Binarization of Synchronous Context-Free Grammars

Liang Huang*
USC/Information Science Institute

Hao Zhang**
Google Inc.

Daniel Gildea†
University of Rochester

Kevin Knight‡
USC/Information Science 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 as a technique to factor each synchronous grammar rule into a series of binary rules. Although monolingual context-free grammars (CFGs) can always be binarized, this is not the case

* Information Science Institute, 4676 Admiralty Way, Marina del Rey, CA 90292. E-mail: lhuang@isi.edu, liang.huang.sh@gmail.com.
** 1600 Amphitheatre Parkway, Mountain View, CA 94303. E-mail: haozhang@google.com.
† Computer Science Dept., University of Rochester, Rochester NY 14627. E-mail: gildea@cs.rochester.edu.
‡ Information Science Institute, 4676 Admiralty Way, Marina del Rey, CA 90292. E-mail: knight@isi.edu.

Submission received: 14 March 2007; revised submission received: 8 January 2009; accepted for publication: 25 March 2009.

for all synchronous rules; we investigate algorithms for non-binarizable rules as well. 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 Chinese-to-English 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.
- In the final, theoretical, sections of this article, we investigate the general problem of finding the most efficient synchronous parsing or decoding strategy for arbitrary synchronous context-free grammar (SCFG) rules, including non-binarizable cases. Although 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 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[1] show that binarization improves machine translation speed and quality. Some rules cannot be binarized, and we present a decoding strategy for these rules in Section 6. Section 7 gives a solution to the general theoretical problem of finding optimal decoding and synchronous parsing strategies for arbitrary SCFGs, and presents complexity results on the nonbinarizable rules from our Chinese–English data. These final two sections are of primarily theoretical interest, as nonbinarizable rules have not been shown to benefit real-world machine translation systems. However, the algorithms presented may become relevant as machine translation systems improve.

1 A preliminary version of Section 1–5 appeared in Zhang et al. (2006).

2. Motivation

Consider the following Chinese sentence and its English translation:

(1)

鲍威尔	与	沙龙	举行	了	会谈
Bàowēier	*yǔ*	*Shālóng*	*jǔxíng*	*le*	*huìtán*
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	→	NP[1] PP[2] VP[3],	NP[1] VP[3] PP[2]
NP	→	鲍威尔 / *Bàowēier*,	Powell
VP	→	举行 了 会谈 / *jǔxíng le huìtán*,	held a meeting
PP	→	与 沙龙 / *yǔ Shālóng*,	with Sharon

Decoding can be cast as a (monolingual) parsing problem because we only need to parse the source-language side of the SCFG, as if we were constructing a CFG by projecting the SCFG onto its Chinese side:

(3)

S	→	NP PP VP
NP	→	鲍威尔 / *Bàowēier*
VP	→	举行 了 会谈 / *jǔxíng le huìtán*
PP	→	与 沙龙 / *yǔ Shālóng*

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 following discussion 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):

S	→	NP-PP VP	or	S	→	NP PP-VP
NP-PP	→	NP PP		PP-VP	→	PP VP

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

Figure 1
A pair of synchronous parse trees in the SCFG (2). The superscript symbols ($^{\diamond\star\circ\bullet}$) indicate pairs of synchronous nonterminals (and subtrees).

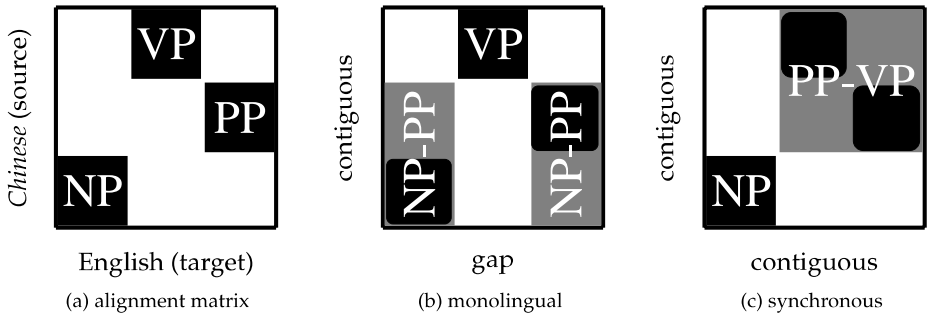


Figure 2
The alignment matrix and two binarization schemes, with virtual nonterminals in gray. (a) A two-dimensional matrix representation of the first SCFG rule in grammar 2. Rows are positions in Chinese: columns are positions in English, and black cells indicate positions linked by the SCFG rule. (b) This scheme groups NP and PP into an intermediate state which contains a gap on the English side. (c) This scheme groups PP and VP into an intermediate state which is contiguous on both sides.

These two binarizations are no different in the translation-model-only decoding described previously, 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 has been shown to be very important for translation quality (Chiang 2005). To do bigram-integrated decoding (Wu 1996), we need to augment each chart item (X, i, j) with two target-language **boundary words** u and v to produce a bigram-item which we denote $\binom{u \cdots v}{i \quad \mathrm{X} \quad j}$.[2]

Now the two binarizations have very different effects. In the first case, we first combine NP with PP. This step is written as follows in the weighted deduction notation of Nederhof (2003):

$$\frac{\left(\begin{smallmatrix} \text{Powell} & \cdots & \text{Powell} \\ 1 & \text{NP} & 2 \end{smallmatrix}\right) : p \qquad \left(\begin{smallmatrix} \text{with} & \cdots & \text{Sharon} \\ 2 & \text{PP} & 4 \end{smallmatrix}\right) : q}{\left(\begin{smallmatrix} \text{Powell} \cdots \text{Powell} & \cdots & \text{with} \cdots \text{Sharon} \\ 1 & \text{NP-PP} & 4 \end{smallmatrix}\right) : pq}$$

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 $\binom{\text{held} \cdots \text{meeting}}{2 \quad \text{VP} \quad 4}$ to form an S item. We call this binarization method **monolingual binarization** because 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

2 An alternative to integrated decoding is **rescoring**, where one first computes the k-best translations according to the TM only, and then reranks 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 because the true best translation is often outside of the k-best list, especially for longer sentences.

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, we have:

$$\frac{\begin{pmatrix} \text{with} & \cdots & \text{Sharon} \\ 2 & \text{PP} & 4 \end{pmatrix} : r \qquad \begin{pmatrix} \text{held} & \cdots & \text{meeting} \\ 4 & \text{VP} & 7 \end{pmatrix} : s}{\begin{pmatrix} \text{held} & \cdots & \text{Sharon} \\ 2 & \text{PP-VP} & 7 \end{pmatrix} : rs \cdot \Pr(\text{with} \mid \text{meeting})}$$

Here, because PP and VP are contiguous (but swapped) in the target language, we can include the language model score by multiplying in $\Pr(\text{with} \mid \text{meeting})$, and the resulting item again has two boundary words. Later we multiply in $\Pr(\text{held} \mid \text{Powell})$ when the resulting item is combined with $\begin{pmatrix} \text{Powell} & \cdots & \text{Powell} \\ 1 & \text{NP} & 2 \end{pmatrix}$ 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:

$$\begin{array}{rcll} \text{S} & \rightarrow & \text{NP}\boxed{1}\ \text{PP-VP}\boxed{2}, & \text{NP}\boxed{1}\ \text{PP-VP}\boxed{2} \\ \text{PP-VP} & \rightarrow & \text{VP}\boxed{1}\ \text{PP}\boxed{2}, & \text{PP}\boxed{2}\ \text{VP}\boxed{1} \end{array} \tag{4}$$

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.

Some recent syntax-based MT systems (Galley et al. 2004) have adopted the formalism of tree transducers (Rounds 1970), modeling translation as a set of rules for a transducer that takes a syntax tree in one language as input and transforms it into a tree (or string) in the other language. The same decoding algorithms are used for machine translation in this formalism, and the following example shows that the same issues of binarization arise.

Suppose we have the following transducer rules:

$$\begin{array}{lcl} \text{S}(x_1\text{:NP}\ x_2\text{:PP}\ x_3\text{:VP}) & \rightarrow & \text{S}(x_1\ \text{VP}(x_3\ x_2)) \\ \text{NP}(\text{鲍威尔}\ /\ \textit{Bàowēier}) & \rightarrow & \text{NP}(\text{NNP}(\text{Powell})) \\ \text{VP}(\text{举行}\ \text{了}\ \text{会谈}\ /\ \textit{jǔxíng le huìtán}) & \rightarrow & \text{VP}(\text{VBD}(\text{held})\ \text{NP}(\text{DT}(\text{a})\ \text{NPS}(\text{meeting}))) \\ \text{PP}(\text{与}\ \text{沙龙}\ /\ \textit{yǔ Shālóng}) & \rightarrow & \text{PP}(\text{TO}(\text{with})\ \text{NP}(\text{NNP}(\text{Sharon}))) \end{array} \tag{5}$$

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, learning, for example, 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 better parameter estimation. By creating a

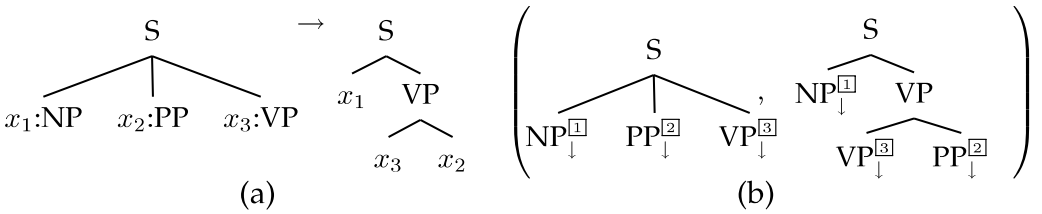


Figure 3
Two equivalent representations of the first rule in Example (5): (a) tree transducer; (b) Synchronous Tree Subsitution Grammar (STSG). The ↓ arrows denote substitution sites, which correspond to variables in tree transducers.

nonterminal for each right-hand-side tree, we can convert the transducer representation to an SCFG with the same generative capacity. We can again create a projected CFG which will be exactly the same as in Example (3), and build English subtrees 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.

