Binarizing Syntax Trees to Improve Syntax-Based Machine Translation Accuracy

Wei Wang and **Kevin Knight** and **Daniel Marcu**
Language Weaver, Inc.
4640 Admiralty Way, Suite 1210
Marina del Rey, CA, 90292
{wwang,kknight,dmarcu}@languageweaver.com

Abstract

We show that phrase structures in Penn Treebank style parses are not optimal for syntax-based machine translation. We exploit a series of binarization methods to restructure the Penn Treebank style trees such that syntactified phrases smaller than Penn Treebank constituents can be acquired and exploited in translation. We find that by employing the EM algorithm for determining the binarization of a parse tree among a set of alternative binarizations gives us the best translation result.

1 Introduction

Syntax-based translation models (Eisner, 2003; Galley et al., 2006; Marcu et al., 2006) are usually built directly from Penn Treebank (PTB) (Marcus et al., 1993) style parse trees by composing treebank grammar rules. As a result, often no substructures corresponding to partial PTB constituents are extracted to form translation rules.

Syntax translation models acquired by composing treebank grammar rules assume that long rewrites are not decomposable into smaller steps. This effectively restricts the generalization power of the induced model. For example, suppose we have an xRs (Knight and Graehl, 2004) rule R_1 in Figure 1 that translates the Chinese phrase RUSSIA MINISTER VIKTOR-CHERNOMYRDIN into an English NPB tree fragment yielding an English phrase. Also suppose that we want to translate a Chinese phrase

VIKTOR-CHERNOMYRDIN AND HIS COLLEAGUE

into English. What we desire is that if we have another rule R_2 as shown in Figure 1, we could somehow compose it with R_1 to obtain the desirable translation. We unfortunately cannot do this because R_1 and R_2 are not further decomposable and their substructures cannot be re-used. The requirement that all translation rules have exactly one root node does not enable us to use the translation of VIKTOR-CHERNOMYRDIN in any other contexts than those seen in the training corpus.

A solution to overcome this problem is to right-binarize the left-hand side (LHS) (or the English-side) tree of R_1 such that we can decompose R_1 into R_3 and R_4 by factoring `NNP(viktor) NNP(chernomyrdin)` out as R_4 according to the word alignments; and left-binarize the LHS of R_2 by introducing a new tree node that collapses the two NNP's, so as to generalize this rule, getting rule R_5 and rule R_6. We also need to consistently syntactify the root labels of R_4 and the new frontier label of R_6 such that these two rules can be composed. Since labeling is not a concern of this paper, we simply label new nodes with X-bar where X here is the parent label. With all these in place, we now can translate the foreign sentence by composing R_6 and R_4 in Figure 1.

Binarizing the syntax trees for syntax-based machine translation is similar in spirit to generalizing parsing models via markovization (Collins, 1997; Charniak, 2000). But in translation modeling, it is unclear how to effectively markovize the translation rules, especially when the rules are complex like those proposed by Galley et al. (2006).

In this paper, we explore the generalization ability of simple binarization methods like left-, right-, and head-binarization, and also their combinations. Simple binarization methods binarize syntax trees in a consistent fashion (left-, right-, or head-) and

Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 746–754, Prague, June 2007.

