# Terminal-Aware Synchronous Binarization

**Licheng Fang**, **Tagyoung Chung** and **Daniel Gildea**
Department of Computer Science
University of Rochester
Rochester, NY 14627

## Abstract

We present an SCFG binarization algorithm that combines the strengths of early terminal matching on the source language side and early language model integration on the target language side. We also examine how different strategies of target-side terminal attachment during binarization can significantly affect translation quality.

## 1 Introduction

Synchronous context-free grammars (SCFG) are behind most syntax-based machine translation models. Efficient machine translation decoding with an SCFG requires converting the grammar into a binarized form, either explicitly, as in synchronous binarization (Zhang et al., 2006), where virtual nonterminals are generated for binarization, or implicitly, as in Earley parsing (Earley, 1970), where dotted items are used.

Given a source-side binarized SCFG with terminal set $\mathcal{T}$ and nonterminal set $\mathcal{N}$, the time complexity of decoding a sentence of length $n$ with a $m$-gram language model is (Venugopal et al., 2007):

$$\mathcal{O}(n^3(|\mathcal{N}| \cdot |\mathcal{T}|^{2(m-1)})^K)$$

where $K$ is the maximum number of right-hand-side nonterminals. SCFG binarization serves two important goals:

- Parsing complexity for unbinarized SCFG grows exponentially with the number of nonterminals on the right-hand side of grammar rules. Binarization ensures cubic time decoding in terms of input sentence length.
- In machine translation, integrating language model states as early as possible is essential to reducing search errors. Synchronous binarization (Zhang et al., 2006) enables the decoder to incorporate language model scores as soon as a binarized rule is applied.

In this paper, we examine a CYK-like synchronous binarization algorithm that integrates a novel criterion in a unified semiring parsing framework. The criterion we present has explicit consideration of source-side terminals. In general, terminals in a rule have a lower probability of being matched given a sentence, and therefore have the effect of "anchoring" a rule and limiting its possible application points. Hopkins and Langmead (2010) formalized this concept as the *scope* of a rule. A rule of scope of $k$ can be parsed in $O(n^k)$. The scope of a rule can be calculated by counting the number of adjacent nonterminal pairs and boundary nonterminals. For example,

$$A \rightarrow w_1BCw_2D$$

has scope two. Building on the concept of scope, we define a cost function that estimates the expected number of hyperedges to be built when a particular binarization tree is applied to unseen data. This effectively puts hard-to-match derivations at the bottom of the binarization tree, which enables the decoder to decide early on whether an unbinarized rule can be built or not.

We also investigate a better way to handle target-side terminals during binarization. In theory, different strategies should produce equivalent translation results. However, because decoding always involves



*Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics:shortpapers*, pages 401–406,
Portland, Oregon, June 19-24, 2011. 

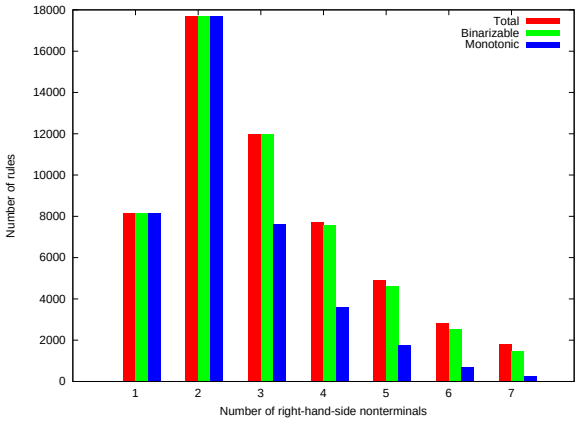


Figure 1: Rule Statistics

pruning, we show that different strategies do have a significant effect in translation quality.

Other works investigating alternative binarization methods mostly focus on the effect of nonterminal sharing. Xiao et al. (2009) also proposed a CYK-like algorithm for synchronous binarization. Apparently the lack of virtual nonterminal sharing in their decoder caused heavy competition between virtual nonterminals, and they created a cost function to "diversify" binarization trees, which is equivalent to minimizing nonterminal sharing.

DeNero et al. (2009b) used a greedy method to maximize virtual nonterminal sharing on the source side during the -LM parsing phase. They show that effective source-side binarization can improve the efficiency of parsing SCFG. However, their method works only on the source side, and synchronous binarization is put off to the +LM decoding phase (DeNero et al., 2009a).