In SCFG-based frameworks, the problem of finding a word-level alignment between two sentences is an instance of the synchronous parsing problem: Given two strings and a synchronous grammar, find a parse tree that generates both input strings. The benefit of binary grammars also applies in this case. 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 $O(|w|^8)$, because we have to maintain four indices on either side, giving a total of eight; parsing the monolingually binarized grammar (b) involves seven indices, three on the Chinese side and four 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." An efficient alignment algorithm is guaranteed if a binarization is found, and the same binarization can be used for decoding and alignment. We show how to find optimal alignment algorithms for non-binarizable rules in Section 7; in this case different grammar factorizations may be optimal for alignment and for decoding with n-gram models of various orders. Handling difficult rules may in fact be more important for alignment than for decoding, because although we may be able to find good translations during decoding within the space permitted by computationally friendly rules, during alignment we must handle the broader spectrum of phenomena found in real bitext data.

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 α and the nonterminal occurrences in β. Each co-indexed child nonterminal pair is a pair of synchronous nonterminals and will be further rewritten as a unit.

Note that this notation, due to Satta and Peserico (2005), 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

(6) $\text{VP} \rightarrow \text{VB}^{\boxed{1}}\ \text{NN}^{\boxed{2}},\ \ \text{VP} \rightarrow \text{VBZ}^{\boxed{1}}\ \text{NNS}^{\boxed{2}}$

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:

$$(\text{VP, VP}) \rightarrow (\text{VB, VBZ})^{\boxed{1}}\ (\text{NN, NNS})^{\boxed{2}},\ \ (\text{VB, VBZ})^{\boxed{1}}\ (\text{NN, NNS})^{\boxed{2}}.$$

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:

$$X \rightarrow X_1^{\boxed{1}} \ldots X_n^{\boxed{n}},\ Y \rightarrow Y_{\pi(1)}^{\boxed{\pi(1)}} \ldots Y_{\pi(n)}^{\boxed{\pi(n)}}$$

where X_i and Y_i are variables ranging over nonterminals in the source and target projections of the synchronous grammar, respectively, and π is the **permutation** of the rule. For example, in rule (6), we have $n = 2$, $X = Y = \text{VP}$, $X_1 = \text{VB}$, $X_2 = \text{NN}$, $Y_1 = \text{VBZ}$, $Y_2 = \text{NNS}$, and π is the identity permutation. Note that this general notation includes cases where a nonterminal occurs more than once in the right-hand side, for example, when $n = 2$, $X = Y = A$, and $X_1 = X_2 = Y_1 = Y_2 = B$, we can have the following two rules:

$$A \rightarrow B^{\boxed{1}}\ B^{\boxed{2}},\ A \rightarrow B^{\boxed{2}}\ B^{\boxed{1}};$$
$$A \rightarrow B^{\boxed{1}}\ B^{\boxed{2}},\ A \rightarrow B^{\boxed{1}}\ B^{\boxed{2}}.$$

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.

Definition 2
A **permuted sequence** is a permutation of consecutive integers. If a permuted sequence **a** can be split into the concatenation of two permuted sequences **b** and **c**, then $(\mathbf{b}; \mathbf{c})$ is called a **proper split** of **a**. We say $\mathbf{b} < \mathbf{c}$ if each element in **b** is smaller than any element in **c**.

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

Lemma 1
For a permuted sequence a, $\mathbf{a} = (\mathbf{b}; \mathbf{c})$ is a proper split 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 sequences, $(\mathbf{b}; \mathbf{c})$ is a proper split. The case when $\mathbf{b} > \mathbf{c}$ is similar. ■

Definition 3
A permuted sequence $\mathbf{a}$ is said to be **binarizable** if either

1. $\mathbf{a}$ is a singleton, that is, $\mathbf{a} = (a)$, 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 now rigorously define.

Definition 4
A **binarization tree** $bi(\mathbf{a})$ of a binarizable sequence $\mathbf{a}$ is either

1. a if $\mathbf{a} = (a)$, or
2. $[bi(\mathbf{b}), bi(\mathbf{c})]$ if $\mathbf{b} < \mathbf{c}$, or $\langle bi(\mathbf{b}), bi(\mathbf{c}) \rangle$ 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 $\mathbf{c}$.

Here we use [] 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**, then, is 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 π, there is a binarization tree $bi(\pi)$ which essentially

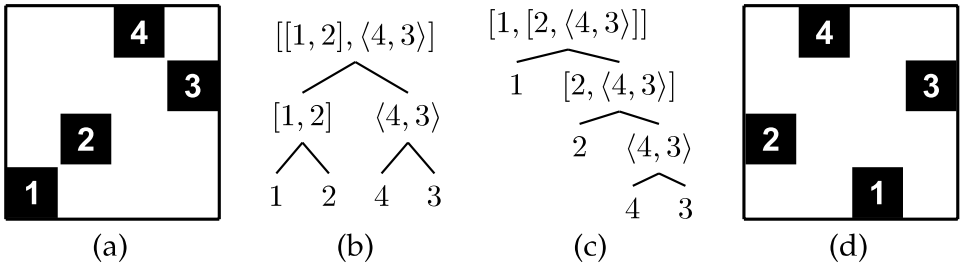


Figure 4
(a)–(c) Alignment matrix and two binarization trees for the permutation sequence (1, 2, 4, 3). (d) Alignment matrix for the non-binarizable sequence (2, 4, 1, 3).

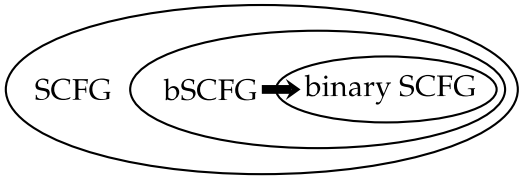


Figure 5
Subclasses of SCFG. The thick arrow denotes the direction of synchronous binarization and indicates bSCFG can collapse to binary SCFG.

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 $(\pi(1)...\pi(n))$. 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 $O(n^4)$ because there is no insertion or deletion in a permutation.

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, Section 7) that prefers left-heavy binarization trees so that for each permutation there is a unique synchronous derivation.[3] 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, whereas $[1, [2, \langle 4, 3 \rangle]]$ 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, page 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 because 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 tree, there

3 We are not aiming at *optimal sharing*, that is, 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 SynchronousBinarizer(π)
2:      top ← 0                                              ▷ stack top pointer
3:      Push(stack, (π(1), π(1)))                            ▷ initial shift
4:      for i ← 2 to |π| do                                  ▷ for each remaining element
5:          Push(stack, (π(i), π(i)))                        ▷ shift
6:          while top > 1 and Consecutive(stack[top], stack[top − 1]) do
7:                                                           ▷ keep reducing if possible
8:              (p, q) ← Combine(stack[top], stack[top − 1])
9:              top ← 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:     A = {min(a, c)...max(b, d)}
17:     B = {a...b}
18:     C = {c...d}
19:     rule[A] = A → B C
20:     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 3-4	2	reduce	$[3, 4]$
	1 3-5	2	reduce	$\langle 5, [3, 4] \rangle$
4	1 3-5 2		shift	
	1 2-5		reduce	$\langle \langle 5, [3, 4] \rangle, 2 \rangle$
	1-5		reduce	$[1, \langle \langle 5, [3, 4] \rangle, 2 \rangle]$

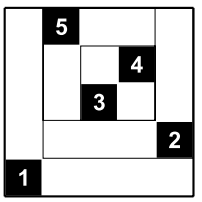


Figure 6
Example of Algorithm 1 on the binarizable input (1, 5, 3, 4, 2). The rightmost column shows the binarization-trees generated at each reduction step.

exists a (binarizable) split which is nearest to the root and splits $\mathbf{c}$ into two parts. Let the split be $(\mathbf{b}_1, \mathbf{c}_1; \mathbf{c}_2, \mathbf{b}_2)$, where $\mathbf{c} = (\mathbf{c}_1; \mathbf{c}_2)$, and either $\mathbf{b}_1$ or $\mathbf{b}_2$ can be empty. By Lemma 1, we have $\mathbf{c}_1 < \mathbf{c}_2$ or $\mathbf{c}_1 > \mathbf{c}_2$. By Lemma 1 again, we have that $(\mathbf{c}_1; \mathbf{c}_2)$ is a proper split of $\mathbf{c}$, i.e., both $\mathbf{c}_1$ and $\mathbf{c}_2$ are themselves permuted sequences. We also know both $(\mathbf{b}_1, \mathbf{c}_1)$ and $(\mathbf{c}_2, \mathbf{b}_2)$ are binarizable. By the induction hypothesis, $\mathbf{c}_1$ and $\mathbf{c}_2$ are both binarizable. So we conclude that $\mathbf{c} = (\mathbf{c}_1; \mathbf{c}_2)$ is binarizable (See figure 8). ■

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 $\mathbf{a}$ is binarizable, in which case the binarization tree recovered is canonical.

