Better Synchronous Binarization for Machine Translation

Tong Xiao[*], Mu Li[+], Dongdong Zhang[+], Jingbo Zhu[*], Ming Zhou[+]

[*]Natural Language Processing Lab
Northeastern University
Shenyang, China, 110004
xiaotong@mail.neu.edu.cn
zhujingbo@mail.neu.edu.cn

[+]Microsoft Research Asia
Sigma Center
Beijing, China, 100080
muli@microsoft.com
dozhang@microsoft.com
mingzhou@microsoft.com

Abstract

Binarization of Synchronous Context Free Grammars (SCFG) is essential for achieving polynomial time complexity of decoding for SCFG parsing based machine translation systems. In this paper, we first investigate the excess edge competition issue caused by a left-heavy binary SCFG derived with the method of Zhang et al. (2006). Then we propose a new binarization method to mitigate the problem by exploring other alternative equivalent binary SCFGs. We present an algorithm that iteratively improves the resulting binary SCFG, and empirically show that our method can improve a string-to-tree statistical machine translations system based on the synchronous binarization method in Zhang et al. (2006) on the NIST machine translation evaluation tasks.

1 Introduction

Recently Statistical Machine Translation (SMT) systems based on Synchronous Context Free Grammar (SCFG) have been extensively investigated (Chiang, 2005; Galley et al., 2004; Galley et al., 2006) and have achieved state-of-the-art performance. In these systems, machine translation decoding is cast as a synchronous parsing task. Because general SCFG parsing is an NP-hard problem (Satta and Peserico, 2005), practical SMT decoders based on SCFG parsing requires an equivalent binary SCFG that is directly learned from training data to achieve polynomial time complexity using the CKY algorithm (Kasami, 1965; Younger, 1967) borrowed from CFG parsing techniques. Zhang et al. (2006) proposed *synchronous binarization*, a principled method to binarize an SCFG in such a way that both the source-side and target-side virtual non-terminals have contiguous spans. This property of synchronous binarization guarantees the polynomial time complexity of SCFG parsers even when an *n*-gram language model is integrated, which has been proved to be one of the keys to the success of a string-to-tree syntax-based SMT system.

However, as shown by Chiang (2007), SCFG-based decoding with an integrated *n*-gram language model still has a time complexity of $\Theta(m^3|T|^{4(n-1)})$, where m is the source sentence length, and $|T|$ is the vocabulary size of the language model. Although it is not exponential in theory, the actual complexity can still be very high in practice. Here is an example extracted from real data. Given the following SCFG rule:

$$\begin{aligned} \boldsymbol{VP} \;\rightarrow\; & \boldsymbol{VB}\ \boldsymbol{NP}\ 会\ \boldsymbol{JJR}\ ,\\ & \boldsymbol{VB}\ \boldsymbol{NP}\ \text{will be}\ \boldsymbol{JJR} \end{aligned}$$

we can obtain a set of equivalent binary rules using the synchronous binarization method (Zhang et al., 2006) as follows:

$$\begin{array}{ll} \boldsymbol{VP} \rightarrow \boldsymbol{V}_1\ \boldsymbol{JJR}\ , & \boldsymbol{V}_1\ \boldsymbol{JJR} \\ \boldsymbol{V}_1 \rightarrow \boldsymbol{VB}\ \boldsymbol{V}_2\ , & \boldsymbol{VB}\ \boldsymbol{V}_2 \\ \boldsymbol{V}_2 \rightarrow \boldsymbol{NP}\ 会\ , & \boldsymbol{NP}\ \text{will be} \end{array}$$

This binarization is shown with the solid lines as binarization (a) in Figure 1. We can see that binarization (a) requires that “NP 会” should be reduced at first. Data analysis shows that “NP 会” is a frequent pattern in the training corpus, and there are 874 binary rules of which the source language sides are “NP 会”. Consequently these binary rules generate a large number of competing edges in the chart when “NP 会” is matched in decoding. To reduce the number of edges proposed in decoding, hypothesis re-combination is used to combine the equivalent edges in terms of dynamic programming. Generally, two edges can be re-combined if they satisfy the following two constraints: 1) the LHS (left-hand side) non-terminals are identical and the sub-alignments are the same (Zhang et al., 2006); and 2) the boundary words[1] on both sides of the partial translations are equal between the two edges (Chiang, 2007). However, as shown in Figure 2, the decoder still generates 801 edges after the hypothesis re-combination. As a result, aggressive pruning with beam search has to be employed to reduce the search space to make the decoding practical. Usually in beam search only a very small number of edges are kept in the beam of each chart cell (e.g. less than 100). These edges have to compete with each other to survive from the pruning. Obviously, more competing edges proposed during decoding can lead to a higher risk of making search errors.