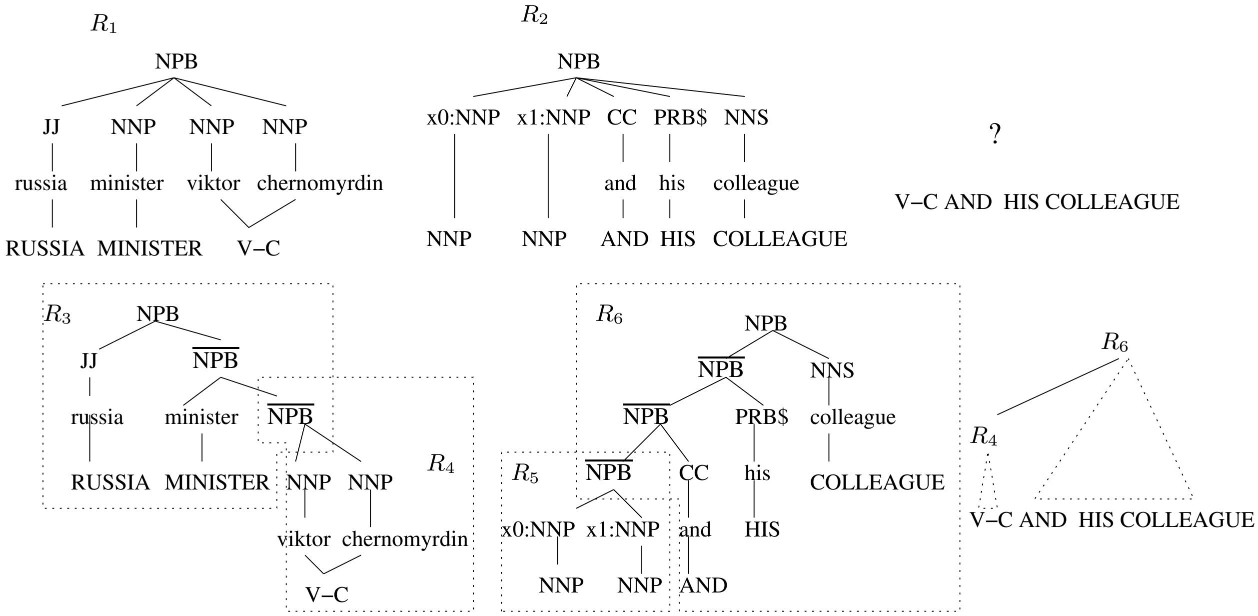

Figure 1: Generalizing translation rules by binarizing trees.

thus cannot guarantee that all the substructures can be factored out. For example, right binarization on the LHS of R_1 makes available R_4, but misses R_6 on R_2. We then introduce a *parallel restructuring* method, that is, one can binarize both to the left and right at the same time, resulting in a binarization forest. We employ the EM (Dempster et al., 1977) algorithm to learn the binarization bias for each tree node from the parallel alternatives. The EM-binarization yields best translation performance.

The rest of the paper is organized as follows. Section 2 describes related research. Section 3 defines the concepts necessary for describing the binarizations methods. Section 4 describes the tree binarization methods in details. Section 5 describes the forest-based rule extraction algorithm, and section 6 explains how we restructure the trees using the EM algorithm. The last two sections are for experiments and conclusions.

2 Related Research

Several researchers (Melamed et al., 2004; Zhang et al., 2006) have already proposed methods for binarizing synchronous grammars in the context of machine translation. Grammar binarization usually maintains an equivalence to the original grammar such that binarized grammars generate the same language and assign the same probability to each string as the original grammar does. Grammar binarization is often employed to make the grammar fit in a CKY parser. In our work, we are focused on binarization of parse trees. Tree binarization generalizes the resulting grammar and changes its probability distribution. In tree binarization, synchronous grammars built from restructured (binarized) training trees still contain non-binary, multi-level rules and thus still require the binarization transformation so as to be employed by a CKY parser.

The translation model we are using in this paper belongs to the xRs formalism (Knight and Graehl, 2004), which has been proved successful for machine translation in (Galley et al., 2004; Galley et al., 2006; Marcu et al., 2006).

3 Concepts

We focus on tree-to-string (in noisy-channel model sense) translation models. Translation models of this type are typically trained on tuples of a source-language sentence **f**, a target language (e.g., English) parse tree π that yields $\mathbf{e}$ and translates from **f**, and the word alignments **a** between **e** and **f**. Such a tuple is called an alignment graph in (Galley et al., 2004). The graph (1) in Figure 2 is such an alignment graph.

