Synchronous Binarization for Machine Translation

Hao Zhang
Computer Science Department
University of Rochester
Rochester, NY 14627
zhanghao@cs.rochester.edu

Liang Huang
Dept. of Computer & Information Science
University of Pennsylvania
Philadelphia, PA 19104
lhuang3@cis.upenn.edu

Daniel Gildea
Computer Science Department
University of Rochester
Rochester, NY 14627
gildea@cs.rochester.edu

Kevin Knight
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
knight@isi.edu

Abstract

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, and rules extracted from parallel corpora can be quite large. We devise a linear-time algorithm for factoring syntactic re-orderings by binarizing synchronous rules when possible and show that the resulting rule set significantly improves the speed and accuracy of a state-of-the-art syntax-based machine translation system.

1 Introduction

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

- We develop a technique called *synchronous binarization* and devise a fast 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 machine translation quality, compared to a more typical baseline method.
- We examine cases of non-binarizable rules in a large, empirically-derived rule set, and we investigate the effect on translation quality when excluding such rules.

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

Suppose we have the following SCFG, where superscripts indicate reorderings (formal definitions of

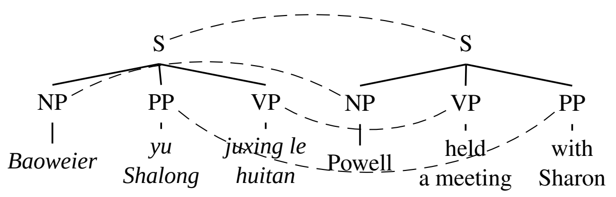


Figure 1: A pair of synchronous parse trees in the SCFG (1). The dashed curves indicate pairs of synchronous nonterminals (and sub trees).

SCFGs can be found in Section 2):

$$
(1)\quad
\begin{array}{rl}
\mathrm{S} \rightarrow & \mathrm{NP}^{(1)}\ \mathrm{VP}^{(2)}\ \mathrm{PP}^{(3)},\ \mathrm{NP}^{(1)}\ \mathrm{PP}^{(3)}\ \mathrm{VP}^{(2)} \\
\mathrm{NP} \rightarrow & \text{Powell, } \textit{Baoweier} \\
\mathrm{VP} \rightarrow & \text{held a meeting, } \textit{juxing le huitan} \\
\mathrm{PP} \rightarrow & \text{with Sharon, } \textit{yu Shalong}
\end{array}
$$

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

To efficiently decode with CKY, we need to binarize the projected CFG grammar.[1] Rules can be binarized in different ways. For example, we could binarize the first rule left to right or right to left:

$$
\begin{array}{rl}
\mathrm{S} \rightarrow & V_{\text{NP-PP}}\ \mathrm{VP} \\
V_{\text{NP-PP}} \rightarrow & \mathrm{NP}\ \mathrm{PP}
\end{array}
\quad \text{or} \quad
\begin{array}{rl}
\mathrm{S} \rightarrow & \mathrm{NP}\ V_{\text{PP-VP}} \\
V_{\text{PP-VP}} \rightarrow & \mathrm{PP}\ \mathrm{VP}
\end{array}
$$

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

These two binarizations are no different in the translation-model-only decoding described above, just as in monolingual parsing. However, in the source-channel approach to machine translation, we need to combine probabilities from the translation model (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, we need to augment each chart item (X, i, j) with two target-language *boundary words* u and v to produce a bigram-item like $\left(\begin{smallmatrix} u \cdots v \\ \mathrm{X} \\ i \qquad j \end{smallmatrix}\right)$, following the dynamic programming algorithm of Wu (1996).

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

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

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 $V_{\text{NP-PP}}$ item. Instead, we have to maintain all four boundary words (rather than two) and postpone the language model scoring till the next step where $V_{\text{NP-PP}}$ is combined with $\left(\begin{smallmatrix} \text{held} \cdots \text{meeting} \\ \mathrm{VP} \\ 2 \qquad\quad 4 \end{smallmatrix}\right)$ to form an S item. We call this binarization method *monolingual binarization* since it works only on the source-language projection of the rule without respecting the constraints from the other side.

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

In the second case, however:

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

Here since PP and VP are contiguous (but swapped) in the target-language, we can include the

[1]Other parsing strategies like the Earley algorithm use an internal binary representation (e.g. dotted-rules) of the original grammar to ensure cubic time complexity.