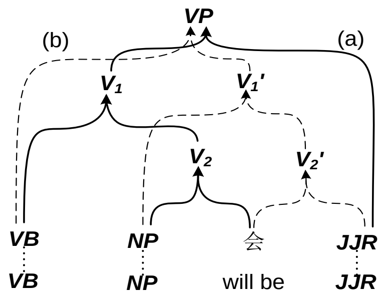

Figure 1: Two different binarizations (a) and (b) of the same SCFG rule distinguished by the solid lines and dashed lines

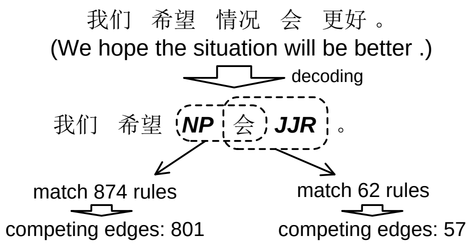

Figure 2: Edge competitions caused by different binarizations

The edge competition problem for SMT decoding is not addressed in previous work (Zhang et al., 2006; Huang, 2007) in which each SCFG rule is binarized in a fixed way. Actually the results of synchronous binarization may not be the only solution. As illustrated in Figure 1, the rule can also be binarized as binarization (b) which is shown with the dashed lines.

We think that this problem can be alleviated by choosing better binarizations for SMT decoders, since there is generally more than one binarization for a SCFG rule. In our investigation, about 96% rules that need to be binarized have more than one binarization under the contiguous constraint. As shown in binarization (b) (Figure 1), "会 JJR" is reduced first. In the decoder, the number of binary rules with the source-side "会 JJR" is 62, and the corresponding number of edges is 57 (Figure 2). The two numbers are both much smaller than those of "NP 会" in (a). This is an informative clue that the binarization (b) could be better than the binarization (a) based on the following: the probability of pruning the rule in (a) is higher than that in (b) as the rule in (b) has fewer competitors and has more chances to survive during pruning.

In this paper we propose a novel binarization method, aiming to find better binarizations to improve an SCFG-based machine translation system. We formulate the binarization optimization as a cost reduction process, where the cost is defined as the number of rules sharing a common source-side derivation in an SCFG. We present an algorithm, *iterative cost reduction* algorithm, to obtain better binarization for the SCFG learnt automatically from the training corpus. It can work with an efficient CKY-style binarizer to search for the lowest-cost binarization. We apply our method into a state-of-the-art string-to-tree SMT system. The experimental results show that our method outperforms the synchronous binarization method (Zhang et al., 2006) with over 0.8 BLEU scores on both NIST 2005 and NIST 2008 Chinese-to-English evaluation data sets.

2 Related Work

The problem of binarization originates from the parsing problem in which several binarization methods are studied such as left/right binarization (Charniak et al., 1998; Tsuruoka and Tsujii, 2004) and head binarization (Charniak et al., 2006). Generally, the pruning issue in SMT decoding is unnecessary for the parsing problem, and the accuracy of parsing does not rely on the binarization method heavily. Thus, many efforts on the binarization in parsing are made for the efficiency improvement instead of the accuracy improvement (Song et al., 2008).

Binarization is also an important topic in the research of syntax-based SMT. A synchronous

[1] For the case of n-gram language model integration, $2 \times (n-1)$ boundary words needs to be examined.

binarization method is proposed in (Zhang et al., 2006) whose basic idea is to build a left-heavy binary synchronous tree (Shapiro and Stephens, 1991) with a left-to-right shift-reduce algorithm. Target-side binarization is another binarization method which is proposed by Huang (2007). It works in a left-to-right way on the target language side. Although this method is comparatively easy to be implemented, it just achieves the same performance as the synchronous binarization method (Zhang et al., 2006) for syntax-based SMT systems. In addition, it cannot be easily integrated into the decoding of some syntax-based models (Galley et al., 2004; Marcu et al., 2006), because it does not guarantee contiguous spans on the source language side.

3 Synchronous Binarization Optimization by Cost Reduction

As discussed in Section 1, binarizing an SCFG in a fixed (left-heavy) way (Zhang et al., 2006) may lead to a large number of competing edges and consequently high risk of making search errors. Fortunately, in most cases a binarizable SCFG can be binarized in different ways, which provides us with an opportunity to find a better solution than the default left-heavy binarization. An ideal solution to this problem could be that we define an exact edge competition estimation function and choose the best binary SCFG based on it. However, even for the rules with a common source-side, generally it is difficult to estimate the exact number of competing edges in the dynamic SCFG parsing process for machine translation, because in order to integrate an n-gram language model, the actual number of edges not only depends on SCFG rules, but also depends on language model states which are specific to input sentences. Instead, we have to employ certain kinds of approximation of it. First we will introduce some notations frequently used in later discussions.

3.1 Notations

We use $G = \{R_i : X_i \rightarrow \alpha_i, \beta_i\}$ to denote an SCFG, where R_i is the i^{th} rule in G; X_i is the LHS (left hand side) non-terminal of R_i; α_i and β_i are the source-side and target-side RHS (right hand side) derivations of R_i respectively. We use $\mathcal{B}(G)$ to denote the set of equivalent binary SCFG of G. The goal of SCFG binarization is to find an appropriate binary SCFG $G' \in \mathcal{B}(G)$. For R_i , $\mathcal{B}(R_i) = \{v_{ij}\} \subseteq G' \in \mathcal{B}(G)$ is the set of equivalent binary rules based on R_i, where v_{ij} is the j^{th} binary rule in $\mathcal{B}(R_i)$. Figure 3 illustrates the meanings of these notations with a sample grammar.