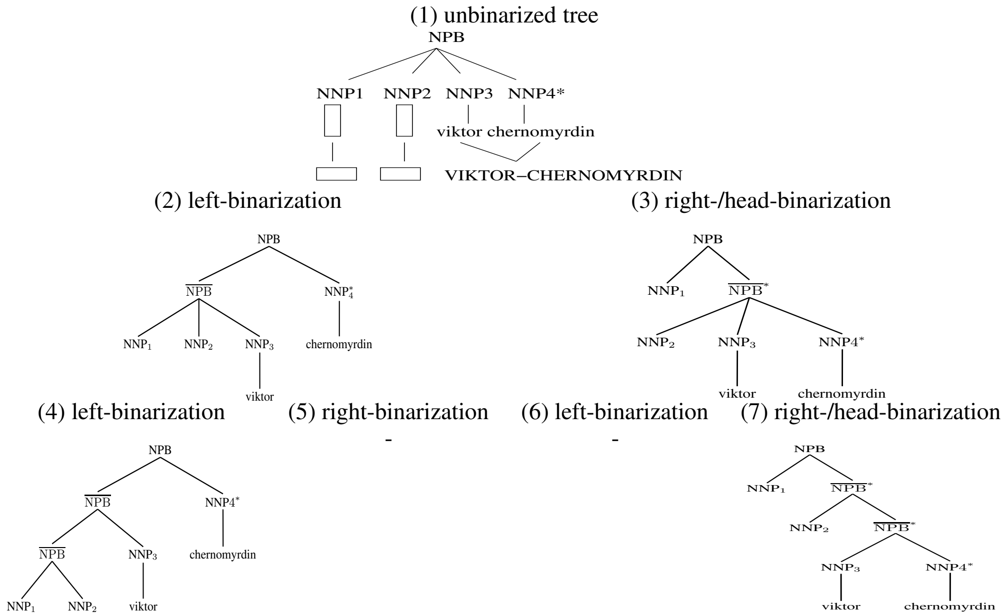

Figure 2: Left, right, and head binarizations. Heads are marked with *'s. New nonterminals introduced by binarization are denoted by X-bars.

A tree node in π is *admissible* if the **f** string covered by the node is contiguous but not empty, and if the **f** string does not align to any **e** string that is not covered by π. An xRs rule can be extracted only from an admissible tree node, so that we do not have to deal with dis-contiguous **f** spans in decoding (or synchronous parsing). For example, in tree (2) in Figure 2, node $\overline{\text{NPB}}$ is not admissible because the **f** string that the node covers also aligns to `NNP4`, which is not covered by the $\overline{\text{NPB}}$. Node $\overline{\text{NPB}}$ in tree (3), on the other hand, is admissible.

A set of sibling tree nodes is called *factorizable* if we can form an admissible new node dominating them. For example, in tree (1) of Figure 2, sibling nodes NNP_2 NNP_3 and NNP_4 are factorizable because we can factorize them out and form a new node $\overline{\text{NPB}}$, resulting in tree (3). Sibling tree nodes NNP_1 NNP_2 and NNP_3 are not factorizable. In synchronous parse trees, not all sibling nodes are factorizable, thus not all sub-phrases can be acquired and syntactified. The main purpose of our paper is to restructure parse trees by factorization such that syntactified sub-phrases can be employed in translation.

4 Binarizing Syntax Trees

We are going to binarize a tree node n that dominates r children n_1, ..., n_r. Restructuring will be performed by introducing new tree nodes to dominate a subset of the children nodes. To avoid overgeneralization, we allow ourselves to form only one new node at a time. For example, in Figure 2, we can binarize tree (1) into tree (2), but we are not allowed to form two new nodes, one dominating NNP_1 NNP_2 and the other dominating NNP_3 NNP_4. Since labeling is not the concern of this paper, we relabel the newly formed nodes as $\overline{n}$.

4.1 Simple binarization methods

The *left binarization* of node n (i.e., the NPB in tree (1) of Figure 2) factorizes the leftmost $r - 1$ children by forming a new node $\overline{n}$ (i.e., $\overline{\text{NPB}}$ in tree (2)) to dominate them, leaving the last child n_r untouched; and then makes the new node $\overline{n}$ the left child of n. The method then recursively left-binarizes the newly formed node $\overline{n}$ until two leaves are reached. In Figure 2, we left-binarize tree (1) into (2) and then into (4).

The *right binarization* of node n factorizes the rightmost $r - 1$ children by forming a new node $\overline{n}$ (i.e., $\overline{\text{NPB}}$ in tree (3)) to dominate them, leaving the first child n_1 untouched; and then makes the new node $\overline{n}$ the right child of n. The method then recursively right-binarizes the newly formed node $\overline{n}$. In Figure 2, we right-binarize tree (1) into (3) and then into (7).

The *head binarization* of node n left-binarizes n if the head is the first child; otherwise, right-binarizes n. We prefer right-binarization to left-binarization when both are applicable under the head restriction because our initial motivation was to generalize the `NPB`-rooted translation rules. As we will show in the experiments, binarization of other types of phrases contribute to the translation accuracy improvement as well.

