time, the number of spans of either the inside pebbles or the outside pebbles is lower-bounded by  $2m$  because both  $T_L$  and  $T_R$  only partially cover



**Figure 17**

An actual example of reorganization of a parsing strategy to build a continuous span  $S$  first. Before, the overall strategy cost is  $(7m - 3)$ . After, the cost is  $(5m - 2)$ . ( $m \geq 2$ ) We use black to represent pebbles in the right branch of the root node and white for the left branch. Gray areas are continuous blocks within the permutation. The reorganized strategy can be further improved by making another such transformation to allow for the lower right corner pebbles to group before interacting with the upper left corner.

$S$ . Hence, we have the following set of inequalities:

$$\text{best}(T_{LI}) \leq \text{best}(T_L) \quad (17)$$

$$\text{best}(T_{RI}) \leq \text{best}(T_R) \quad (18)$$

$$\text{best}(T_{RO}) \leq \text{best}(T_R) \quad (19)$$

Now we consider what happens when the pebble of  $S$  joins  $T_{LO}$ . Since  $T_{LO}$  is created from  $T_L$  by pruning away the pebbles that are inside  $S$ , the pebble of  $S$  can join  $T_{LO}$  by taking place of any trace of the pruned leaves and make the number of spans from the bottom up to the root no greater than in the counterpart nodes in  $T_L$ .

So the fragment of the new left subtree  $T'_{LO}$  with  $S$  being its leaf has a better yield than the original  $T_L$ :

$$\text{best}(T'_{LO}/S) \leq \text{best}(T_L) \quad (20)$$

where we use  $T'_{LO}/S$  to denote the tree fragment excluding the nodes under  $S$ . The number of spans for each node in the reorganized tree is shown in Figure 16 (bottom),

where  $r$  ( $\leq 2m$ ) is the reduction in spans after combining the new pebble  $S$  with  $T_{LO}$ .  $r$  sums up the reductions achievable on the four boundaries of  $S$  with  $T_{LO}$ , while  $r_L$  sums up the reductions on some of the four boundaries. Thus,

$$r_L \leq r \quad (21)$$

The final yield of the updated strategy is:

$$best(T') = \max \left\{ \begin{array}{l} best(T'_{LO}/S), best(T_{LI}), best(T_{RI}), best(T_{RO}), \\ (d_{LI} + d_{RI} + m), \\ ((d_{LO} + m - r) + d_{RO} + d) \end{array} \right\} \quad (22)$$

We have shown the first four terms inside the maximization are bounded by the yield of the old strategy (Equation 17, Equation 18, Equation 19, Equation 20). We need to bound the remaining terms. Both of them can be bounded by the third term inside the maximization of Equation 16. The first inequality is:

$$d_{LI} + d_{RI} + m \leq (d_{LO} + d_{LI} - r_L) + (d_{RI} + d_{RO} - r_R) + d \quad (23)$$

which is equivalent to

$$m \leq (d_{LO} + d_{RO}) - (r_L + r_R) + d$$

which is true because  $d \geq m$  (since  $T$  has at least one pebble), and  $d_{LO} \geq r_L$  and  $d_{RO} \geq r_R$  (since the number of reducible spans is less than the total number of outside spans).

The other bounding inequality is

$$((d_{LO} + m - r) + d_{RO} + d) \leq ((d_{LO} + d_{LI} - r_L) + (d_{RI} + d_{RO} - r_R) + d) \quad (24)$$

equivalently

$$m - r \leq d_{LI} + d_{RI} - r_R - r_L$$

because  $m \leq d_{LI}$  meaning there is at least one inside pebble in  $T_L$ , and  $d_{RI} \geq r_R$  because  $d_{RI} \geq m \geq r_R$ , referring to Equation 15, and finally  $r \geq r_L$  which has been shown in Equation 21.

Figure 17 also demonstrates the re-distribution of numbers of spans after the reorganization. In the example, the updated parsing/decoding complexity is  $O(|w|^{5m-2})$ , better than before ( $O(|w|^{7m-3})$ ).

Therefore, any synchronous parsing/decoding strategy that crosses decomposition boundaries can not be better than an optimized strategy that respects such boundaries.  $\square$