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 4-5	1 3	reduce	$\langle 5, 4 \rangle$
4	2 4-5 1	3	shift	
5	2 4-5 1 3		shift	

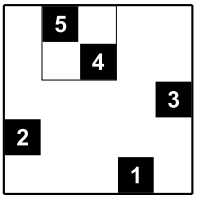


Figure 7
Example of Algorithm 1 on the non-binarizable input $(2, 5, 4, 1, 3)$.

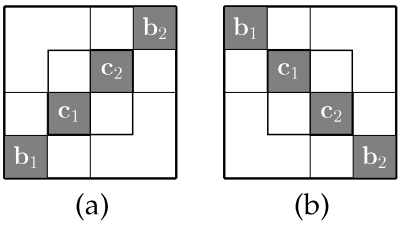


Figure 8
Illustration of the proof of Lemma 2. The arrangement of $(\mathbf{b}_1, \mathbf{c}_1; \mathbf{c}_2, \mathbf{b}_2)$ must be either all straight as in (a) or all inverted as in (b).

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 $\mathbf{a} = (\mathbf{b}; \mathbf{c})$ 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**. Because $\mathbf{a} = (\mathbf{b}; \mathbf{c})$ is a proper split, we know that either $\mathbf{b} < \mathbf{c}$ or $\mathbf{c} < \mathbf{b}$. Now if the algorithm makes a combination of $(\mathbf{b}; \mathbf{c}_1)$ for some proper prefix $\mathbf{c}_1$ where $\mathbf{c} = (\mathbf{c}_1; \mathbf{c}_2)$, we have either $\mathbf{c}_1 < \mathbf{c}_2$ or $\mathbf{c}_1 > \mathbf{c}_2$. By Lemma 1, $(\mathbf{c}_1; \mathbf{c}_2)$ is a proper split. Because **c** is binarizable, 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 rightmost binarizable split (see Figure 9).

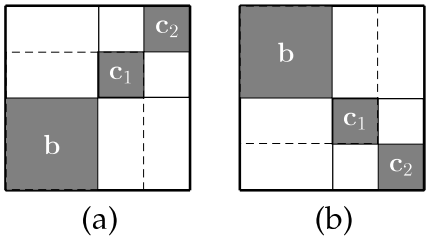


Figure 9
Illustration of the proof of Theorem 2. The combination of $(\mathbf{b}; \mathbf{c}_1)$ (in dashed squares) contradicts the assumption that $(\mathbf{b}; \mathbf{c})$ is the rightmost binarizable split of $\mathbf{a}$.

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 $bi(\mathbf{c})$. Because $\mathbf{b}$ 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 Dealing with Terminals and Adapting to Tree Transducers

Thus far we have discussed how to binarize synchronous productions involving only nonterminals through binarizing the corresponding skeleton permutations. We now turn to technical details for the implementation of a synchronous binarizer in real MT systems. We will first show how to deal with the terminal symbols, and then describe how to adapt it to tree transducers.

Consider the following SCFG rule:

(7) ADJP → RB[1] 负责 / *fùzé* PP[2] 的 / *de* NN[3],
ADJP → RB[1] responsible for the NN[3] PP[2]

whose permutation is $(1, 3, 2)$. We run the skeleton binarization algorithm and get the (canonical) binarization tree $[1, \langle 3, 2\rangle]$, which corresponds to $[\text{RB}, \langle \text{NN}, \text{PP}\rangle]$ (see Figure 10(a)). The alignment matrix is shown in Figure 11.

We will then do a post-order traversal of the skeleton tree, and attach the terminals from both languages when appropriate. It turns out we can do this quite freely as long as we can uniquely reconstruct the original rule from its binary parse tree. We use the following rules for this step:

1. Attach source-language terminals to the leaf nodes of the skeleton tree. Consecutive terminals are attached to the nonterminal on their left

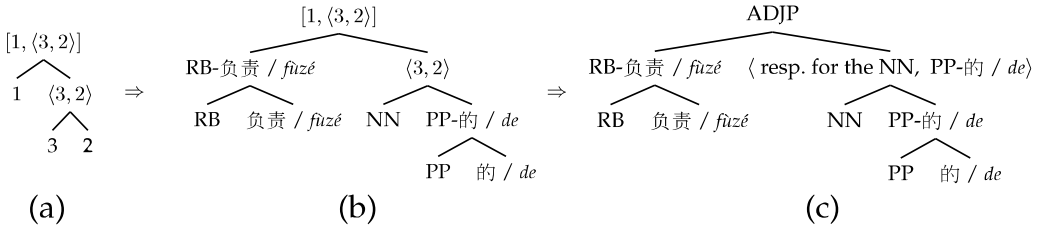


Figure 10
Attaching terminals in SCFG binarization. (a) The skeleton binarization tree, (b) attaching Chinese words at leaf nodes, (c) attaching English words at internal nodes.

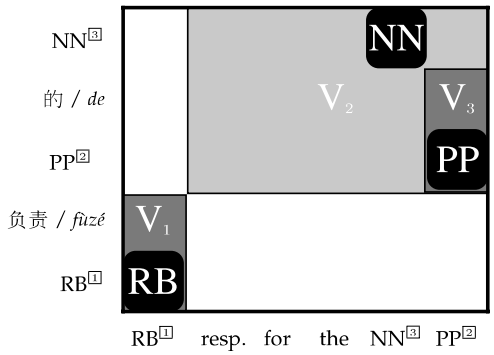


Figure 11
Alignment matrix of the SCFG rule (7). Areas shaded in gray and light gray denote virtual nonterminals (see rules in Example (8)).

(except for the initial ones which are attached to the nonterminal on their right).

2. Attach target-language terminals to the internal nodes (virtual nonterminals) of the skeleton tree. These terminals are attached greedily: When combining two nonterminals, all target-side terminal strings neighboring either nonterminal will be included. This greedy merging is motivated by the idea that the language model score helps to guide the decoder and should be computed as early as possible.

For example, at the leaf nodes, the Chinese word 负责 / *fùzé* is attached to $RB^{[1]}$, and 的 / *de* to $PP^{[1]}$ (Figure 10(b)). Next, when combining $NN^{[3]}$ and the virtual nonterminal PP-的 / *de*, we also include the English-side string *responsible for the* (Figure 10(c)). In order to do this rigorously we need to keep track of sub-alignments including both aligned nonterminals and incorporated terminals. A pre-order traversal of the fully decorated binarization tree gives us the following binary SCFG rules:

$$
(8)\quad
\begin{array}{lcl@{\qquad}lcl}
\text{ADJP} & \rightarrow & \text{V}_1^{[1]}\ \text{V}_2^{[2]}, & \text{ADJP} & \rightarrow & \text{V}_1^{[1]}\ \text{V}_2^{[2]} \\
\text{V}_1 & \rightarrow & \text{RB}^{[1]}\ 负责\ /\ fùzé, & \text{V}_1 & \rightarrow & \text{RB}^{[1]} \\
\text{V}_2 & \rightarrow & \text{V}_3^{[1]}\ \text{NN}^{[2]}, & \text{V}_2 & \rightarrow & \text{responsible for the NN}^{[2]}\ \text{V}_3^{[1]} \\
\text{V}_3 & \rightarrow & \text{PP}^{[1]}\ 的\ /\ de, & \text{V}_3 & \rightarrow & \text{PP}^{[1]}
\end{array}
$$

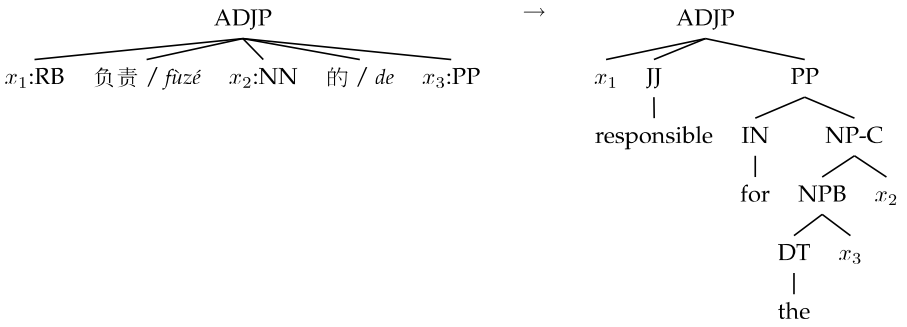
where the virtual nonterminals (illustrated in Figure 11) are:

V_1: RB-负责 / *fùzé*
V_2: ⟨ resp. for the NN, PP-的 / *de*⟩
V_3: PP-的 / *de*

Analogous to the "dotted rules" in Earley parsing for monolingual CFGs, the names we create for the virtual nonterminals reflect the underlying sub-alignments, 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).