Any of these simple binarization methods is easy to implement, but is incapable of giving us all the factorizable sub-phrases. Binarizing all the way to the left, for example, from tree (1) to tree (2) and to tree (4) in Figure 2, does not enable us to acquire a substructure that yields NNP_3 NNP_4 and their translational equivalences. To obtain more factorizable sub-phrases, we need to *parallel-binarize* in both directions.

4.2 Parallel binarization

Simple binarizations transform a parse tree into another single parse tree. Parallel binarization will transform a parse tree into a binarization forest, desirably packed to enable dynamic programming when extracting translation rules from it.

Borrowing terms from parsing semirings (Goodman, 1999), a packed forest is composed of additive forest nodes ($\oplus$-nodes) and multiplicative forest nodes ($\otimes$-nodes). In the binarization forest, a $\otimes$-node corresponds to a tree node in the unbinarized tree; and this $\otimes$-node composes several $\oplus$-nodes, forming a one-level substructure that is observed in the unbinarized tree. A $\oplus$-node corresponds to alternative ways of binarizing the same tree node in the unbinarized tree and it contains one or more $\otimes$-nodes. The same $\oplus$-node can appear in more than one place in the packed forest, enabling sharing. Figure 3 shows a packed forest obtained by packing trees (4) and (7) in Figure 2 via the following parallel binarization algorithm.

To parallel-binarize a tree node n that has children $n_1, ..., n_r$, we employ the following steps:

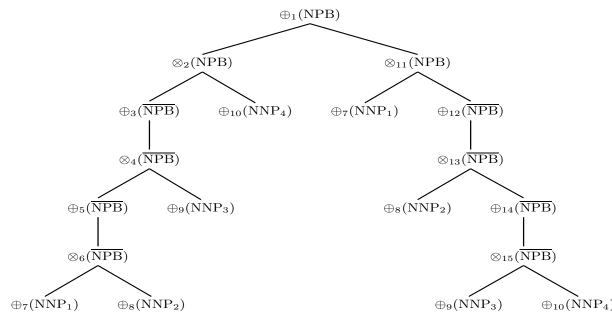

Figure 3: Packed forest obtained by packing trees (4) and (7) in Figure 2

- We recursively parallel-binarize children nodes n_1, ..., n_r, producing binarization $\oplus$-nodes $\oplus(n_1)$, ..., $\oplus(n_r)$, respectively.
- We right-binarize n, if any contiguous[1] subset of children $n_2, ..., n_r$ is factorizable, by introducing an intermediate tree node labeled as $\overline{n}$. We recursively parallel-binarize $\overline{n}$ to generate a binarization forest node $\oplus(\overline{n})$. We form a multiplicative forest node $\otimes_R$ as the parent of $\oplus(n_1)$ and $\oplus(\overline{n})$.
- We left-binarize n if any contiguous subset of $n_1, ..., n_{r-1}$ is factorizable and if this subset contains n_1. Similar to the above right-binarization, we introduce an intermediate tree node labeled as $\overline{n}$, recursively parallel-binarize $\overline{n}$ to generate a binarization forest node $\oplus(\overline{n})$, form a multiplicative forest node $\otimes_L$ as the parent of $\oplus(\overline{n})$ and $\oplus(n_1)$.
- We form an additive node $\oplus(n)$ as the parent of the two already formed multiplicative nodes $\otimes_L$ and $\otimes_R$.

The (left and right) binarization conditions consider any subset to enable the factorization of small constituents. For example, in tree (1) of Figure 2, although NNP_1 NNP_2 NNP_3 of NPB are not factorizable, the subset NNP_1 NNP_2 is factorizable. The binarization from tree (1) to tree (2) serves as a relaying step for us to factorize NNP_1 NNP_2 in tree (4). The left-binarization condition is stricter than

[1]We factorize only subsets that cover contiguous spans to avoid introducing dis-contiguous constituents for practical purpose. In principle, the algorithm works fine without this binarization condition.

the right-binarization condition to avoid spurious binarization; i.e., to avoid the same subconstituent being reached via both binarizations. We could transform tree (1) directly into tree (4) without bothering to generate tree (3). However, skipping tree (3) will create us difficulty in applying the EM algorithm to choose a better binarization for each tree node, since tree (4) can neither be classified as left binarization nor as right binarization of the original tree (1) — it is the result of the composition of two left-binarizations.