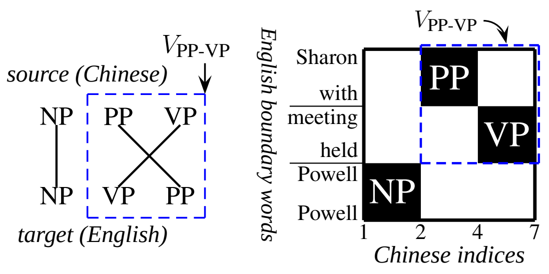


Figure 2: The alignment pattern (left) and alignment matrix (right) of the synchronous production.

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

$$(2) \quad \begin{array}{rl} \text{S} \rightarrow & \text{NP}^{(1)}\, V_{\text{PP-VP}}{}^{(2)},\ \text{NP}^{(1)}\, V_{\text{PP-VP}}{}^{(2)} \\ V_{\text{PP-VP}} \rightarrow & \text{VP}^{(1)}\, \text{PP}^{(2)},\ \text{PP}^{(2)}\, \text{VP}^{(1)} \end{array}$$

In this case m-gram integrated decoding can be done in $O(|w|^{3+4(m-1)})$ time which is much lower-order polynomial and no longer depends on rule size (Wu, 1996), allowing the search to be much faster and more accurate facing pruning, as is evidenced in the Hiero system of Chiang (2005) where he restricts the hierarchical phrases to be a binary SCFG. The benefit of binary grammars also lies in synchronous parsing (alignment). Wu (1997) shows that parsing a binary SCFG is in $O(|w|^6)$ while parsing SCFG is NP-hard in general (Satta and Peserico, 2005).

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

(3)
S(x_0:NP, VP(x_2:VP, x_1:PP)) $\rightarrow x_0\ x_1\ x_2$
NP(NNP(Powell)) $\rightarrow$ *Baoweier*
VP(VBD(held), NP(DT(a) NPS(meeting))) $\rightarrow$ *juxing le huitan*
PP(TO(with), NP(NNP(Sharon))) $\rightarrow$ *yu Shalong*

where the reorderings of nonterminals are denoted by variables x_i.

Notice that the first rule has a multi-level left-hand side subtree. This system can model non-isomorphic transformations on English parse trees to "fit" another language, for example, learning that the $(S\ (V\ O))$ structure in English should be transformed into a $(V\ S\ O)$ structure in Arabic, 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). This larger locality is linguistically motivated and leads to a better parameter estimation. By imagining the left-hand-side trees as special nonterminals, we can virtually create an SCFG with the same generative capacity. The technical details will be explained in Section 3.2.

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.

2 Synchronous Binarization

A **synchronous CFG** (SCFG) is a context-free rewriting system for generating string pairs. Each rule (*synchronous production*) rewrites a nonterminal in two dimensions subject to the constraint that the sequence of nonterminal children on one side is a permutation of the nonterminal sequence on the other side. Each co-indexed child nonterminal pair will be further rewritten as a unit.[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 symbol.

As shown in Section 3.2, terminals do not play an important role in binarization. So we now write rules in the following notation:

$$X \rightarrow X_1^{(1)} ... X_n^{(n)},\ X_{\pi(1)}^{(\pi(1))} ... X_{\pi(n)}^{(\pi(n))}$$

where each X_i is a variable which ranges over nonterminals in the grammar and π is the **permutation** of the rule. We also define an SCFG rule as n-ary if its permutation is of n and call an SCFG n-ary if its longest rule is n-ary. Our goal is to produce an equivalent *binary* SCFG for an input n-ary SCFG.

[2]In making one nonterminal play dual roles, we follow the definitions in (Aho and Ullman, 1972; Chiang, 2005), originally known as Syntax Directed Translation Schema (SDTS). An alternative definition by Satta and Peserico (2005) allows co-indexed nonterminals taking different symbols in two dimensions. Formally speaking, we can construct an equivalent SDTS by creating a cross-product of nonterminals from two sides. See (Satta and Peserico, 2005, Sec. 4) for other details.

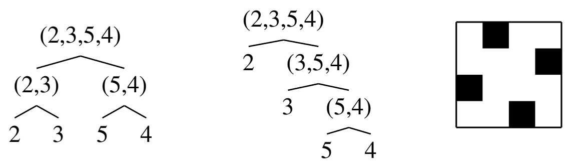


Figure 3: (a) and (b): two binarization patterns for $(2, 3, 5, 4)$. (c): alignment matrix for the non-binarizable permuted sequence $(2, 4, 1, 3)$

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). So we now start to rigorously define the binarizability of permutations.