We now turn to 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 uniquely identifies the left-hand side subtree. This gives the following two SCFG rules:

ADJP → T_{859} [1], T_{859} → RB[1] 负责 / *fùzé* PP[2] 的 / *de* NN[3],
ADJP → T_{859} [1], T_{859} → RB[1] resp. for the NN[3] PP[2]

The newly created nonterminals ensure that the newly created rules can only combine with one another to reconstruct the original rule, leaving the output of the transducer, as well as the probabilities it assigns to transductions, unchanged. The problem of binarizing tree transducers is now reduced to the binarization of SCFG rules, which we solved previously.

5. Experiments

In this section, we investigate two empirical questions with regard to synchronous binarization.

5.1 How Many Rules are (Synchronously) Binarizable?

Shapiro and Stephens (1991) and Wu (1997, Section 4) show that the percentage of binarizable cases over all permutations of length n quickly approaches 0 as n grows

(see Figure 12). However, for machine translation, the percentage of synchronous rules that are binarizable is what we care about. 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 the 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 12 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% of rules that are not binarizable, human evaluations show that the majority are due to alignment errors. Because the rule extraction process looks for rules that are consistent with both the automatic parses of the English sentences, and automatic word level alignments from GIZA++, errors in either parsing or word-level alignment can lead to noisy rules being input to the binarizer. It is also interesting to know that 86.8% of the rules have monotonic permutations, i.e., either taking identical or totally inverted order.

5.2 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 non-binarizable cases. On the other hand, simply by preferring monotonic alignments, GIZA++ might tend to miss complex non-binarizable patterns. To test this, we carried out experiments on hand-aligned sentence pairs with three language pairs: Chinese–English, French–English, and German–English.

Chinese–English Data. For Chinese–English, we used the data of Liu, Liu, and Lin (2005) which contains 935 pairs of parallel sentences. Of the 13,713 rules extracted using the same method described herein, 0.3% (44) are non-binarizable, which is exactly the

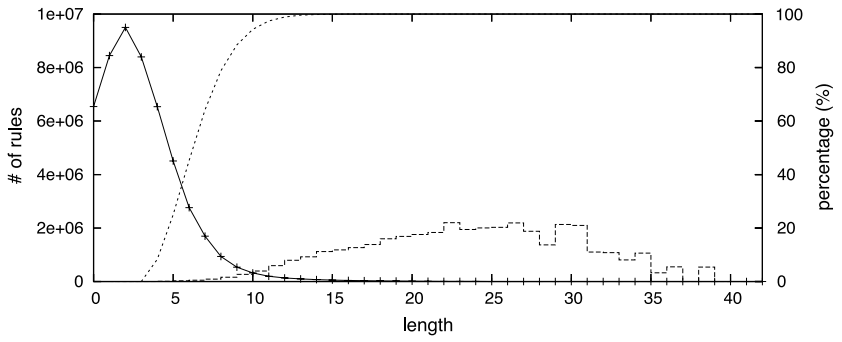
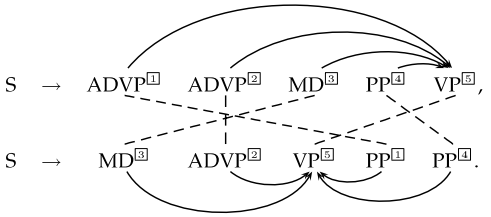


Figure 12
The solid-line curve represents the distribution of all rules against permutation lengths. The dashed-line stairs indicate the percentage of non-binarizable rules in our initial rule set, and the dotted line denotes that percentage among all permutations.

same ratio as the GIZA-aligned data. The following is an interesting example of non-binarizable rules:

(9) ... 当天$_1$ 还$_2$ 将$_3$ [与 ... 米士拉]$_4$ 会晤$_5$
... *dāngtiān* *hái* *jiāng* *yǔ* ... *Mishila* *huìwù*
... that-day also will with ... Mishira meet
... will$_3$ also$_2$ meet$_5$ [on the same day]$_1$ [with Mishira, chief secretary to the Indian prime minister and national security advisor]$_4$

where ... in *with ... Mishira* is the long phrase in shadow modifying *Mishira*. Here the non-binarizable permutation is $(3, 2, 5, 1, 4)$, which is reducible to $(2, 4, 1, 3)$. The SCFG version of the tree-transducer rule is as follows:

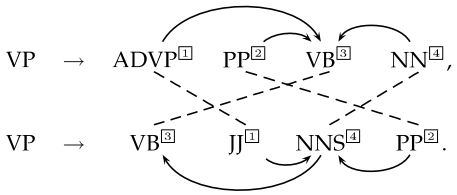


where we indicate dependency links in solid arcs and permutation in dashed lines. It is interesting to examine dependency structures, as some authors have argued that they are more likely to generalize across languages than phrase structures. The Chinese ADVP[1] 当天 / *dāngtiān* (lit., *that day*) is translated into an English PP[1] (*on the same day*), but the dependency structures on both sides are isomorphic (i.e., this is an extremely literal translation).

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

(10) ... 进一步$_1$ [就 中东 危机]$_2$ 举行$_3$ 会谈$_4$
... *jìnyībù* *jiù zhōngdōng wēijī* *jǔxíng* *huìtán*
... further on Mideast crisis hold talk
... hold$_3$ further$_1$ talks$_4$ [on the Mideast crisis]$_2$

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



Note that the Chinese ADVP[1] 进一步 / *jìnyībù* modifying the verb VB[3] becomes a JJ[1] (*further*) in the English translation modifying the object of the verb, NNS[4], and this change also happens to PP[2]. This is an example of syntactic divergence, where the dependency structures are not isomorphic between the two languages (Eisner 2003).

Wu (1997, page 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 correct (99.7% in our data sets), but also provide, for the first time, "real examples" between English and Chinese, verified by native speakers. It is interesting to note that our non-binarizable examples include both cases of isomorphic and non-isomorphic dependency structures, indicating that it is difficult to find any general linguistic explanation that covers all such examples. 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 as in our case, and reported 5% non-binarizable cases for a different hand-aligned English–Chinese data set, but they did not provide real examples.

French–English Data. We analyzed 447 hand-aligned French–English sentences from the NAACL 2003 alignment workshop (Mihalcea and Pederson 2003). We found only 2 out of 2,659 rules to be non-binarizable, or 0.1%. One of these two is an instance of topicalization:

(11) [*Je pense que*]$_1$ [*indépendent de notre parti*]$_2$ [*nous trouvons tous*]$_3$ *cela*$_4$ *inacceptable*$_5$

[That,]$_4$ [I believe,]$_1$ [we all find]$_3$ unacceptable$_5$ [regardless of our party]$_2$

The second instance is due to movement of an adverbial:

(12) [*L' entreprise,*]$_1$ [*dans le secteur privé*]$_2$ [*fut rebaptiseée*]$_3$ [*en 1934*]$_4$ [...NP...]$_5$

[In 1934,]$_4$ [the company,]$_1$ [was renamed]$_3$ [...NP...]$_5$ [still under private ownership]$_2$

German–English Data. We analyzed 220 sentences from the Europarl corpus, hand-aligned by a native German speaker (Callison-Burch, personal communication). Of 2,328 rules extracted, 13 were non-binarizable, or 0.6%. Some cases are due to separable German verb prefixes:

(13) [*Ich*]$_1$ *fordere*$_2$ [*die Kommission*]$_3$ *dringend*$_4$ *auf*$_5$ [...VP...]$_6$

I$_1$ urgently$_4$ call$_2$ on$_5$ [the commission]$_3$ [to ...VP...]$_6$

Here the German prefix *auf* is separated from the verb *auffordern* (request). Another cause of non-binarizability is verb-final word order in German in embedded clauses:

(14) ... begrüße auch ich [*den Beschluss ...*]$_1$, [...NP...]$_2$ *nicht*$_3$ [...NP...]$_4$ *zu*$_5$ *machen*$_6$

... I too welcome [the ... agreement]$_1$

not$_3$ to$_5$ make$_6$ [a solution ...]$_2$ [a pre-condition ...]$_4$

Although fewer than 1% of the rules were non-binarizable in each language pair we studied, German–English had the highest percentage with 0.6%. The fact that the German–English examples are due to syntactic phenomena such as separable prefixes and verb-final word order may indicate that an MT system would have less freedom to choose an equivalent binarizable reordering than in the case of the examples due to adverbial placement, heavy NP shift, and topicalization that we see in the Chinese–

English and French–English data. The results on binarizability of hand-aligned data for the three language pairs are summarized in Table 1.