In parallel binarization, nodes are not always binarizable in both directions. For example, we do not need to right-binarize tree (2) because NNP_2 NNP_3 are not factorizable, and thus cannot be used to form sub-phrases. It is still possible to right-binarize tree (2) without affecting the correctness of the parallel binarization algorithm, but that will spuriously increase the branching factor of the search for the rule extraction, because we will have to expand more tree nodes.

A restricted version of parallel binarization is the *headed parallel binarization*, where both the left and the right binarization must respect the head propagation property at the same time.

A nice property of parallel binarization is that for any factorizable substructure in the unbinarized tree, we can always find a corresponding admissible $\oplus$-node in the parallel-binarized packed forest. A leftmost substructure like the lowest $\overline{\mathrm{NPB}}$-subtree in tree (4) of Figure 2 can be made factorizable by several successive left binarizations, resulting in $\oplus_5(\overline{\mathrm{NPB}})$-node in the packed forest in Figure 3. A substructure in the middle can be factorized by the composition of several left- and right-binarizations. Therefore, after a tree is parallel-binarized, to make the sub-phrases available to the MT system, all we need to do is to extract rules from the admissible nodes in the packed forest. Rules that can be extracted from the original unrestructured tree can be extracted from the packed forest as well.

Parallel binarization results in parse forests. Thus translation rules need to be extracted from training data consisting of (e-forest, **f**, **a**)-tuples.

5 Extracting translation rules from (e-forest, f, a)-tuples

The algorithm to extract rules from (e-forest, **f**, **a**)-tuples is a natural generalization of the (e-parse, **f**, **a**)-based rule extraction algorithm in (Galley et al., 2006). The input to the forest-based algorithm is a (e-forest, **f**, **a**)-triple. The output of the algorithm is a derivation forest (Galley et al., 2006) composed of xRs rules. The algorithm recursively traverses the e-forest top-down and extracts rules only at admissible forest nodes.

The following procedure transforms the packed e-forest in Figure 3 into a packed synchronous derivation in Figure 4.

Condition 1: Suppose we reach an additive e-forest node, e.g. $\oplus_1(\mathrm{NPB})$ in Figure 3. For each of $\oplus_1(\mathrm{NPB})$'s children, e-forest nodes $\otimes_2(\mathrm{NPB})$ and $\otimes_{11}(\mathrm{NPB})$, we go to condition 2 to recursively extract rules on these two e-forest nodes, generating multiplicative derivation forest nodes, i.e., $\otimes(\mathrm{NPB}(\overline{\mathrm{NPB}} : x_0\ \mathrm{NNP}_3(\mathrm{viktor})\ \mathrm{NNP}_4(\mathrm{chernomyrdin})_4) \rightarrow x_0\ \mathrm{V\text{-}C})$ and $\otimes(\mathrm{NPB}(\mathrm{NNP}_1\ \overline{\mathrm{NPB}}(\mathrm{NNP}_2 : x_0\ \overline{\mathrm{NPB}} : x_1)) \rightarrow x_0\ x_1\ x_2)$ in Figure 4. We make these new $\otimes$ nodes children of $\oplus(\mathrm{NPB})$ in the derivation forest.

Condition 2: Suppose we reach a multiplicative parse forest node, i.e., $\otimes_{11}(\mathrm{NPB})$ in Figure 3. We extract rules rooted at it using the procedure in (Galley et al., 2006), forming multiplicative derivation forest nodes, i.e., $\otimes(\mathrm{NPB}(\mathrm{NNP}_1\ \overline{\mathrm{NPB}}(\mathrm{NNP}_2 : x_0\ \overline{\mathrm{NPB}} : x_1)) \rightarrow x_0\ x_1\ x_2)$ We then go to condition 1 to form the derivation forest on the additive frontier e-forest nodes of the newly extracted rules, generating additive derivation forest nodes, i.e., $\oplus(\mathrm{NNP}_1)$, $\oplus(\mathrm{NNP}_2)$ and $\oplus(\overline{\mathrm{NPB}})$. We make these $\oplus$ nodes the children of node $\otimes(\mathrm{NPB}(\mathrm{NNP}_1\ \overline{\mathrm{NPB}}(\mathrm{NNP}_2 : x_0\ \overline{\mathrm{NPB}} : x_1)) \rightarrow x_0\ x_1\ x_2)$ in the derivation forest.

This algorithm is a natural extension of the extraction algorithm in (Galley et al., 2006) in the sense that we have an extra condition (1) to relay rule extraction on additive e-forest nodes.