Although these ideas all lead to faster decoding and reduced search errors, there can be conflicts in the constraints each of them has on the form of rules and accommodating all of them can be a challenge. In this paper, we present a cubic time algorithm to find the best binarization tree, given the conflicting constraints.

## 2 The Binarization Algorithm

An SCFG rule is synchronously binarizable if when simultaneously binarizing source and target sides, virtual nonterminals created by binarizations always have contiguous spans on both sides (Huang, 2007).

**Algorithm 1** The CYK binarization algorithm.

CYK-BINARIZE($X \rightarrow \langle \gamma, \alpha \rangle$)
  **for** $i = 0 \ldots |\gamma| - 1$ **do**
    $T[i, i+1] \leftarrow c_{init}(i)$
  **for** $s = 2 \ldots |\gamma|$ **do**
    **for** $i = 0 \ldots |\gamma|$-1 **do**
      $j \leftarrow i + s$
      **for** $k = i + 1 \ldots j - 1$ **do**
        $t \leftarrow T[i, k] + T[k, j] + c(\langle i, k, j \rangle)$
        $T[i, j] \leftarrow \min(T[i, j], t)$

Even with the synchronous binarization constraint, many possible binarizations exist. Analysis of our Chinese-English parallel corpus has shown that the majority of synchronously binarizable rules with arity smaller than 4 are *monotonic*, i.e., the target-side nonterminal permutation is either strictly increasing or decreasing (See Figure 1). For monotonic rules, any source-side binarization is also a permissible synchronous binarization.

The binarization problem can be formulated as a semiring parsing (Goodman, 1999) problem. We define a cost function that considers different binarization criteria. A CYK-like algorithm can be used to find the best binarization tree according to the cost function. Consider an SCFG rule $X \rightarrow \langle \gamma, \alpha \rangle$, where $\gamma$ and $\alpha$ stand for the source side and the target side. Let $B(\gamma)$ be the set of all possible binarization trees for $\gamma$. With the cost function $c$ defined over hyperedges in a binarization tree $t$, the optimal binarization tree $\hat{t}$ is

$$\hat{t} = \underset{t \in B(\gamma)}{\operatorname{argmin}} \sum_{h \in t} c(h)$$

where $c(h)$ is the cost of a hyperedge $h$ in $t$.

The optimization problem can be solved by Algorithm 1. $\langle i, k, j \rangle$ denotes a hyperedge $h$ that connects the spans $(i, k)$ and $(k, j)$ to the span $(i, j)$. $c_{init}$ is the initialization for the cost function $c$. We can recover the optimal source-side binarization tree by augmenting the algorithm with back pointers. Binarized rules are generated by iterating over the nodes in the optimal binarization tree, while attaching unaligned target-side terminals. At each tree node, we generate a virtual nonterminal symbol by concatenating the source span it dominates.

We define the cost function $c(h)$ to be a tuple of component cost functions: $c(h) =$

$(c_1(h), c_2(h), ...)$. When two costs $a$ and $b$ are compared, the components are compared piecewise, i.e.

$$c < c' \Leftrightarrow c_1 < c'_1 \vee (c_1 = c'_1 \wedge c_2 < c'_2) \vee \ldots$$

If the $(\min, +)$ operators on each component cost satisfy the semiring properties, the cost tuple is also a semiring. Next, we describe our cost functions and how we handle target-side terminals.

### 2.1 Synchronous Binarization as a Cost

We use a binary cost $b$ to indicate whether a binarization tree is a permissible synchronous binarization. Given a hyperedge $\langle i, k, j \rangle$, we say $k$ is a *permissible* split of the span $(i, j)$ if and only if the spans $(i, k)$ and $(k, j)$ are both synchronously binarizable and the span $(i, j)$ covers a consecutive sequence of nonterminals on the target side. A span is *synchronously binarizable* if and only if the span is of length one, or a permissible split of the span exists. The cost $b$ is defined as:

$$b(\langle i, k, j \rangle) = \begin{cases} T & \text{if } k \text{ is a permissible split of } (i, j) \\ F & \text{otherwise} \end{cases}$$
$$b_{init}(i) = T$$

Under this configuration, the semiring operators $(\min, +)$ defined for the cost $b$ are $(\vee, \wedge)$. Using $b$ as the first cost function in the cost function tuple guarantees that we will find a tree that is a synchronously binarized if one exists.