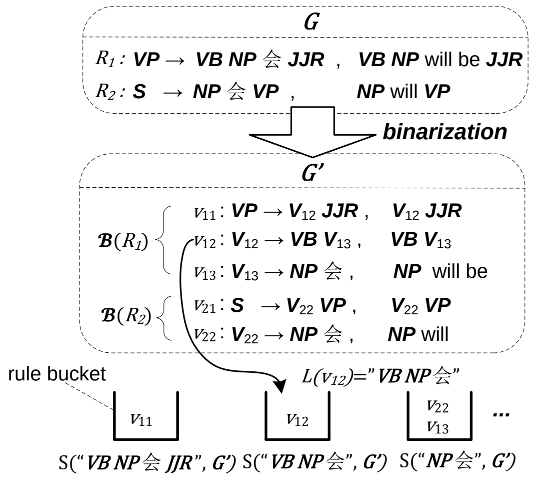
G
R_1: VP → VB NP 会 JJR , VB NP will be JJR
R_2: S → NP 会 VP , NP will VP
binarization
G'
$\mathcal{B}(R_1)$
v_{11}: VP → V_{12} JJR , V_{12} JJR
v_{12}: V_{12} → VB V_{13} , VB V_{13}
v_{13}: V_{13} → NP 会 , NP will be
$\mathcal{B}(R_2)$
v_{21}: S → V_{22} VP , V_{22} VP
v_{22}: V_{22} → NP 会 , NP will
rule bucket
$L(v_{12})$=" VB NP 会"
v_{11}
v_{12}
v_{22}
v_{13}
...
S(" VB NP 会 JJR", G')
S(" VB NP 会", G')
S(" NP 会", G')

Figure 3: Binarization on a sample grammar

The function $L(\cdot)$ is defined to map a resulting binary rule $v_{ij} \epsilon G'$ to the sub-sequence in α_i derived from v_{ij}. For example, as shown in Figure 3, the binary rule v_{13} covers the source sub-sequence "NP 会" in R_1, so $L(v_{13}) =$ "NP 会". Similarly, $L(v_{12}) =$ "VB NP 会".

The function $L(\cdot)$ is used to group the rules in G' with a common right-hand side derivation for source language. Given a binary rule $v \in G'$, we can put it into a *bucket* in which all the binary rules have the same source sub-sequence $L(v)$. For example (Figure 3), as $L(v_{12}) =$ "VB NP 会", v_{12} is put into the bucket indexed by "VB NP 会". And v_{13} and v_{22} are put into the same bucket, since they have the same source sub-sequence "NP 会". Obviously, G' can be divided into a set of mutual exclusive rule buckets by $L(\cdot)$.

In this paper, we use $S(L(v), G')$ to denote the bucket for the binary rules having the source sub-sequence $L(v)$. For example, $S(\text{"}NP \text{会"}, G')$ denotes the bucket for the binary rules having the source-side "NP 会". For simplicity, we also use $S(v, G')$ to denote $S(L(v), G')$.

3.2 Cost Reduction for SCFG Binarization

Given a binary SCFG G', it can be easily noticed that if a rule v in the bucket $S(v, G')$ can be applied to generate one or more new edges in SCFG parsing, any other rules in this bucket can also be applied because all of them can be reduced from the same underlying derivation $L(v)$.

Each application of other rules in the bucket $S(v, G')$ can generate competing edges with the one based on v. Intuitively, the size of bucket can be used to approximately indicate the actual number of competing edges on average, and reducing the size of bucket could help reduce the edges generated in a parsing chart by applying the rules in the bucket. Therefore, if we can find a method to greedily reduce the size of each bucket $S(v, G')$, we can reduce the overall expected edge competitions when parsing with G'.

However, it can be easily proved that the numbers of binary rules in any $G' \in \mathcal{B}(G)$ are same, which implies that we cannot reduce the sizes of all buckets at the same time – removing a rule from one bucket means adding it to another. Allowing for this fact, the excess edge competition example shown in Section 1 is essentially caused by the uneven distribution of rules among different buckets $S(\cdot)$. Accordingly, our optimization objective should be a more even distribution of rules among buckets.

In the following, we formally define a metric to model the evenness of rule distribution over buckets. Given a binary SCFG G' and a binary SCFG rule $v \in G'$, $Q(v)$ is defined as the *cost function* that maps v to the size of the bucket $S(v, G')$:

$$Q(v) = |S(v, G')| \qquad (1)$$

Obviously, all the binary rules in $S(v, G')$ share a common cost value $|S(v, G')|$. For example (Figure 3), both v_{13} and v_{22} are put into the same bucket $S(\text{"}NP\text{ 会"}, G')$, so $Q(v_{13}) = Q(v_{22}) = 2$.