It is worth noting that for most of these non-binarizable examples, there do exist alternative translations that only involve binarizable permutations. For example, in Chinese–English Example (9), we can move the English PP *on the same day* to the first position (before *will*), which results in a binarizable permutation $(1, 3, 2, 5, 4)$. Similarly, we can avoid non-binarizability in French–English Example (12) by moving the English adverbial *still under private ownership* to the third position. German–English Example (13) would also become binarizable by replacing *call on* with a single word *request* on the English side. However, the point of this experiment is to test the ITG hypothesis by attempting to *explain* existing real data (the hand-aligned parallel text), rather than to *generate* fresh translations for a given source sentence, which is the topic of the subsequent decoding experiment. This subsection not only provides the first solid confirmation of the existence of linguistically-motivated non-binarizable reorderings, but also motivates further theoretical studies on parsing and decoding with these non-binarizable synchronous grammars, which is the topic of Sections 6 and 7.

5.3 Does Synchronous Binarization 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 because 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 this 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, taken from the NIST 2002 test set. Both systems use trigram as the integrated language model. Figure 13 demonstrates that decoding accuracy is significantly improved after synchronous binarization. The number of edges (or *items*, in the deductive parsing terminology) proposed during

Table 1
Summary of non-binarizable ratios from hand-aligned data.

Language Pair	Sentence Pairs	Rules	Non-Binarizable Rules	Major Causes for Non-Binarization
Chinese–English	935	13,713	44 (0.3%)	Adverbials, Heavy NP Shift
French–English	447	2,659	2 (0.1%)	Adverbials, Topicalization
German–English	220	2,328	13 (0.6%)	V-final, separable prefix, etc.

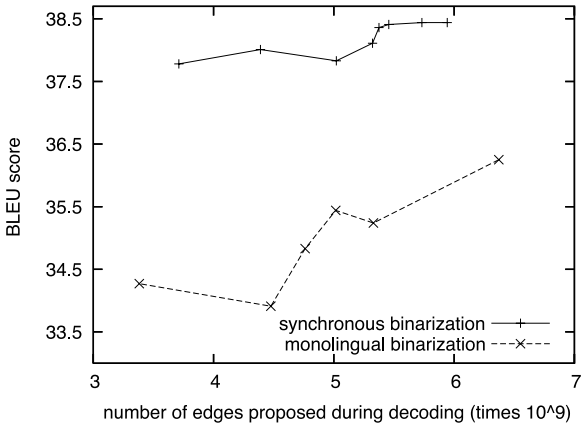


Figure 13
Comparing the two binarization methods in terms of translation quality against search effort.

Table 2
Machine translation results for syntax-based systems vs. the phrase-based Alignment Template System.

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

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.

We also compare the top result of our synchronous binarization system with the state-of-the-art alignment-template system (ATS) (Och and Ney 2004). The results are shown in Table 2. 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.[4] The readers are referred to Galley et al. (2004) for details of the decoder and the overall system.

6. One-Sided Binarization

In this section and the following section, we discuss techniques for handling rules that are not binarizable. This is primarily of theoretical interest, as we found that they constitute a small fraction of all rules, and removing these did not affect our Chinese-to-English translation results. However, non-binarizable rules are shown to be important in explaining existing hand-aligned data, especially for other language pairs such as German–English (see Section 5.2, as well as Wellington, Waxmonsky, and Melamed [2006]). Non-binarizable rules may also become more important as machine translation

4 NIST 2006 Machine Translation Evaluation Official Results: see `http://www.nist.gov/speech/tests/mt/2006/doc/mt06eval_official_results.html`.

Table 3
A summary of the four factorization algorithms, and the "incremental relaxation" theme of the whole paper. Algorithms 2–4 are for non-binarizable SCFGs, and are mainly of theoretical interest. Algorithms 1–3 make fewer and fewer assumptions on the strategy space, and produce parsing strategies closer and closer to the optimal. Algorithm 4 further improves Algorithm 3.

Section	Algorithm	Assumptions of Strategy Space	Complexity
3–4	Alg. 1 (synchronous)	Contiguous on both sides	$O(n)$
6	Alg. 2 (one-sided, CKY)	Contiguous on one side	$O(n^3)$
7.2	Alg. 3 (optimal)	No assumptions	$O(3^n)$
	$\Rightarrow$ Alg. 4 (best-first)		$O(9^k n^{2k})$

systems improve. Synchronous grammars that go beyond the power of SCFG (and therefore binary SCFG) have been defined by Shieber and Schabes (1990) and Rambow and Satta (1999), and motivated for machine translation by Melamed (2003), although previous work has not given algorithms for finding efficient and optimal parsing strategies for general SCFGs, which we believe is an important problem.

In the remainder of this section and the next section, we will present a series of algorithms that produce increasingly faster parsing strategies, by gradually relaxing the strong "continuity" constraint made by the synchronous binarization technique. As that technique requires continuity on *both* languages, we will first study a relaxation where binarized rules are always continuous in *one* of the two languages, but may be discontinuous in the other. We will present a CKY-style algorithm (Section 6.2) for finding the best parsing strategy under this new constraint, which we call **one-sided binarization**. In practice, this factorization has the advantage that we need to maintain only one set of language model boundary words for each partial hypothesis.

We will see, however, that it is not always possible to achieve the best asymptotic complexity within this constraint. But most importantly, as the synchronous binarization algorithm covers most of the SCFG rules in real data, the one-sided binarization we discuss in this section is able to achieve optimal parsing complexity for *most* of the non-binarizable rules in real data. So this section can be viewed as a middle step between the synchronous binarization we focus on in the previous sections and the optimal factorization coming in Section 7, and also a trade-off point between simplicity and asymptotic complexity for parsing strategies of SCFGs. Table 3 summarizes this incremental structure of the whole paper.

6.1 Formalizing the Problem

The complexity for decoding given a grammar factorization can be expressed in terms of the number of spans of the items being combined at each step. As an example, Figure 14 shows the three combination steps for one factorization of the non-binarizable rule:

$$X \rightarrow A^{\boxed{1}}\ B^{\boxed{2}}\ C^{\boxed{3}}\ D^{\boxed{4}},\quad X \rightarrow B^{\boxed{2}}\ D^{\boxed{4}}\ A^{\boxed{1}}\ C^{\boxed{3}} \tag{15}$$

At each step, we consider all positions in the Chinese string as possible end-points for the rule's child nonterminals. Each step combines two dynamic programming items covering disjoint spans of the Chinese input, and creates a new item covering the union of the spans. For example, in the first combination step shown in Figure 14, where nonterminals A and B are combined, A has one span in Chinese, from position y_1 to y_2 in the string, and B has one span from y_3 to y_4. The chart entry for the nonterminal

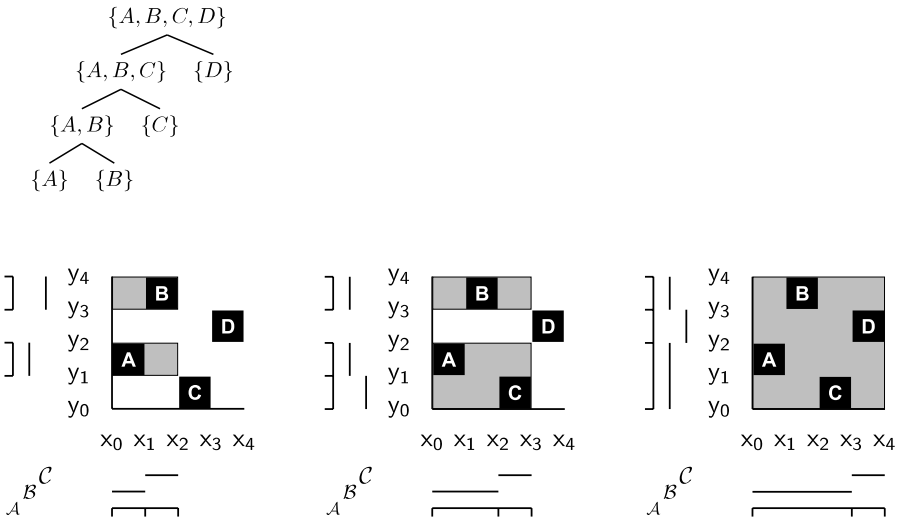


Figure 14
The tree at the top of the figure defines a three-step decoding strategy for rule (15), building an English output string on the horizontal axis as we process the Chinese input on the vertical axis. In each step, the two subsets of nonterminals in the inner marked spans are combined into a new chart item with the outer spans. The intersection of the outer spans, shaded, has now been processed.

pair $\{A, B\}$ must record a total of four string indices: positions y_1, y_2, y_3, and y_4 in the Chinese string.