2.1 Binarizable Permutations

A **permuted sequence** is a permutation of consecutive integers. For example, $(3, 5, 4)$ is a permuted sequence while $(2, 5)$ is not. As special cases, single numbers are permuted sequences as well.

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

1. $\mathbf{a}$ is a singleton, i.e. $\mathbf{a} = (a)$, or
2. $\mathbf{a}$ can be split into two sub sequences, i.e. $\mathbf{a} = (\mathbf{b}; \mathbf{c})$, where $\mathbf{b}$ and $\mathbf{c}$ are both binarizable permuted sequences. We call such a division $(\mathbf{b}; \mathbf{c})$ a **binarizable split** of $\mathbf{a}$.

This is a recursive definition. Each binarizable permuted sequence has at least one hierarchical binarization pattern. For instance, the permuted sequence $(2, 3, 5, 4)$ is binarizable (with two possible binarization patterns) while $(2, 4, 1, 3)$ is not (see Figure 3).

2.2 Binarizable SCFG

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 represents an important subclass of SCFG that is easy to handle (parsable in $O(|w|^6)$) and covers many interesting longer-than-two rules.[3]

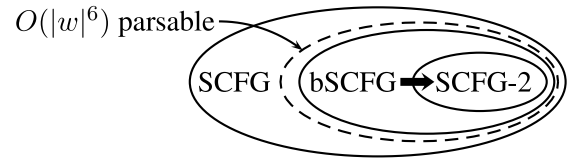


Figure 4: Subclasses of SCFG. The thick arrow denotes the direction of synchronous binarization. For clarity reasons, binary SCFG is coded as SCFG-2.

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

Proof. Once we decompose the permutation of n in the original rule into binary permutations, all that remains is to decorate the skeleton binary parse with nonterminal symbols and attach terminals to the skeleton appropriately. We explain the technical details in the next section. $\square$

3 Binarization Algorithms

We have reduced the problem of binarizing an SCFG rule into the problem of binarizing its permutation. This problem can be cast as 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. In fact, as demonstrated previously, some permutations may have more than one binarization patterns among which we only need one. Wu (1997, Sec. 7) introduces a non-ambiguous ITG that prefers left-heavy binary trees so that for each permutation there is a unique synchronous derivation (binarization pattern).

However, this problem has more efficient solutions. Shapiro and Stephens (1991, p. 277) informally present an iterative procedure where in each pass it scans the permuted sequence from left to right and combines two adjacent sub sequences whenever possible. This procedure produces a left-heavy binarization tree consistent with the unambiguous ITG and runs in $O(n^2)$ time since we need n passes in the worst case. We modify this procedure and improve

[3]Although we factor the SCFG rules individually and define bSCFG accordingly, there are some grammars (the dashed circle in Figure 4), which can be binarized only by analyzing interactions between rules. Below is a simple example:

$$\begin{array}{ll} \mathrm{S} \rightarrow & \mathrm{X}^{(1)}\,\mathrm{X}^{(2)}\,\mathrm{X}^{(3)}\,\mathrm{X}^{(4)},\ \mathrm{X}^{(2)}\,\mathrm{X}^{(4)}\,\mathrm{X}^{(1)}\,\mathrm{X}^{(3)} \\ \mathrm{X} \rightarrow & \mathrm{a}\ ,\ \mathrm{a} \end{array}$$

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

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

it into a linear-time shift-reduce algorithm that only needs one pass through the sequence.

3.1 The linear-time skeleton algorithm

The (unique) **binarization tree** $bi(\mathbf{a})$ for a binarizable permuted sequence $\mathbf{a}$ is recursively defined as follows:

- if $\mathbf{a} = (a)$, then $bi(\mathbf{a}) = a$;
- otherwise let $\mathbf{a} = (\mathbf{b}; \mathbf{c})$ to be the *rightmost* binarizable split of $\mathbf{a}$. then

$$bi(\mathbf{a}) = \begin{cases} [bi(\mathbf{b}), bi(\mathbf{c})] & b_1 < c_1 \\ \langle bi(\mathbf{b}), bi(\mathbf{c})\rangle & b_1 > c_1. \end{cases}$$

For example, the binarization tree for $(2, 3, 5, 4)$ is $[[2, 3], \langle 5, 4\rangle]$, which corresponds to the binarization pattern in Figure 3(a). We use [] and $\langle\rangle$ for straight and inverted combinations respectively, following the ITG notation (Wu, 1997). The rightmost split ensures left-heavy binary trees.