The cost of the SCFG G' is computed by summing up all the costs of SCFG rules in it:

$$Q(G') = \sum_{v \in G'} Q(v) \qquad (2)$$

Back to our task, we are to find an equivalent binary SCFG G' of G with the lowest cost in terms of the cost function $Q(.)$ given in Equation (2):

$$G^* = \text{argmin}_{G' \in \mathcal{B}(G)} Q(G') \qquad (3)$$

Next we will show how G^* is related to the evenness of rule distribution among different buckets. Let $S(G') = \{S_1, \ldots, S_M\}$ be the set of rule buckets containing rules in G', then the value of $Q(G')$ can also be written as:

$$Q(G') = \sum_{1 \le i \le M} |S_i|^2 \qquad (4)$$

Assume $Y_i = |S_i|$ is an empirical distribution of a discrete random variable Y, then the square deviation of the empirical distribution is:

$$\sigma^2 = \frac{1}{M} \sum_i (|S_i| - \bar{Y})^2 \qquad (5)$$

Noticing that $\Sigma|S_i| = |G'|$ and $\bar{Y} = |G'|/M$, Equation (5) can be written as:

$$\sigma^2 = \frac{1}{M}\left(Q(G') - \frac{|G'|^2}{M}\right) \qquad (6)$$

Since both M and $|G'|$ are constants, minimizing the cost function $Q(G')$ is equivalent to minimizing the square deviation of the distribution of rules among different buckets. A binary SCFG with the lower cost indicates the rules are more evenly distributed in terms of derivation patterns on the source language side.

3.3 Static Cost Reduction

Before moving on discussing the algorithm which can optimize Equation (3) based on rule costs specified in Equation (1), we first present an algorithm to find the optimal solution to Equation (3) if we have known the cost setting of G^* and can use the costs as static values during binarization. Using this simplification, the problem of finding the binary SCFG G^* with minimal costs can be reduced to find the optimal binarization $\mathcal{B}^*(R_i)$ for each rule R_i in G.

To obtain $\mathcal{B}^*(R_i)$, we can employ a CKY-style binarization algorithm which builds a compact binarization forest for the rule R_i in bottom-up direction. The algorithm combines two adjacent spans of α_i each time, in which two spans can be combined if and only if they observe the BTG constraints – their translations are either sequentially or reversely adjacent in β_i, the target-side derivation of R_i. The key idea of this algorithm is that we only use the binarization tree with the lowest cost of each span for later combination, which can avoid enumerating all the possible binarization trees of R_i using dynamic programming.

Let α_p^q be the sub-sequence spanning from p to q on the source-side, $v[p, q]$ be optimal binarization tree spanning α_p^q, $Q_v[p, q]$ be the cost of $v[p, q]$, and $Q_r[p, q]$ be the cost of any binary rules whose source-side is α_p^q, then the cost of optimal binarization tree spanning α_p^q can be computed as:

$$Q_v[p, q] = \min_{p \le k \le q-1} (Q_r[p, q] + Q_v[p, k] + Q_v[k + 1, q])$$

The algorithm is shown as follows:

CYK-based binarization algorithm

```
    Input: a SCFG rule $R_i$ and the cost function $Q(.)$.
    Output: the lowest cost binarization on $R_i$
1:  Function CkyBinarization($R_i$, $Q$)
2:     for $l$ = 2 to $n$ do                      ▻ Length of span
3:        for $p$ = 1 to $n - l + 1$ do           ▻ Start of span
4:           $q = p + l$                          ▻ End of span
5:           for $k = p$ to $q - 1$ do            ▻ Partition of span
6:              if not Consecutive($T(p,k)$, $T(k+1,q)$)
                   then next loop
7:              $Q_r[p,q] \leftarrow Q(\alpha_p^q)$
8:              $curCost \leftarrow Q_r[p,q]$+$Q_v[p,k]$+$Q_v[k+1,q]$
9:              if $curCost < minCost$ then
10:                $minCost \leftarrow curCost$
11:                $v[p,q] \leftarrow$ Combine($v[p,k]$, $v[k+1,q]$)
12:          $Q_v[p,q] \leftarrow minCost$
13:    return $v[1,n]$
14: Function Consecutive(( $a$, $b$), ($c$, $d$))
15:    return ($b = c - 1$) or ($d = a - 1$)
```

where n is the number of tokens (consecutive terminals are viewed as a single token) on the source-side of R_i. COMBINE($v[p,k]$, $v[k+1,q]$) combines the two binary sub-trees into a larger sub-tree over α_p^q. $T(p,q) = (a,b)$ means that the non-terminals covering α_p^q have the consecutive indices ranging from a to b on the target-side. If the target non-terminal indices are not consecutive, we set $T(p,q) = (-1,-1)$. $Q(\alpha_p^q) = Q(v')$ where v' is any rule in the bucket $S(\alpha_p^q, G')$.

In the algorithm, lines 9-11 implement dynamic programming, and the function CONSECUTIVE checks whether the two spans can be combined.