Any combination of two subsets of the rule's nonterminals involves the indices for the spans of each subset. However, some of the indices are tied together: If we are joining two spans into one span in the new item, one of the original spans' end-points must be equal to another span's beginning point. For example, the index y_2 is the end-point of A in Chinese, as well as the beginning position of D. In general, if we are combining a subset $\mathcal{B}$ of nonterminals having b spans with a subset $\mathcal{C}$ having c spans, to produce a spans for a combined subset $\mathcal{A} = \mathcal{B} \cup \mathcal{C}$, the number of linked indices is $b + c - a$. In the example of the first step of Figure 14, subset $\{A\}$ has two spans (one in each language) so $b = 1$, and $\{B\}$ also has two spans, so $c = 1$. The combined subset $\{A, B\}$ has two spans, so $a = 2$. The total number of indices involved in a combination of two subsets is

$$2(b + c) - (b + c - a) = a + b + c \tag{16}$$

where $2(b + c)$ represents the original beginning and end points, and $b + c - a$ the number of shared indices. In the first step of Figure 14, $a + b + c = 1 + 1 + 2 = 4$ total indices, and therefore the complexity of this step is $O(|w|^4)$ where $|w|$ is the length of the input Chinese strings, and we ignore the language model for the moment. Applying this formula to the second and third step, we see that the second is $O(|w|^5)$, and the third is again $O(|w|^4)$.

In order to find a good decoding strategy for a given grammar rule, we need to search over possible orders in which partial translation hypotheses can be built by successively combining nonterminals. Any strategy we find can be used for synchronous parsing as well as decoding. For example, the strategy shown in Figure 14 can be used to parse an input Chinese/English string pair. The complexity of each step is determined by the total number of indices into *both* the Chinese and English strings. Each step in

Algorithm 2 An $O(n^3)$ CKY-style algorithm for parsing strategies, keeping continuous spans in one language. Takes an SCFG rule's permutation as input and returns the complexity of the best parsing strategy found.

```
1: function BESTCONTINUOUSPARSER(π)
2:     n = |π|
3:     for span ← 1 to n − 1 do
4:         for i ← 1 to n − span do
5:             A = {i...i + span}
6:             best[A] = ∞
7:             for j ← i + 1 to i + span do
8:                 B = {i...j − 1}
9:                 C = {j...i + span}
10:                Let a_π, b_π, and c_π denote the number of π(A), π(B), and π(C)'s spans
11:                compl[A → BC] = max {a_π + b_π + c_π, best[B], best[C]}
12:                if compl[A → BC] < best[A] then
13:                    best[A] = compl[A → BC]
14:                    rule[A] = A → BC
15:    return best[{1...n}]
```

the diagram has three indices into the English string, so the complexity of the first step is $O(|w|^{4+3}) = O(|w|^7)$, the second step is $O(|w|^8)$, and the third is again $O(|w|^7)$.

6.2 A CKY-Style Algorithm for Parsing Strategies

The $O(n^3)$ algorithm we present in this section can find good factorizations for most non-binarizable rules; we discuss optimal factorization in the next section. This algorithm, shown in Algorithm 2, considers only factorizations that have only one span in one of the two languages, and efficiently searches over all such factorizations by combining adjacent spans with CKY-style parsing.[5] The input is an SCFG grammar rule in its abstract form, which is a permutation, and *best* is a dynamic programming table used to store the lowest complexity with which we can parse a given subset of the input rule's child nonterminals.

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

$$\begin{array}{ll} 1 \leftrightarrow 1 & n/2+1 \leftrightarrow 2 \\ 2 \leftrightarrow n/2+1 & n/2+2 \leftrightarrow n/2+2 \\ 3 \leftrightarrow 3 & n/2+3 \leftrightarrow 4 \\ 4 \leftrightarrow n/2+3 & n/2+4 \leftrightarrow n/2+4 \\ \ldots & \ldots \end{array}$$

This pattern, shown graphically in Figure 15 for $n = 16$, can be parsed in time $O(|w|^{10})$ by maintaining a partially completed item with two spans in each dimension, one

5 A special case of this algorithm, **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.

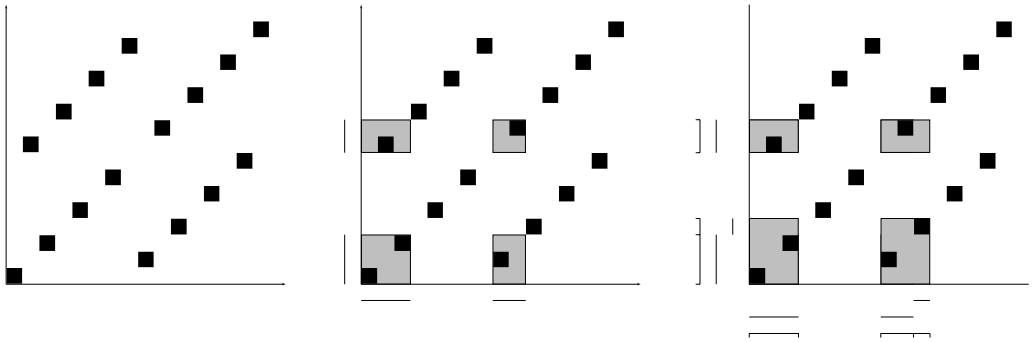


Figure 15
Left, a general pattern of non-binarizable permutations. Center, a partially completed chart item with two spans in each dimension; the intersection of the completed spans is shaded. Right, the combination of the item from the center panel with a singleton item. The two subsets of nonterminals in the inner marked spans are combined into a new chart item with the outer spans.

beginning at 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 15 (right). However, our CKY factorization algorithm will give a factorization 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.

7. Optimal Factorization

The method presented in the previous section is not optimal for all permutations, because in some cases it is better to maintain multiple spans in the output language (despite the extra language model state that is needed) in order to maintain continuous spans in the input language. In this section we give a method for finding decoding strategies that are guaranteed to be optimal in their asymptotic complexity.

This method can also be used to find the optimal strategy for synchronous parsing (alignment) using complex rules. This answers a question left open by earlier work in synchronous grammars: Although Satta and Peserico (2005) show that tabular parsing of a worst-case SCFG can be NP-hard, they do not give a procedure for finding the complexity of an arbitrary input grammar. Similarly, Melamed (2003) defines the cardinality of a grammar, and discusses the interaction of this property with parsing complexity, but does not show how to find a normal form for a grammar with the lowest possible cardinality.

We show below how to analyze parsing and decoding strategies for a given SCFG rule in Section 7.1, and then present an exponential-time dynamic programming algorithm for finding the best strategy in Section 7.2. We prove that factorizing an SCFG rule into smaller SCFG rules is a safe preprocessing step for finding the best strategy in Section 7.4, which leads to much faster computation in many cases. First, however, we take a brief detour to discuss modifying our $a + b + c$ formula from the previous section in order to take the state from an m-gram language model into account during MT decoding.

7.1 Taking the Language Model into Account

First, how do we analyze algorithms that create discontinuous spans in both the source and target language? It turns out that the analysis in Section 6.1 for counting string indices in terms of spans in fact applies in the same way to both of our languages. For synchronous parsing, if we are combining item $\mathcal{B}$ with b_e target spans and b_f source spans with item $\mathcal{C}$ having c_e target spans and c_f source spans to form new item $\mathcal{A}$ having a_e target spans and a_f source spans, the complexity of the operation is $O(|w|^{a_e+b_e+c_e+a_f+b_f+c_f})$.

The $a + b + c$ formula can also be generalized for decoding with an integrated m-gram language model. At first glance, because 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 by factorizing the dynamic programming combination rule into two steps. One step incorporates the language model probability, and the other step combines adjacent spans in the input language and incorporates the SCFG rule probability. The hook trick for a bigram language model and binary SCFG is shown in Figure 16. In the equations of Figure 16, i,j,k range over 1 to $|w|$, the length of the input foreign sentence, and u,v,v_1,u_2 (or u,v,v_2,u_1) range over possible English words, which we assume to take $O(|w|)$ possible values. There are seven free variables related to input size for doing the maximization computation, hence the algorithmic complexity is $O(|w|^7)$.

The two terms in Figure 16 (top) within the first level of the max operator correspond to straight and inverted ITG rules. Figure 16 (bottom) shows how to decompose the first term; the same method applies to the second term. Counting the free variables enclosed in the innermost max operator, we get five: i, k, u, v_1, and u_2. The

$$\beta(X[i,j,u,v]) = \max \left\{ \begin{array}{l} \max\limits_{k,v_1,u_2,Y,Z} \left[\begin{array}{l} \beta(Y[i,k,u,v_1]) \cdot \beta(Z[k,j,u_2,v]) \\ \cdot\, P(X \rightarrow [YZ]) \cdot bigram(v_1,u_2) \end{array} \right], \\ \max\limits_{k,v_2,u_1,Y,Z} \left[\begin{array}{l} \beta(Y[i,k,u_1,v]) \cdot \beta(Z[k,j,u,v_2]) \\ \cdot\, P(X \rightarrow \langle YZ \rangle) \cdot bigram(v_2,u_1) \end{array} \right] \end{array} \right\}$$

where

$$\max_{k,v_1,u_2,Y,Z} \Big[\beta(Y[i,k,u,v_1]) \cdot \beta(Z[k,j,u_2,v]) \cdot P(X \rightarrow [YZ]) \cdot bigram(v_1,u_2) \Big]$$

$$= \max_{k,u_2,Z} \left[\max_{v_1,Y} \Big[\beta(Y[i,k,u,v_1]) \cdot P(X \rightarrow [YZ]) \cdot bigram(v_1,u_2) \Big] \cdot \beta(Z[k,j,u_2,v]) \right]$$

Figure 16
The hook trick for machine translation decoding with a binary SCFG (equivalent to Inversion Transduction Grammar). The fundamental dynamic programming equation is shown at the top, with an efficient factorization shown below.

decomposition eliminates one free variable, v_1. In the outermost level, there are six free variables left. The maximum number of interacting variables is six overall. So, we have reduced the complexity of ITG decoding using the bigram language model from $O(|w|^7)$ to $O(|w|^6)$.