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, like 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 details and Figure 5 for an example.

Theorem 2. *Algorithm 1 succeeds if and only if the input permuted sequence* $\mathbf{a}$ *is binarizable, and in case of success, the binarization pattern recovered is the binarization tree of* $\mathbf{a}$.

Proof. $\rightarrow$: it is obvious that if the algorithm succeeds then $\mathbf{a}$ is binarizable using the binarization pattern recovered.

$\leftarrow$: by a complete induction on n, the length of $\mathbf{a}$.

Base case: $n = 1$, trivial.

Assume it holds for all $n' < n$.

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

Let $\mathbf{c} = (\mathbf{c}_1; \mathbf{c}_2)$. If $\mathbf{c}_1$ is binarizable and triggers our binarizer to make a straight combination of $(\mathbf{b}; \mathbf{c}_1)$, based on the property of permutations, it must be true that $(\mathbf{c}_1; \mathbf{c}_2)$ is a valid straight concatenation. We claim that $\mathbf{c}_2$ must be binarizable in this situation. So, $(\mathbf{b}, \mathbf{c}_1; \mathbf{c}_2)$ is a binarizable split to the right of the rightmost binarizable split $(\mathbf{b}; \mathbf{c})$, which is a contradiction. A similar contradiction will arise if $\mathbf{b}$ and $\mathbf{c}_1$ can make an inverted concatenation.

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 $s[1]$ on the stack and recover its binarization tree $bi(\mathbf{c})$. Since $\mathbf{b}$ and $\mathbf{c}$ are combinable, the algorithm reduces $s[0]$ and $s[1]$ in the last step, forming the binarization tree for $\mathbf{a}$, which is either $[bi(\mathbf{b}), bi(\mathbf{c})]$ or $\langle bi(\mathbf{b}), bi(\mathbf{c})\rangle$. □

The running time of Algorithm 1 is linear in n, the length of the input sequence. This is because there are exactly n shifts and at most $n-1$ reductions, and each shift or reduction takes $O(1)$ time.

3.2 Binarizing tree-to-string transducers

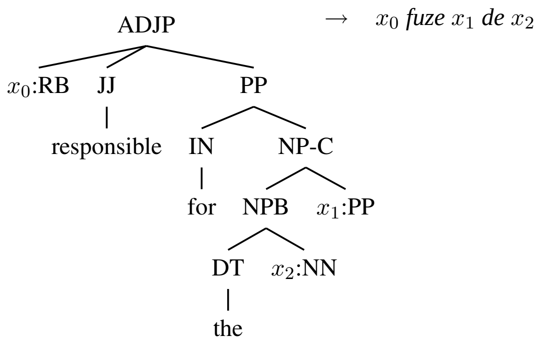
Without loss of generality, we have discussed how to binarize synchronous productions involving only nonterminals through binarizing the corresponding skeleton permutations. We still need to tackle a few technical problems in the actual system.

First, we are dealing with tree-to-string 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 tree-to-string derivation using the two-step SCFG derivation. For example,

Algorithm 1 The Linear-time Binarization Algorithm

```
1:  function BINARIZABLE($\mathbf{a}$)
2:      $top \leftarrow 0$                                   ▷ stack top pointer
3:      PUSH($a_1, a_1$)                                    ▷ initial shift
4:      for $i \leftarrow 2$ to $|\mathbf{a}|$ do              ▷ for each remaining element
5:          PUSH($a_i, a_i$)                                ▷ shift
6:          while $top > 1$ and CONSECUTIVE($s[top], s[top-1]$) do   ▷ keep reducing if possible
7:              $(p, q) \leftarrow$ COMBINE($s[top], s[top-1]$)
8:              $top \leftarrow top - 2$
9:              PUSH($p, q$)
10:     return $(top = 1)$    ▷ if reduced to a single element then the input is binarizable, otherwise not
11: function CONSECUTIVE($(a, b), (c, d)$)
12:     return $(b = c - 1)$ or $(d = a - 1)$               ▷ either straight or inverted
13: function COMBINE($(a, b), (c, d)$)
14:     return $(min(a, c), max(b, d))$
```

consider the following tree-to-string rule:



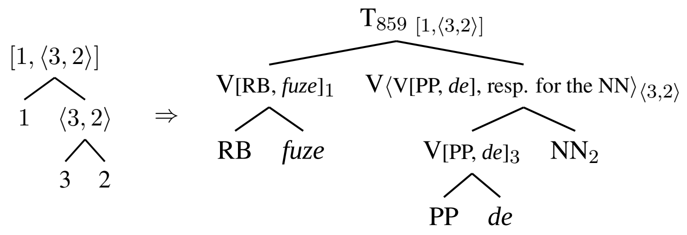
We create a specific nonterminal, say, T_{859}, which is a unique identifier for the left-hand side subtree and generate the following two SCFG rules:

$$\text{ADJP} \rightarrow T_{859}{}^{(1)},\ T_{859}{}^{(1)}$$

$$T_{859} \rightarrow \begin{array}{l}\text{RB}^{(1)} \text{ resp. for the NN}^{(2)}\ \text{PP}^{(3)},\\ \text{RB}^{(1)}\ \textit{fuze}\ \text{PP}^{(3)}\ \textit{de}\ \text{NN}^{(2)}\end{array}$$

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

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



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

$$\begin{array}{lcl}
T_{859} & \rightarrow & V_1{}^{(1)}\ V_2{}^{(2)},\ V_1{}^{(1)}\ V_2{}^{(2)}\\
V_1 & \rightarrow & \text{RB}^{(1)},\ \text{RB}^{(1)}\ \textit{fuze}\\
V_2 & \rightarrow & \text{resp. for the NN}^{(1)}\ V_3^{(2)},\ V_3^{(2)}\ \text{NN}^{(1)}\\
V_3 & \rightarrow & \text{PP}^{(1)},\ \text{PP}^{(1)}\ \textit{de}
\end{array}$$

where the virtual nonterminals are:

V_1: V[RB, *fuze*]
V_2: V⟨V[PP, *de*], resp. for the NN⟩
V_3: V[PP, *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 tree-to-string 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).

4 Experiments

In this section, we answer two empirical questions.

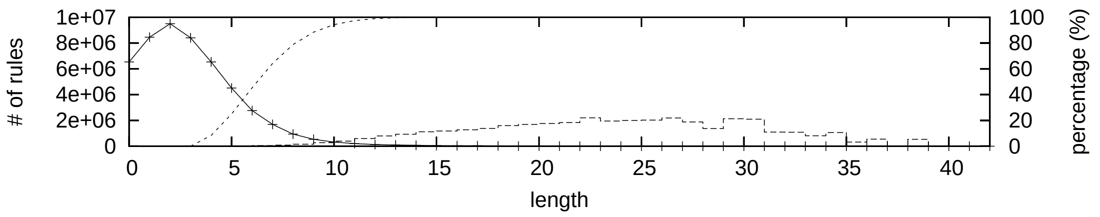


Figure 6: The solid-line curve represents the distribution of all rules against permutation lengths. The dashed-line stairs indicate the percentage of non-binarizable rules in our initial rule set while the dotted-line denotes that percentage among all permutations.

4.1 How many rules are binarizable?

It has been shown by Shapiro and Stephens (1991) and Wu (1997, Sec. 4) that the percentage of binarizable cases over all permutations of length n quickly approaches 0 as n grows (see Figure 6). However, for machine translation, it is more meaningful to compute the ratio of binarizable rules extracted from real text. Our rule set is obtained by first doing word alignment using GIZA++ on a Chinese-English parallel corpus containing 50 million words in English, then parsing the English sentences using a variant of Collins parser, and finally extracting rules using the graph-theoretic algorithm of Galley et al. (2004). We did a "spectrum analysis" on the resulting rule set with 50,879,242 rules. Figure 6 shows how the rules are distributed against their lengths (number of nonterminals). We can see that the percentage of non-binarizable rules in each bucket of the same length does not exceed 25%. Overall, 99.7% of the rules are binarizable. Even for the 0.3% non-binarizable rules, human evaluations show that the majority of them are due to alignment errors. It is also interesting to know that 86.8% of the rules have monotonic permutations, i.e. either taking identical or totally inverted order.