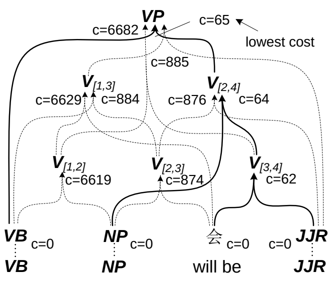

Figure 4: Binarization forest for an SCFG rule

$L(v)$	$Q(v)$	$L(v)$	$Q(v)$
VB NP	6619	VB NP 会	10
NP 会	874	NP 会 JJR	2
会 JJR	62	VB NP 会 JJR	1

Table 1: Sub-sequences and corresponding costs

Figure 4 shows an example of the compact forest the algorithm builds, where the solid lines indicate the optimal binarization of the rule, while other alternatives pruned by dynamic programming are shown in dashed lines. The costs for binarization trees are computed based on the cost table given in Table 1.

The time complexity of the CKY-based binarization algorithm is $\Theta(n^3)$, which is higher than that of the linear binarization such as the synchronous binarization (Zhang et al., 2006). But it is still efficient enough in practice, as there are generally only a few tokens ($n < 5$) on the source-sides of SCFG rules. In our experiments, the linear binarization method is just 2 times faster than the CKY-based binarization.

3.4 Iterative Cost Reduction

However, $Q(\cdot)$ cannot be easily predetermined in a static way as is assumed in Section 3.3 because it depends on G' and should be updated whenever a rule in G is binarized differently. In our work this problem is solved using the *iterative cost reduction* algorithm, in which the update of G' and the cost function $Q(\cdot)$ are coupled together.

Iterative cost reduction algorithm

```
    Input: An SCFG $G$
    Output: An equivalent binary SCFG $G'$ of $G$
1:  Function IterativeCostReduction($G$)
2:    $G' \leftarrow G_0$
3:    for each $v \in G_0$ do
4:       $Q(v) = |S(v, G_0)|$
5:    while $Q(G')$ does not converge do
6:       for each $R_i \in G$ do
7:          $G_{[-R_i]} \leftarrow G' - \mathcal{B}(R_i)$
8:          for each $v \in \mathcal{B}(R_i)$ do
9:             for each $v' \in S(v, G')$ do
10:               $Q(v') \leftarrow Q(v') - 1$
11:         $\mathcal{B}(R_i) \leftarrow$ CkyBinarization($R_i$, $Q$)
12:         $G' \leftarrow G_{[-R_i]} \cup \mathcal{B}(R_i)$
13:         for each $v \in \mathcal{B}(R_i)$ do
14:            for each $v' \in S(v, G')$ do
15:               $Q(v') \leftarrow Q(v') + 1$
16: return $G'$
```

In the *iterative cost reduction* algorithm, we first obtain an initial binary SCFG G_0 using the synchronous binarization method proposed in (Zhang et al., 2006). Then G_0 is assigned to an iterative variable G'. The cost of each binary rule in G_0 is computed based on G_0 according to Equation (1) (lines 3-4 in the algorithm).

After initialization, G' is updated by iteratively finding better binarization for each rule in G. The basic idea is: for each R_i in G, we remove the current binarization result for R_i from G' (line 7), while the cost function $Q(\cdot)$ is updated accordingly since the removal of binary rule $v \in \mathcal{B}(R_i)$ results in the reduction of the size of the corresponding bucket $S(v, G')$. Lines 8-10 implement the cost reduction of each binary rule in the bucket $S(v, G')$.

Next, we find the lowest cost binarization for R_i based on the updated cost function $Q(\cdot)$ with the CKY-based binarization algorithm presented in Section 3.3 (line 11).

At last, the new binarization for R_i is added back to G' and $Q(\cdot)$ is re-updated to synchronize with this change (lines 12-15). Figure 5 illustrates the differences between the static cost reduction and the iterative cost reduction.

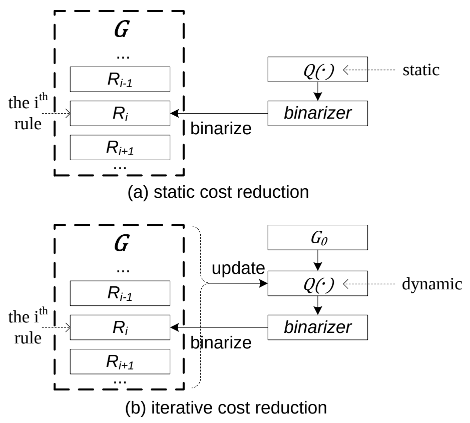

Figure 5: Comparison between the static cost reduction and the iterative cost reduction

The algorithm stops when $Q(G')$ does not decrease any more. Next we will show that $Q(G')$ is guaranteed not to increase in the iterative process.

For any $\mathcal{B}(R_i)$ on R_i, we have

$$Q\left(G_{[-R_i]} \cup \mathcal{B}(R_i)\right) = 2 \times Q\big(\mathcal{B}(R_i)\big) + |\mathcal{B}(R_i)| + Q\big(G_{[-R_i]}\big)$$

As both $|\mathcal{B}(R_i)|$ and $Q\big(G_{[-R_i]}\big)$ are constants with respect to $Q(\mathcal{B}(R_i))$, $Q\left(G_{[-R_i]} \cup \mathcal{B}(R_i)\right)$ is a linear function of $Q(\mathcal{B}(R_i))$, and the corresponding slope is positive. Thus $Q\left(G_{[-R_i]} \cup \mathcal{B}(R_i)\right)$ reaches the lowest value only when $Q(\mathcal{B}(R_i))$ reaches the lowest value. So $Q\left(G_{[-R_i]} \cup \mathcal{B}(R_i)\right)$ achieves the lowest cost when we replace the current binarization with the new binarization $\mathcal{B}^*(R_i)$ (line 12). Therefore $Q\left(G_{[-R_i]} \cup \mathcal{B}(R_i)\right)$ does not increase in the processing on each R_i (lines 7-15), and $Q(G')$ will finally converge to a local minimum when the algorithm stops.