When we generalize the hook trick for any m-gram language model and more complex SCFGs, 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 $m-1$:

$$(m-1)(a_e + b_e + c_e) + a_f + b_f + c_f \tag{17}$$

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 $(m-1)$:

$$a_m = a_f + (m-1)a_e \tag{18}$$

Using this notation, we can rewrite the expression for the complexity of decoding in Equation 17 as a simple sum of the numbers of weighted spans of constituent subsets $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$:

$$a_m + b_m + c_m \tag{19}$$

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

$$(a_f + (m-1)a_e) + \sum_{i=1}^{k} (b_{fi} + (m-1)b_{ei}) = a_m + \sum_{i=1}^{k} b_{mi} \tag{20}$$

It can be seen that, as m grows, the parsing/decoding strategies that favor contiguity on the output side will prevail. This effect is demonstrated by the experimental results in Section 7.5.

This analysis 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 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 super-

exponentially, as $0.175n!n^{-3/2}2.59^n = \Theta(\Gamma(n+1)2.59^n)$ (Schröder 1870, Problem IV).[6] More formally, the optimization over the space of all recursive partitions is expressed as:

$$best(\mathcal{A}) = \min_{\mathcal{B}:\mathcal{A}=\bigcup_{i=1}^{k}\mathcal{B}_i} \max\{compl(\mathcal{A} \rightarrow \mathcal{B}_1 ... \mathcal{B}_k),\ \max_{i=1}^{k} best(\mathcal{B}_i)\} \tag{21}$$

where $compl(\mathcal{A} \rightarrow \mathcal{B}_1 ... \mathcal{B}_k)$ is given by Equation (20). This recursive equation implies that we can solve the optimization problem using dynamic programming techniques.

7.2 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 (20)) 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 (18), and we drop both the adjective "weighted" and the m subscript from this point forward.

If we are combining k subsets $\mathcal{B}_i$ ($i = 1, ..., k, k \geq 2$) together to produce a new subset $\mathcal{A} = \bigcup_i \mathcal{B}_i$, the generalized formula for counting the total number of indices is

$$a + \sum_{i=1}^{k} b_i \tag{22}$$

where b_i is the number of spans for $\mathcal{B}_i$ and a is the number of spans for the resulting item $\mathcal{A}$.

Consider a ternary rule $\mathcal{X} \rightarrow \mathcal{A}\,\mathcal{B}\,\mathcal{C}$ where $\mathcal{X}$ has x spans, $\mathcal{A}$ has a spans, $\mathcal{B}$ has b spans, and $\mathcal{C}$ has c spans. In the example shown in Figure 17, $x = 2$, $a = 2$, $b = 1$, and $c = 2$. The complexity for parsing this is $O(|w|^{a+b+c+x})$. Now factor this rule into two rules:

$$\mathcal{X} \rightarrow \mathcal{Y}\,\mathcal{C}$$

$$\mathcal{Y} \rightarrow \mathcal{A}\,\mathcal{B}$$

6 We use Θ to indicate an asymptotic bound that is tight in both directions.

and refer to the number of spans in partial constituent $\mathcal{Y}$ as y. Parsing $\mathcal{Y} \rightarrow \mathcal{A}\mathcal{B}$ 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 $\mathcal{Y}$ corresponds to a left edge. (In the case of decoding, each edge has a multiplicity of $(m - 1)$ on the output language side.) The left edge in each span of $\mathcal{Y}$ corresponds to the left edge of a span in $\mathcal{X}$ or to the right edge of a span in $\mathcal{C}$. Therefore, $\mathcal{Y}$ has at most one span for each span in $\mathcal{C} \cup \mathcal{X}$, so $y \leq c + x$.

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

$$y \leq a + b$$

since $\mathcal{Y}$ was formed from $\mathcal{A}$ and $\mathcal{B}$. Therefore

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

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

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).

Although the total number of binary recursive partitions is still superexponential, the binary branching property also enables a straightforward dynamic programming algorithm, shown in Algorithm 3. 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 9 refer to the total number of spans of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ (Equation (16)),

Figure 17
Left, example spans for a ternary rule decomposition $\mathcal{X} \rightarrow \mathcal{A}\mathcal{B}\mathcal{C}$. Each symbol represents a subset of nonterminals from the original SCFG rule, and the subsets may cover discontinuous spans in either language. Line segments represent the projection of each set of child nonterminals into a single language, as in Figure 15. Right, factorization into $\mathcal{X} \rightarrow \mathcal{Y}\mathcal{C}$ and $\mathcal{Y} \rightarrow \mathcal{A}\mathcal{B}$.

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

```
1:  function BestDiscontinuousParser(π)
2:      n = |π|
3:      for i ← 2 to n do
4:          for A ⊂ {1...n} s.t. |A| = i do
5:              best[A] ← ∞
6:              for B, C s.t. A = B ∪ C ∧ B ∩ C = ∅ do
7:                  Let a, b, and c denote the number of
8:                  (A, π(A)), (B, π(B)), and (C, π(C))'s spans
9:                  compl[A → B C] = max {a + b + c, best[B], best[C]}
10:                 if compl[A → B C] < best[A] then
11:                     best[A] ← compl[A → B C]
12:                     rule[A] ← A → B C
13:     return best[{1...n}]
```

while for decoding, a, b, and c refer to weighted spans (Equation (19)). 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 $\mathcal{B}$, $\mathcal{C}$, or neither $\mathcal{B}$ nor $\mathcal{C}$, making the total running time of the algorithm $O(3^n)$. Although 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 4, guarantees polynomial-time processing on input permutations of bounded complexity. To see why this is, observe that each rule of the form $\mathcal{A} \rightarrow \mathcal{B}\,\mathcal{C}$ 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 $\mathcal{B}$, starts a span of subset $\mathcal{C}$, or ends a span of $\mathcal{B} \cup \mathcal{C}$. 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 $O(9^k n^{2k})$. Although 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.

7.3 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 18, can be parsed in time $O(|w|^8)$ using unconstrained subsets, but only in

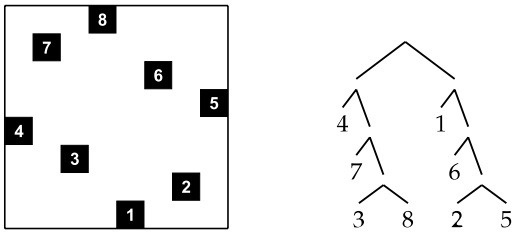


Figure 18
The permutation $(4, 7, 3, 8, 1, 6, 2, 5)$ cannot be efficiently parsed by adding one nonterminal at a time. The optimal grouping of nonterminals is shown on the right.

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.

7.4 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 4, where k_c is the complexity of the longest SCFG rule after factorizations, implying that $k_c \leq (n + 4)$. 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 19 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

Algorithm 4 Best-first search for the optimal parsing strategy.

```
1:  function BestDiscontinuousParser(π)
2:      n = |π|
3:      for A ⊂ {1...n} do
4:          chart[A] = ∞
5:      for i ← 1...n do
6:          push[heap, 0, {i}]                          ▷ Priority queue of good subsets
7:      while chart[{1...n}] = ∞ do
8:          B ← pop(heap)
9:          chart[B] ← best[B]          ▷ guaranteed to have found optimal analysis for subset B
10:         for C s.t. B ∩ C = ∅ ∧ chart[C] < ∞ do
11:             A ← B ∪ C
12:             Let a, b, and c denote the number of
13:             (A, π(A)), (B, π(B)), and (C, π(C))'s spans
14:             compl[A → B C] = max{a + b + c, best[B], best[C]}
15:             if compl[A → B C] < best[A] then
16:                 best[A] ← compl[A → B C]
17:                 rule[A] ← A → B C
18:                 push(heap, best[A], A)
19:     return best[{1...n}]
```

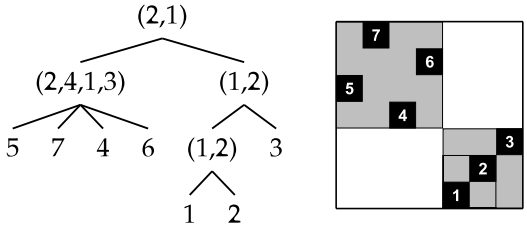


Figure 19
A permutation that can be decomposed into smaller permutations hierarchically. We prove that this decomposition corresponds to the optimal parsing strategy for an SCFG rule with this permutation.

optimal parsing strategy for the entire permutation does not have to involve interactions 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 19, 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 19 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 a 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 the Appendix for the proof. See Table 3 for a summary of the four factorization algorithms presented in this article (Algorithms 3 and 4 can be improved by first factorizing the permutation into smaller permutations [Section 7.4]).