### 2.2 Early Source-Side Terminal Matching

When a rule is being applied while parsing a sentence, terminals in the rule have less chance of being matched. We can exploit this fact by taking terminals into account during binarization and placing terminals lower in the binarization tree. Consider the following SCFG rule:

VP → PP 提出 JJ NN,
propose a JJ NN PP

The synchronous binarization algorithm of Zhang et al. (2006) binarizes the rule[1] by finding the rightmost binarizable points on the source side:

VP → PP [提出 [JJ NN]$_1$]$_2$,
[[propose a JJ NN]$_1$]$_2$ PP

The source side of the first binarized rule "[]$_1$ → JJ NN, propose a JJ NN" contains a very frequent nonterminal sequence "JJ NN". If one were to parse with the binarized rule, and if the virtual nonterminal []$_1$ has been built, the parser needs to continue following the binarization tree in order to determine whether the original rule would be matched. Furthermore, having two consecutive nonterminals adds to complexity since the parser needs to test each split point.

The following binarization is equally valid but integrates terminals early:

VP → PP [[提出 JJ]$_1$ NN]$_2$,
[[propose a JJ]$_1$ NN]$_2$ PP

Here, the first binarized rule "[]$_1$ → 提出 JJ, propose a JJ" anchors on a terminal and enables earlier pruning of the original rule.

We formulate this intuition by asking the question: given a source-side string $\gamma$, what binarization tree, on average, builds the smallest number of hyperedges when the rule is applied? This is realized by defining a cost function $e$ which estimates the probability of a hyperedge $\langle i, k, j \rangle$ being built. We use a simple model: assume each terminal or nonterminal in $\gamma$ is matched independently with a fixed probability, then a hyperedge $\langle i, k, j \rangle$ is derived if and only if all symbols in the source span $(i, j)$ are matched. The cost $e$ is thus defined as[2]

$$e(\langle i, k, j \rangle) = \prod_{i \le \ell < j} p(\gamma_\ell)$$
$$e_{init}(i) = 0$$

For terminals, $p(\gamma_\ell)$ can be estimated by counting the source side of the training corpus. For nonterminals, we simply assume $p(\gamma_\ell) = 1$.

With the hyperedge cost $e$, the cost of a binarization tree $t$ is $\sum_{h \in t} e(h)$, i.e., the expected number of hyperedges to be built when a particular binarization of a rule is applied to unseen data.[3] The operators

[1] We follow Wu (1997) and use square brackets for straight rules and pointed brackets for inverted rules. We also mark brackets with indices to represent virtual nonterminals.

[2] In this definition, $k$ does not appear on the right-hand side of the equation because all edges leading to the same span share the same cost value.

[3] Although this cost function is defined as an expectation, it does not form an *expectation semiring* (Eisner, 2001) because

for the cost $e$ are the usual $(\min, +)$ operators on real numbers.

### 2.3 Maximizing Nonterminal Sharing

During binarization, newly created virtual nonterminals are named according to the symbols (terminals and nonterminals) that they generate. For example, a new virtual nonterminal covering two nonterminals NP and VP is named NP+VP. To achieve maximum virtual nonterminal sharing, we also define a cost function $n$ to count the number new nonterminals generated by a binarization tree. We keep track of all the nonterminals that have been generated when binarizing a rule set. When the $i$'th rule is being binarized, a nonterminal is considered new if it is previously unseen in binarizing rules 1 to $i-1$. This greedy approach is similar to that of DeNero et al. (2009b). The cost function is thus defined as:

$$n(\langle i,k,j\rangle) = \begin{cases} 1 & \text{if the VT for span } (i,j) \text{ is new} \\ 0 & \text{otherwise} \end{cases}$$

$$n_{init}(i) = 0$$

The semiring operators for this cost are also $(\min, +)$ on real numbers.