4.2 Does synchronous binarizer help decoding?

We did experiments on our CKY-based decoder with two binarization methods. It is the responsibility of the binarizer to instruct the decoder how to compute the language model scores from children nonterminals in each rule. The baseline method is monolingual left-to-right binarization. As shown in Section 1, decoding complexity with this method is exponential in the size of the longest rule and since we postpone all the language model scorings, pruning in this case is also biased.

system	bleu
monolingual binarization	36.25
synchronous binarization	*38.44*
alignment-template system	37.00

Table 1: Syntax-based systems vs. ATS

To move on to synchronous binarization, we first did an experiment using the above baseline system without the 0.3% non-binarizable rules and did not observe any difference in BLEU scores. So we safely move a step further, focusing on the binarizable rules only.

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

We compare the two binarization schemes in terms of translation quality with various pruning thresholds. The rule set is that of the previous section. The test set has 116 Chinese sentences of no longer than 15 words. Both systems use trigram as the integrated language model. Figure 7 demonstrates that decoding accuracy is significantly improved after synchronous binarization. The number of edges proposed during decoding is used as a measure of the size of search space, or time efficiency. Our system is consistently faster and more accurate than the baseline system.

We also compare the top result of our synchronous binarization system with the state-of-the-art alignment-template approach (ATS) (Och and Ney, 2004). The results are shown in Table 1. Our system has a promising improvement over the ATS

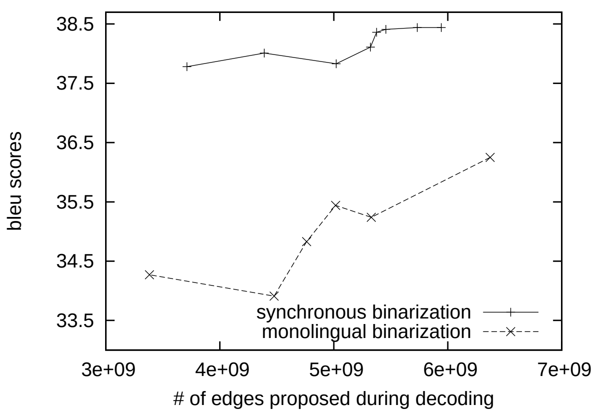


Figure 7: Comparing the two binarization methods in terms of translation quality against search effort.

system which is trained on a larger data-set but tuned independently.

5 Conclusion

Modeling reorderings between languages has been a major challenge for machine translation. This work shows that the majority of syntactic reorderings, at least between languages like English and Chinese, can be efficiently decomposed into hierarchical binary reorderings. From a modeling perspective, on the other hand, it is beneficial to start with a richer representation that has more transformational power than ITG or binary SCFG. Our work shows how to convert it back to a computationally friendly form without harming much of its expressiveness. 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. We believe that extensions of our technique to more powerful models such as synchronous tree-adjoining grammar (Shieber and Schabes, 1990) is an interesting area for further work.

Acknowledgments Much of this work was done when H. Zhang and L. Huang were visiting USC/ISI. The authors wish to thank Wei Wang, Jonathan Graehl and Steven DeNeefe for help with the experiments. We are also grateful to Daniel Marcu, Giorgio Satta, and Aravind Joshi for discussions. This work was partially supported by NSF ITR IIS-09325646 and NSF ITR IIS-0428020.

References

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

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of ACL-05*, pages 263–270, Ann Arbor, Michigan.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of ACL-03, companion volume*, Sapporo, Japan.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of HLT/NAACL-04*.

Liang Huang, Hao Zhang, and Daniel Gildea. 2005. Machine translation as lexicalized parsing with hooks. In *Proceedings of IWPT-05*, Vancouver, BC.

Kevin Knight and Jonathan Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In *Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*. LNCS.

I. Dan Melamed. 2003. Multitext grammars and synchronous parsers. In *Proceedings of NAACL-03*, Edmonton.

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

Giorgio Satta and Enoch Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Proceedings of HLT/EMNLP-05*, pages 803–810, Vancouver, Canada, October.

L. Shapiro and A. B. Stephens. 1991. Bootstrap percolation, the Schröder numbers, and the n-kings problem. *SIAM Journal on Discrete Mathematics*, 4(2):275–280.

Stuart Shieber and Yves Schabes. 1990. Synchronous tree-adjoining grammars. In *COLING-90*, volume III, pages 253–258.

Dekai Wu. 1996. A polynomial-time algorithm for statistical machine translation. In *34th Annual Meeting of the Association for Computational Linguistics*.

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