It is worthwhile to eliminate the spuriously ambiguous rules that are introduced by the parallel bi-

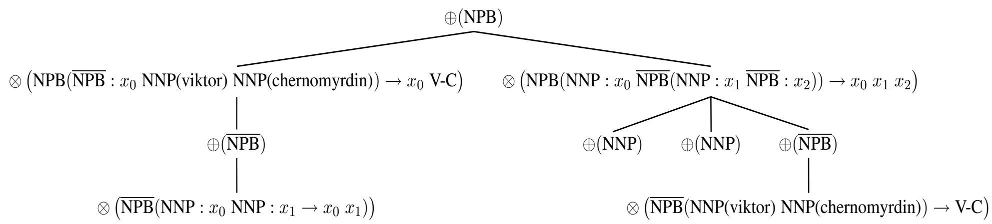

Figure 4: Derivation forest.

narization. For example, we may extract the following two rules:

- $\overline{\mathrm{A}}(\overline{\mathrm{A}}(\mathrm{B}{:}x_0\ \mathrm{C}{:}x_1)\mathrm{D}{:}x_2) \rightarrow x_1\ x_0\ x_2$
- $\overline{\mathrm{A}}(\mathrm{B}{:}x_0\ \overline{\mathrm{A}}(\mathrm{C}{:}x_1\ \mathrm{D}{:}x_2)) \rightarrow x_1\ x_0\ x_2$

These two rules, however, are not really distinct. They both converge to the following rules if we delete the auxiliary nodes $\overline{A}$.

- $\overline{\mathrm{A}}(\mathrm{B}{:}x_0\ C{:}x_1\ D{:}x_2) \rightarrow x_1\ x_0\ x_2$

The forest-base rule extraction algorithm produces much larger grammars than the tree-based one, making it difficult to scale to very large training data. From a 50M-word Chinese-to-English parallel corpus, we can extract more than 300 million translation rules, while the tree-based rule extraction algorithm gives approximately 100 million. However, the restructured trees from the simple binarization methods are not guaranteed to give the best trees for syntax-based machine translation. What we desire is a binarization method that still produces single parse trees, but is able to mix left binarization and right binarization in the same tree. In the following, we shall use the EM algorithm to learn the desirable binarization on the forest of binarization alternatives proposed by the parallel binarization algorithm.

6 Learning how to binarize via the EM algorithm

The basic idea of applying the EM algorithm to choose a restructuring is as follows. We perform a set $\{\beta\}$ of binarization operations on a parse tree τ. Each binarization β is the sequence of binarizations on the necessary (i.e., factorizable) nodes in τ in preorder. Each binarization β results in a restructured tree τ_β. We extract rules from (τ_β, **f**, **a**), generating a translation model consisting of parameters (i.e., rule probabilities) θ. Our aim is to obtain the binarization β^* that gives the best likelihood of the restructured training data consisting of (τ_β, **f** , **a**)-tuples. That is

$$\beta^* \quad = \quad \arg\max_{\beta} p(\tau_\beta, \mathbf{f}, \mathbf{a}|\theta^*) \qquad (1)$$

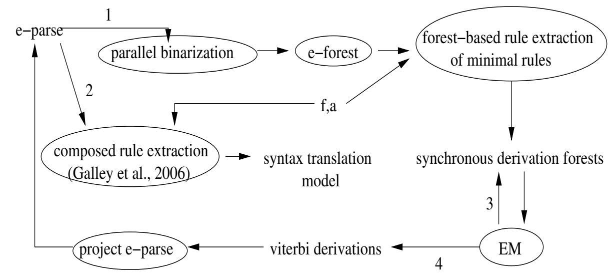

Figure 5: Using the EM algorithm to choose restructuring.

In practice, we cannot enumerate all the exponential number of binarized trees for a given e-parse. We therefore use the packed forest to store all the binarizations that operate on an e-parse in a compact way, and then use the inside-outside algorithm (Lari and Young, 1990; Knight and Graehl, 2004) for model estimation.