### 2.4 Late Target-Side Terminal Attachment

Once the optimal source-side binarization tree is found, we have a good deal of freedom to attach target-side terminals to adjacent nonterminals, as long as the bracketing of nonterminals is not violated. The following example is taken from Zhang et al. (2006):

$$\text{ADJP} \rightarrow \begin{array}{c} \text{RB 负责 PP 的 NN,} \\ \text{RB responsible for the NN PP} \end{array}$$

With the source-side binarization fixed, we can produce distinct binarized rules by choosing different ways of attaching target-side terminals:

$$\text{ADJP} \rightarrow \begin{array}{c} [\text{RB 负责}]_1 \langle\, [\text{PP 的}]_3 \text{ NN} \,\rangle_2, \\ [\text{RB}]_1 \langle \text{ resp. for the NN } [\text{PP}]_3 \,\rangle_2 \end{array}$$

$$\text{ADJP} \rightarrow \begin{array}{c} [\text{RB 负责}]_1 \langle\, [\text{PP 的}]_3 \text{ NN} \,\rangle_2, \\ [\text{RB}]_1 \text{ resp. for the } \langle \text{ NN } [\text{PP}]_3 \,\rangle_2 \end{array}$$

The first binarization is generated by attaching the target-side terminals as low as possible in a post-order traversal of the binarization tree. The conventional wisdom is that early consideration of target-side terminals promotes early language model score integration (Huang et al., 2009). The second binarization, on the contrary, attaches the target-side terminals as high as possible in the binarization tree. We argue that this late target-side terminal attachment is in fact better for two reasons.

First, as in the example above, compare the following two rules resulting from early attachment of target terminals and late attachment of target terminals:

$$\langle\rangle_2 \rightarrow []_3 \text{ NN, resp. for the NN } []_3$$

$$\langle\rangle_2 \rightarrow []_3 \text{ NN, NN } []_3$$

The former has a much smaller chance of sharing the same target side with other binarized rules because on the target side, many nonterminals will be attached without any lexical evidence. We are more likely to have a smaller set of rules with the latter binarization.

Second, with the presence of pruning, dynamic programming states that are generated by rules with many target-side terminals are disadvantaged when competing with others in the same bin because of the language model score. As a result, these would be discarded earlier, even if the original unbinarized rule has a high probability. Consequently, we lose the benefit of using larger rules, which have more contextual information. We show in our experiment that late target side terminal attachment significantly outperforms early target side terminal attachment.

Although the problem can be alleviated by precomputing a language model score for the original unbinarized rule and applying the heuristic to its binarized rules, this still grants no benefit over late terminal attachment. We show in our experiment that late target-side terminal attachment significantly outperforms early target side terminal attachment.

## 3 Experiments

### 3.1 Setup

We test our binarization algorithm on an Chinese-English translation task. We extract a GHKM grammar (Galley et al., 2004) from a parallel corpus with the parsed English side with some modification so

it is defined as an expectation over input strings, instead of an expectation over trees.

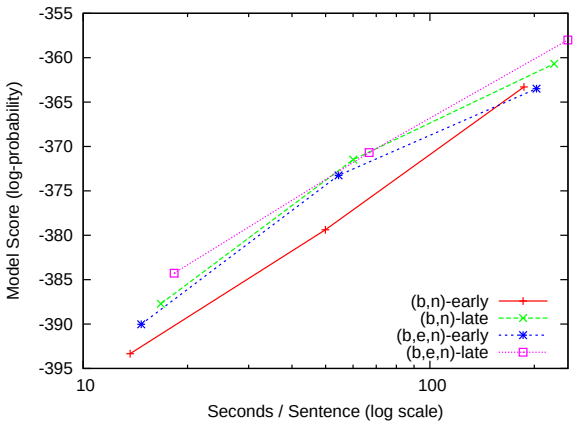


Figure 2: Model Scores vs. Decoding Time

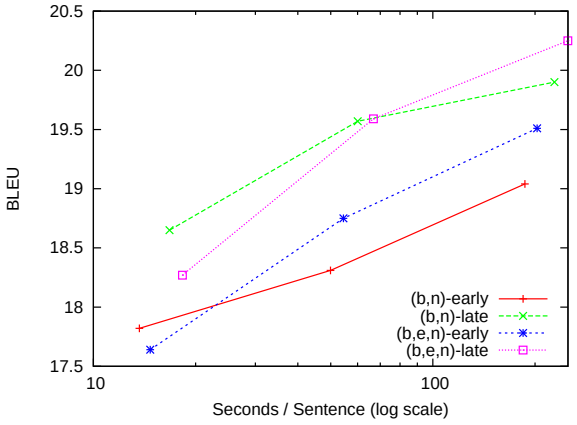


Figure 3: BLEU Scores vs Decoding Time

as not to extract unary rules (Chung et al., 2011). The corpus consists of 250K sentence pairs, which is 6.3M words on the English side. A 392-sentence test set was to evaluate different binarizations.