4 Experiments

The experiments are conducted on Chinese-to-English translation in a state-of-the-art string-to-tree SMT system. All the results are reported in terms of case-insensitive BLEU4(%).

4.1 Experimental Setup

Our bilingual training corpus consists of about 350K bilingual sentences (9M Chinese words + 10M English words)[2]. Giza++ is employed to perform word alignment on the bilingual sentences. The parse trees on the English side are generated using the Berkeley Parser[3]. A 5-gram language model is trained on the English part of LDC bilingual training data and the Xinhua part of Gigaword corpus. Our development data set comes from NIST2003 evaluation data in which the sentences of more than 20 words are excluded to speed up the Minimum Error Rate Training (MERT). The test data sets are the NIST evaluation sets of 2005 and 2008.

Our string-to-tree SMT system is built based on the work of (Galley et al., 2006; Marcu et al., 2006), where both the minimal GHKM and SPMT rules are extracted from the training corpus, and the composed rules are generated by combining two or three minimal GHKM and SPMT rules. Before the rule extraction, we also binarize the parse trees on the English side using Wang et al. (2007) ‘s method to increase the coverage of GHKM and SPMT rules. There are totally 4.26M rules after the low frequency rules are filtered out. The pruning strategy is similar to the cube pruning described in (Chiang, 2007). To achieve acceptable translation speed, the beam size is set to 50 by default. The baseline system is based on the synchronous binarization (Zhang et al., 2006).

4.2 Binarization Schemes

Besides the baseline (Zhang et al., 2006) and iterative cost reduction binarization methods, we also perform right-heavy and random synchronous binarizations for comparison. In this paper, the random synchronous binarization is obtained by: 1) performing the CKY binarization to build the binarization forest for an SCFG rule; then 2) performing a top-down traversal of the forest. In the traversal, we randomly pick a feasible binarization for each span, and then go on the traversal in the two branches of the picked binarization.

Table 2 shows the costs of resulting binary SCFGs generated using different binarization methods. The costs of the baseline (left-heavy)

[2] LDC2003E14, LDC2003E07, LDC2005T06 and LDC2005T10

[3] http://code.google.com/p/berkeleyparser/

and right-heavy binarization are similar, while the cost of the random synchronous binarization is lower than that of the baseline method[4]. As expected, the iterative cost reduction method obtains the lowest cost, which is much lower than that of the other three methods.

Method	cost of binary SCFG G'
Baseline	4,897M
Right-heavy	5,182M
Random	3,479M
Iterative cost reduction	185M

Table 2: Costs of the binary SCFGs generated using different binarization methods.

4.3 Evaluation of Translations

Table 3 shows the performance of SMT systems based on different binarization methods. The iterative cost reduction binarization method achieves the best performance on the test sets as well as the development set. Compared with the baseline method, it obtains gains of 0.82 and 0.84 BLEU scores on NIST05 and NIST08 test sets respectively. Using the statistical significance test described by Koehn (2004), the improvements are significant ($p < 0.05$).

Method	Dev	NIST05	NIST08
Baseline	40.02	37.90	27.53
Right-heavy	40.05	37.87	27.40
Random	40.10	37.99	27.58
Iterative cost reduction	40.97*	38.72*	28.37*

Table 3: Performance (BLUE4(%)) of different binarization methods. * = significantly better than baseline ($p < 0.05$).

The baseline method and the right-heavy binarization method achieve similar performance, while the random synchronous binarization method performs slightly better than the baseline method, which agrees with the fact of the cost reduction shown in Table 2. A possible reason that the random synchronous binarization method can outperform the baseline method lies in that compared with binarizing SCFG in a fixed way, the random synchronous binarization tends to give a more even distribution of rules among buckets, which alleviates the problem of edge competition. However, since the high-frequency source sub-sequences still have high probabilities to be generated in the binarization and lead to the excess competing edges, it just achieves a very small improvement.

[4] We perform random synchronous binarization for 5 times and report the average cost.

4.4 Translation Accuracy vs. Cost of Binary SCFG

We also study the impacts of cost reduction on translation accuracy over iterations in iterative cost reduction. Figure 6 and Figure 7 show the results on NIST05 and NIST08 test sets. We can see that the cost of the resulting binary SCFG drops greatly as the iteration count increases, especially in the first iteration, and the BLEU scores increase as the cost decreases.