The probability $p(\tau_\beta, \mathbf{f}, \mathbf{a})$ of a (τ_β, **f**, **a**)-tuple is what the basic syntax-based translation model is concerned with. It can be further computed by aggregating the rule probabilities $p(r)$ in each derivation ω in the set of all derivations Ω (Galley et al., 2004; Marcu et al., 2006). That is

$$p(\tau_\beta, \mathbf{f}, \mathbf{a}) \quad = \quad \sum_{\omega \in \Omega} \prod_{r \in \omega} p(r) \qquad (2)$$

Since it has been well-known that applying EM with tree fragments of different sizes causes overfitting (Johnson, 1998), and since it is also known that syntax MT models with larger composed rules in the mix significantly outperform rules that minimally explain the training data (minimal rules) in translation accuracy (Galley et al., 2006), we decompose $p(\tau_b, \mathbf{f}, \mathbf{a})$ using minimal rules during running of the EM algorithm, but, after the EM restructuring is finished, we build the final translation model using composed rules for evaluation.

Figure 5 is the actual pipeline that we use for EM binarization. We first generate a packed e-forest via parallel binarization. We then extract minimal translation rules from the (e-forest, **f**, **a**)-tuples, producing synchronous derivation forests. We run the inside-outside algorithm on the derivation forests until convergence. We obtain the Viterbi derivations and project the English parses from the derivations. Finally, we extract composed rules using Galley et al. (2006)'s (e-tree, **f**, **a**)-based rule extraction algorithm. This procedure corresponds to the path 13*42 in the pipeline.

7 Experiments

We carried out a series of experiments to compare the performance of different binarization methods in terms of BLEU on Chinese-to-English translation tasks.

7.1 Experimental setup

Our bitext consists of 16M words, all in the mainland-news domain. Our development set is a 925-line subset of the 993-line NIST02 evaluation set. We removed long sentences from the NIST02 evaluation set to speed up discriminative training. The test set is the full 919-line NIST03 evaluation set.

We used a bottom-up, CKY-style decoder that works with binary xRs rules obtained via a synchronous binarization procedure (Zhang et al., 2006). The decoder prunes hypotheses using strategies described in (Chiang, 2007).

The parse trees on the English side of the bitexts were generated using a parser (Soricut, 2004) implementing the Collins parsing models (Collins, 1997).

We used the EM procedure described in (Knight and Graehl, 2004) to perform the inside-outside algorithm on synchronous derivation forests and to generate the Viterbi derivation forest.

We used the rule extractor described in (Galley et al., 2006) to extract rules from (e-parse, **f**, **a**)-tuples, but we made an important modification: new nodes introduced by binarization will not be counted when computing the rule size limit unless they appear as the rule roots. The motivation is that binarization deepens the parses and increases the number of tree nodes. In (Galley et al., 2006), a composed rule is extracted only if the number of internal nodes it contains does not exceed a limit (i.e., 4), similar to the phrase length limit in phrase-based systems. This means that rules extracted from the restructured trees will be smaller than those from the unrestructured trees, if the $\overline{X}$ nodes are deleted. As shown in (Galley et al., 2006), smaller rules lose context, and thus give lower translation performance. Ignoring $\overline{X}$ nodes when computing the rule sizes preserves the unstructured rules in the resulting translation model and adds substructures as bonuses.

7.2 Experiment results

Table 1 shows the BLEU scores of mixed-cased and detokenized translations of different systems. We see that all the binarization methods improve the baseline system that does not apply any binarization algorithm. The EM-binarization performs the best among all the restructuring methods, leading to 1.0 BLEU point improvement. We also computed the bootstrap p-values (Riezler and Maxwell, 2005) for the pairwise BLEU comparison between the baseline system and any of the system trained from binarized trees. The significance test shows that the EM binarization result is statistically significant better than the baseline system ($p > 0.005$), even though the baseline is already quite strong. To our best knowledge, 37.94 is the highest BLEU score on this test set to date.

Also as shown in Table 1, the grammars trained from the binarized training trees are almost two times of the grammar size with no binarization. The extra rules are substructures factored out by these binarization methods.