Decoding is performed by a general CYK SCFG decoder developed in-house and a trigram language model is used. The decoder runs the CYK algorithm with cube-pruning (Chiang, 2007). In all our experiments, we discard unbinarizable rules, which have been shown by Zhang et al. (2006) to have no significant effect on translation accuracy.

### 3.2 Results

We first discuss effects of maximizing nonterminal sharing. Having nonterminal sharing maximization as a part of the cost function for binarization did yield slightly smaller grammars. However, we could not discern any noticeable difference or trend in terms of BLEU score, decoding speed, or model score when comparing translation results that used grammars that employed nonterminal sharing maximization and ones that did not. In the rest of this section, all the results we discuss use nonterminal sharing maximization as a part of the cost function.

We then compare the effects of early target-side terminal attachment and late attachment. Figure 2 shows model scores of each decoder run with varying bin sizes, and Figure 3 shows BLEU scores for corresponding runs of the experiments. (b,n)-early is conventional synchronous binarization with early target-side terminal attachment and nonterminal sharing maximization, (b,n)-late is the same setting with late target-side terminal attachment. The tuples represent cost functions that are discussed in Section 2. The figures clearly show that late attachment of target-side terminals is better. Although Figure 3 does not show perfect correlation with Figure 2, it exhibits the same trend. The same goes for (b,e,n)-early and (b,e,n)-late.

Finally, we examine the effect of including the source-side terminal-aware cost function, denoted "e" in our cost tuples. Comparing (b,e,n)-late with (b,n)-late, we see that terminal-aware binarization gives better model scores and BLEU scores. The trend is the same when one compares (b,e,n)-early and (b,n)-early.

## 4 Conclusion

We examined binarizing synchronous context-free grammars within a semiring parsing framework. We proposed binarization methods that explicitly take terminals into consideration. We have found that although binarized rules are already scope 3, we can still do better by putting infrequent derivations as low as possible in a binarization tree to promote early pruning. We have also found that attaching target side terminals as late as possible promotes smarter pruning of rules thereby improving model score and translation quality at decoding time. Improvements we discuss in this paper result in better search, and hence better translation.

**Acknowledgments** We thank Hao Zhang for useful discussions and the anonymous reviewers for their helpful comments. This work was supported by NSF grants IIS-0546554 and IIS-0910611.

## References


David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

Tagyoung Chung, Licheng Fang, and Daniel Gildea. 2011. Issues concerning decoding with synchronous context-free grammar. In *Proceedings of the ACL 2011 Conference Short Papers*, Portland, Oregon, June. Association for Computational Linguistics.

J. DeNero, A. Pauls, and D. Klein. 2009a. Asynchronous binarization for synchronous grammars. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 141–144. Association for Computational Linguistics.

John DeNero, Mohit Bansal, Adam Pauls, and Dan Klein. 2009b. Efficient parsing for transducer grammars. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 227–235, Boulder, Colorado, June. Association for Computational Linguistics.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 6(8):451–455.

J. Eisner. 2001. Expectation semirings: Flexible EM for learning finite-state transducers. In *Proceedings of the ESSLLI workshop on finite-state methods in NLP*. Citeseer.

Michel Galley, 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.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–605.

Mark Hopkins and Greg Langmead. 2010. SCFG decoding without binarization. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 646–655, Cambridge, MA, October. Association for Computational Linguistics.

Liang Huang, Hao Zhang, Daniel Gildea, and Kevin Knight. 2009. Binarization of synchronous context-free grammars. *Computational Linguistics*, 35(4):559–595.

Liang Huang. 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.

Ashish Venugopal, Andreas Zollmann, and Stephan Vogel. 2007. An efficient two-pass approach to synchronous-CFG driven statistical MT. In *NAACL07*, Rochester, NY, April.

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

T. Xiao, M. Li, D. Zhang, J. Zhu, and M. Zhou. 2009. Better synchronous binarization for machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 362–370. Association for Computational Linguistics.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proceedings of the 2006 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-06)*, pages 256–263, New York, NY.