7.5 Experiments

The combination of minimizing SCFG rule length as a preprocessing step and then applying the best-first version of Algorithm 3 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 non-binarizable rules (0.3% of the total), the complexity of the optimal parsing strategies is shown in Table 4. Although 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 non-binarizable rules extracted from the 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 ruleset was $O(|w|^{15})$, rather than the optimal $O(|w|^{12})$. If we run CKY factorization from two directions (one for the permutation π and the other for the permutation π^{-1}) and take the minimum of both, we can get an even better approximation. In Table 4, we compare the approximate strategy which takes the minimum of CKY runs for

Table 4
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—for example, 10 means $O(|w|^{10})$. *opt* denotes the optimal parsing strategy and *cky-min* denotes the approximation strategy that takes the better of the CKY results on both sides.

Complexities	Synchronous Parsing /Bigram Decoding		Trigram Decoding (Chinese to English)		Trigram Decoding (English to Chinese)	
	opt	*cky-min*	*opt*	*cky-min*	*opt*	*cky-min*
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				

two languages, which we call CKY-min, 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 non-binarizable rules. The CKY-min strategy was even better, only finding sub-optimal results for six rules out of all rules, which translates to 99.996%. In Table 4, we have 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.

7.6 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 Štefankovič (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.

8. Conclusion

This work develops a theory of binarization for synchronous context-free grammars. We present 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 non-binarizable rules did not improve our translation system, these parsing strategies are primarily of theoretical interest, though they may become more important in future systems.

Acknowledgments
Much of this work was done while the first two authors were visiting USC/ISI. This work was partially funded by NSF grants IIS-0428020 and EIA-0205456.

Appendix A. Proof of Theorem 4

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 at the top of Figure 20, in either or both of which there are some pebbles from outside S. As in Section 7, we count spans in this section using the weighted span value of Equation (18) 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 20.

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 the source corresponds to the reduction of one span. Each boundary in the target corresponds to the 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

$$r_R \leq m \tag{A.1}$$

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

$$best(T) = \max \{\, best(T_L),\ best(T_R),\ ((d_{LO} + d_{LI} - r_L) + (d_{RI} + d_{RO} - r_R) + d) \,\} \tag{A.2}$$

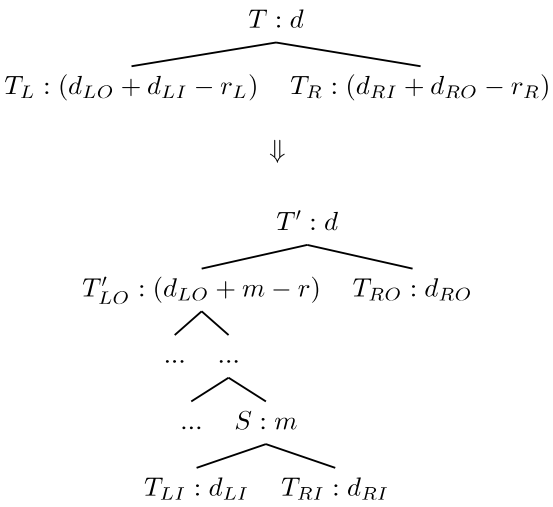


Figure 20
Reorganization of a parsing strategy to build a continuous span S first.

Figure 21 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 at 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.

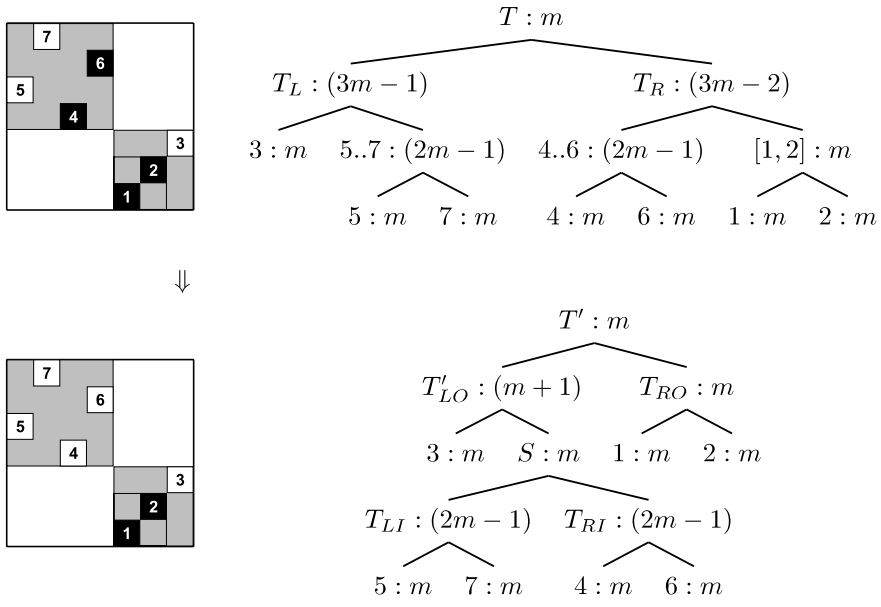


Figure 21
An actual example of reorganization of a parsing strategy to build a continuous span S first. Before, the overall strategy cost is $(7m - 3)$. After, the cost is $(5m - 2)$. Note that $(m \geq 2)$. We use black to represent pebbles in the right branch of the root node and white for the left 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.

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 at the bottom of Figure 20. 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 $a + b + c$ 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 S. Hence, we have the following set of inequalities:

$$best(T_{LI}) \leq best(T_L) \tag{A.3}$$

$$best(T_{RI}) \leq best(T_R) \tag{A.4}$$

$$best(T_{RO}) \leq best(T_R) \tag{A.5}$$

Now we consider what happens when the pebble of S joins T_{LO}. Because 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 the place of any trace of the pruned leaves and making 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:

$$best(T'_{LO}/S) \leq best(T_L) \tag{A.6}$$

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 20 (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,

$$r_L \leq r \tag{A.7}$$

The final yield of the updated strategy is

$$best(T') = \max \left\{ \begin{array}{c} best(T'_{LO}/S),\ best(T_{LI}),\ best(T_{RI}),\ best(T_{RO}), \\ (d_{LI} + d_{RI} + m), \\ ((d_{LO} + m - r) + d_{RO} + d) \end{array} \right\} \tag{A.8}$$

We have shown the first four terms inside the maximization are bounded by the yield of the old strategy (Equations [A.3–A.6]). We need to bound the remaining terms.

Both of them can be bounded by the third term inside the maximization of Equation (A.2). The first inequality is

$$d_{LI} + d_{RI} + m \leq (d_{LO} + d_{LI} - r_L) + (d_{RI} + d_{RO} - r_R) + d \tag{A.9}$$

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).

Our final bound relates the last terms of Equations (A.2) and (A.8)

$$((d_{LO} + m - r) + d_{RO} + d) \leq ((d_{LO} + d_{LI} - r_L) + (d_{RI} + d_{RO} - r_R) + d) \tag{A.10}$$

This simplifies to

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

This inequality is true because $m \leq d_{LI}$, since 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 (A.1), and finally $r \geq r_L$, as shown in Equation (A.7).

Figure 20 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 cannot be better than an optimized strategy that respects such boundaries. ■

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.

Bodlaender, H. L. and D. M. Thilikos. 1997. Treewidth for graphs with small chordality. *Discrete Applied Mathematics*, 79:45–61.

Catalan, Eugène. 1844. Note extraite d'une lettre adressée. *Journal fürdie reine und angewandte Mathematik*, 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, Ann Arbor, MI.

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*, pages 205–208, Sapporo, Japan.

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

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

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)*, pages 33–40, Rochester, NY.

Huang, Liang and David Chiang. 2005. Better k-best parsing. In *International Workshop on Parsing Technologies (IWPT05)*, pages 53–64, Vancouver.

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

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)*, pages 1–24. Mexico City.

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

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)*, pages 459–466, Ann Arbor, MI.

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)*, pages 158–165, Edmonton.

Mihalcea, Rada and Ted Pederson. 2003. An evaluation exercise for word alignment. In *HLT-NAACL 2003 Workshop on Building and Using Parallel Texts: Data Driven Machine Translation and Beyond*, pages 1–10, Edmonton.

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

Och, Franz Josef and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449.

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

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.

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 M. 2004. Synchronous grammars as tree transducers. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+ 7)*, pages 88–95, Vancouver.

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, Helsinki.

Wellington, Benjamin, Sonjia Waxmonsky, and I. Dan Melamed. 2006. Empirical lower bounds on the complexity of translational equivalence. In *Proceedings of the International Conference on Computational Linguistics/Association for Computational Linguistics (COLING/ACL-06)*, pages 977–984, Sydney.

Wu, Dekai. 1996. A polynomial–time algorithm for statistical machine translation. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 152–158, Santa Cruz, CA.

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

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

Zhang, Hao, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proceedings of the NAACL*, pages 256–263, New York.