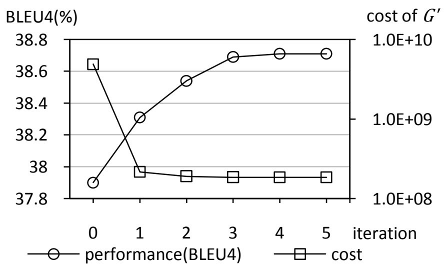

Figure 6: Cost of binary SCFG vs. BLEU4 (NIST05)

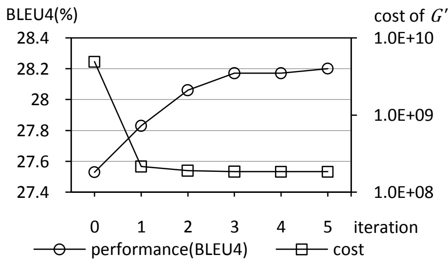

Figure 7: Cost of binary SCFG vs. BLEU4 (NIST08)

4.5 Impact of Beam Size

In this section, we study the impacts of beam sizes on translation accuracy as well as competing edges. To explicitly investigate the issue under large beam sizes, we use a subset of NIST05 and NIST08 test sets for test, which has 50 Chinese sentences of no longer than 10 words.

Figure 8 shows that the iterative cost reduction method is consistently better than the baseline method under various beam settings. Besides the experiment on the test set of short sentences, we also conduct the experiment on NIST05 test set. To achieve acceptable decoding speed, we range the beam size from 10 to 70. As shown in Figure 9, the iterative cost reduction method also outperforms the baseline method under various beam settings on the large test set.

Though enlarging beam size can reduce the search errors and improve the system performance, the decoding speed of string-to-tree SMT drops dramatically when we enlarge the beam size. The problem is more serious when long sentences are translated. For example, when the beam size is set to a larger number (e.g. 200), our decoder takes nearly one hour to translate a sentence whose length is about 20 on a 3GHz CPU. Decoding on the entire NIST05 and NIST08 test sets with large beam sizes is impractical.

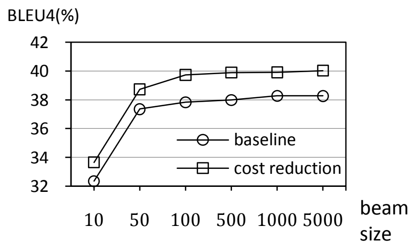

Figure 8: BLEU4 against beam size (small test set)

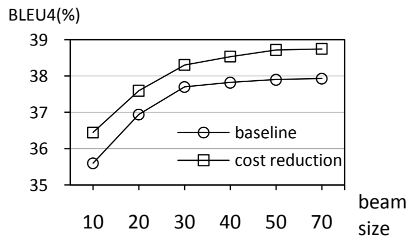

Figure 9: BLEU4 against beam size (NIST05)

Figure 10 compares the baseline method and the iterative cost reduction method in terms of translation accuracy against the number of edges proposed during decoding. Actually, the number of edges proposed during decoding can be regarded as a measure of the size of search space. We can see that the iterative cost reduction method outperforms the baseline method under various search effort.

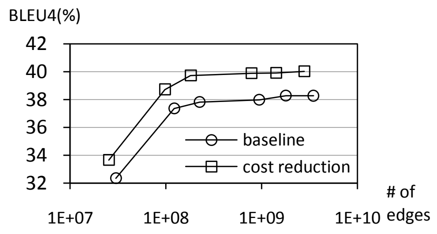

Figure 10: BLEU4 against competing edges

The experimental results of this section show that compared with the baseline method, the iterative cost reduction method can lead to much fewer edges (about 25% reduction) as well as the higher BLEU scores under various beam settings.

4.6 Edge Competition vs. Cost of Binary SCFG

In this section, we study the impacts of cost reduction on the edge competition in the chart cells of our CKY-based decoder. Two metrics are used to evaluate the degree of edge competition. They are the variance and the mean of the number of competing edges in the chart cells, where high variance means that in some chart cells the rules have high risk to be pruned due to the large number of competing edges. The same situation holds for the mean as well. Both of the two metrics are calculated on NIST05 test set, varying with the span length of chart cell.

Figure 11 shows the cost of resulting binary SCFG and the variance of competing edges against iteration count in iterative cost reduction. We can see that both the cost and the variance reduce greatly as the iteration count increases. Figure 12 shows the case for mean, where the reduction of cost also leads to the reduction of the mean value. The results shown in Figure 11 and Figure 12 indicate that the cost reduction is helpful to reduce edge competition in the chart cells.