How many more substructures (or translation rules) can be acquired is partially determined by how many more admissible nodes each binarization method can factorize, since rules are extractable only from admissible tree nodes. According to Table 1, binarization methods significantly increase the number of admissible nodes in the training trees. The EM binarization makes available the largest

EXPERIMENT	NIST03-BLEU	# RULES	# ADMISSIBLE NODES IN TRAINING
no-bin	36.94	63.4M	7,995,569
left binarization	37.47 ($p = 0.047$)	114.0M	10,463,148
right binarization	37.49 ($p = 0.044$)	113.0M	10,413,194
head binarization	37.54 ($p = 0.086$)	113.8M	10,534,339
EM binarization	**37.94** ($p = 0.0047$)	115.6M	10,658,859

Table 1: Translation performance, grammar size and # admissible nodes versus binarization algorithms. BLEU scores are for mixed-cased and detokenized translations, as we usually do for NIST MT evaluations.

nonterminal	left-binarization	right-binarization
NP	**96.97**%	3.03%
NP-C	**97.49**%	2.51%
NPB	0.25%	**99.75**%
VP	**93.90**%	6.10%
PP	**83.75**%	16.25%
ADJP	**87.83**%	12.17%
ADVP	**82.74**%	17.26%
S	**85.91**%	14.09%
S-C	18.88%	**81.12**%
SBAR	**96.69**%	3.31%
QP	**86.40**%	13.60%
PRN	**85.18**%	14.82%
WHNP	**97.93**%	2.07%
NX	**100**%	0
SINV	**87.78**%	12.22%
PRT	**100**%	0
SQ	**93.53**%	6.47%
CONJP	18.08%	**81.92**%

Table 2: Binarization bias learned by EM.

number of admissible nodes, and thus results in the most rules.

The EM binarization factorizes more admissible nodes because it mixes both left and right binarizations in the same tree. We computed the binarization biases learned by the EM algorithm for each nonterminal from the binarization forest of headed-parallel binarizations of the training trees, getting the statistics in Table 2. Of course, the binarization bias chosen by left-/right-binarization methods would be 100% deterministic. One noticeable message from Table 2 is that most of the categories are actually biased toward left-binarization, although our motivating example in our introduction section is for NPB, which needed right binarization. The main reason might be that the head sub-constituents of most categories tend to be on the left, but according to the performance comparison between head binarization and EM binarization, head binarization does not suffice because we still need to choose the binarization between left and right if they both are head binarizations.

8 Conclusions

In this paper, we not only studied the impact of simple tree binarization algorithms on the performance of end-to-end syntax-based MT, but also proposed binarization methods that mix more than one simple binarization in the binarization of the same parse tree. Binarizing a tree node whether to the left or to the right was learned by employing the EM algorithm on a set of alternative binarizations and by choosing the Viterbi one. The EM binarization method is informed by word alignments such that unnecessary new tree nodes will not be "blindly" introduced.

To our best knowledge, our research is the first work that aims to generalize a syntax-based translation model by restructuring and achieves significant improvement on a strong baseline. Our work differs from traditional work on binarization of synchronous grammars in that we are not concerned with the equivalence of the binarized grammar to the original grammar, but intend to generalize the original grammar via restructuring of the training parse trees to improve translation performance.

Acknowledgments

The authors would like to thank David Chiang, Bryant Huang, and the anonymous reviewers for their valuable feedbacks.

References

E. Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, Seattle, May.

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

Michael Collins. 1997. Three generative, lexicalized models for statistical parsing. In *Proceedings of the*

35th Annual Meeting of the Association for Computational Linguistics (ACL), pages 16–23, Madrid, Spain, July.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 205–208, Sapporo, July.

M. Galley, M. Hopkins, K. Knight, and D. Marcu. 2004. What's in a Translation Rule? In *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL)*, Boston, Massachusetts.

M. Galley, J. Graehl, K. Knight, D. Marcu, S. DeNeefe, W. Wang, and I. Thayer. 2006. Scalable Inference and Training of Context-Rich Syntactic Models. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (ACL)*.

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

M. Johnson. 1998. The DOP estimation method is biased and inconsistent. *Computational Linguistics*, 28(1):71–76.

K. Knight and J. Graehl. 2004. Training Tree Transducers. In *Proceedings of NAACL-HLT*.

K. Lari and S. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, pages 35–56.

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

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

I. Dan Melamed, Giorgio Satta, and Benjamin Wellington. 2004. Generalized multitext grammars. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, Barcelona, Spain.

Stefan Riezler and John T. Maxwell. 2005. On some pitfalls in automatic evaluation and significance testing for MT. In *Proc. ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization*.

Radu Soricut. 2004. A reimplementation of Collins's parsing models. Technical report, Information Sciences Institute, Department of Computer Science University of Southern California.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. 2006. Synchronous binarization for machine translation. In *Proceedings of the HLT-NAACL*.