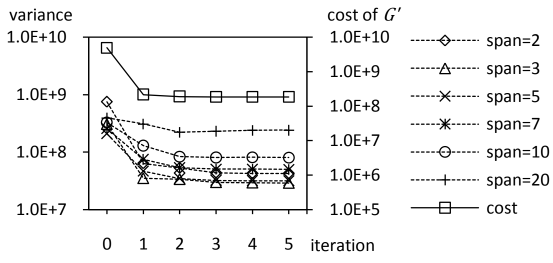

Figure 11: Cost of binary SCFG vs. variance of competing edge number (NIST05)

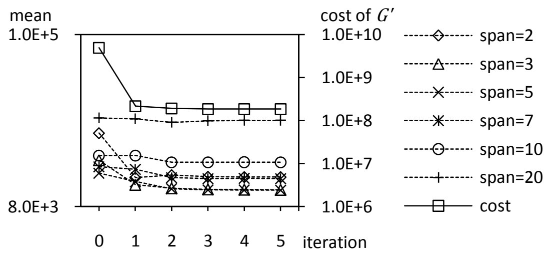

Figure 12: Cost of binary SCFG vs. mean of competing edge number (NIST05)

We also perform decoding without pruning (i.e. beam size = ∞) on a very small set which has 20 sentences of no longer than 7 words. In this experiment, the baseline system and our iterative cost reduction based system propose 14,454M and 10,846M competing edges respectively. These numbers can be seen as the real numbers of the edges proposed during decoding instead of an approximate number observed in the pruned search space. It suggests that our method can reduce the number of the edges in real search space effectively. A possible reason to this result is that the cost reduction based binarization could reduce the probability of rule mismatching caused by binarization, which results in the reduction of the number of edges proposed during decoding.

5 Conclusion and Future Work

This paper introduces a new binarization method, aiming at choosing better binarization for SCFG-based SMT systems. We demonstrate the effectiveness of our method on a state-of-the-art string-to-tree SMT system. Experimental results show that our method can significantly outperform the conventional synchronous binarization method, which indicates that better binarization selection is very beneficial to SCFG-based SMT systems.

In this paper the cost of a binary rule is defined based on the competition among the binary rules that have the same source-sides. However, some binary rules with different source-sides may also have competitions in a chart cell. We think that the cost of a binary rule can be better estimated by taking the rules with different source-sides into account. We intend to study this issue in our future work.

Acknowledgements

The authors would like to thank the anonymous reviewers for their pertinent comments, and Xinying Song, Nan Duan and Shasha Li for their valuable suggestions for improving this paper.

References

Eugene Charniak, Mark Johnson, Micha Elsner, Joseph Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R. Shrivaths, Jeremy Moore, Michael Pozar, and Theresa Vu. 2006. Multilevel Coarse-to-Fine PCFG Parsing. In *Proc. of HLT-NAACL* 2006, New York, USA, 168-175.

Eugene Charniak, Sharon Goldwater, and Mark Johnson. 1998. Edge-Based Best-First Chart Parsing. In *Proc. of the Six Workshop on Very Large Corpora*, pages: 127-133.

David Chiang. 2005. A Hierarchical Phrase-Based Model for Statistical Machine Translation. In *Proc. of ACL 2005*, Ann Arbor, Michigan, pages: 263-270.

David Chiang. 2007. Hierarchical Phrase-based Translation. *Computational Linguistics*. 33(2): 202-208.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable Inference and Training of Context-Rich Syntactic Translation Models. In *Proc. of ACL 2006*, Sydney, Australia, pages: 961-968.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proc. of HLT-NAACL 2004*, Boston, USA, pages: 273-280.

Liang Huang. 2007. Binarization, Synchronous Binarization, and Target-side binarization. In *Proc. of HLT-NAACL 2007 / AMTA workshop on Syntax and Structure in Statistical Translation*, New York, USA, pages: 33-40.

Tadao Kasami. 1965. An Efficient Recognition and Syntax Analysis Algorithm for Context-Free Languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts.

Philipp Koehn. 2004. Statistical Significance Tests for Machine Translation Evaluation. In *Proc. of EMNLP 2004*, Barcelona, Spain , pages: 388–395.

Daniel Marcu, Wei Wang, Abdessamad Echihabi, and Kevin Knight. 2006. SPMT: Statistical machine translation with syntactified target language phrases. In *Proc. of EMNLP 2006*, Sydney, Australia, pages: 44-52.

Giorgio Satta and Enoch Peserico. 2005. Some Computational Complexity Results for Synchronous Context-Free Grammars. In *Proc. of HLT-EMNLP 2005*, Vancouver, pages: 803-810.

L. Shapiro and A. B. Stephens. 1991. Bootstrap percolation, the Sch$\ddot{r}$oder numbers, and the *n*-kings problem. SIAM Journal on Discrete Mathematics, 4(2):275-280.

Xinying Song, Shilin Ding and Chin-Yew Lin. 2008. Better Binarization for the CKY Parsing. In *Proc. of EMNLP 2008*, Hawaii, pages: 167-176.

Yoshimasa Tsuruoka and Junichi Tsujii. 2004. Iterative CKY Parsing for Probabilistic Context-Free Grammars. In *Proc. of IJCNLP 2004*, pages: 52-60.

Wei Wang and Kevin Knight and Daniel Marcu. 2007. Binarizing Syntax Trees to Improve Syntax-Based Machine Translation Accuracy. In *Proc. of EMNLP-CoNLL 2007*, Prague, Czech Republic, pages: 746-754.

D. H. Younger. 1967. Recognition and Parsing of Context-Free Languages in Time n^3. *Information and Control*, 10(2):189-208.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous Binarization for Machine Translation. In *Proc. of HLT-NAACL 2006*, New York, USA, pages: 256- 263.