Dependency Forest for Statistical Machine Translation

Zhaopeng Tu [†] **Yang Liu** [†] **Young-Sook Hwang** [‡] **Qun Liu** [†] **Shouxun Lin** [†]

[†]Key Lab. of Intelligent Info. Processing
Institute of Computing Technology
Chinese Academy of Sciences
{tuzhaopeng,yliu,liuqun,sxlin}@ict.ac.cn

[‡]HILab Convergence Technology Center
C&I Business
SKTelecom
yshwang@sktelecom.com

Abstract

We propose a structure called *dependency forest* for statistical machine translation. A dependency forest compactly represents multiple dependency trees. We develop new algorithms for extracting string-to-dependency rules and training dependency language models. Our forest-based string-to-dependency system obtains significant improvements ranging from 1.36 to 1.46 BLEU points over the tree-based baseline on the NIST 2004/2005/2006 Chinese-English test sets.

1 Introduction

Dependency grammars have become increasingly popular in syntax-based statistical machine translation (SMT). One important advantage of dependency grammars is that they directly capture the dependencies between words, which are key to resolving most parsing ambiguities. As a result, incorporating dependency trees proves to be effective in improving statistical machine translation (Quirk et al., 2005; Ding and Palmer, 2005; Shen et al., 2008).

However, most dependency-based translation systems suffer from a major drawback: they only use 1-best dependency trees for rule extraction, dependency language model training, and decoding, which potentially introduces translation mistakes due to the propagation of parsing errors (Quirk and Corston-Oliver, 2006). While the treelet system (Quirk et al., 2005) takes a dependency tree as input, the string-to-dependency system (Shen et al., 2008) decodes on a source-language string. However, as we will show, the string-to-dependency system still commits to using degenerate rules and dependency language models learned from noisy 1-best trees.

To alleviate this problem, an obvious solution is to offer more alternatives. Recent studies have shown that SMT systems can benefit from widening the annotation pipeline: using packed forests instead of 1-best trees (Mi and Huang, 2008), word lattices instead of 1-best segmentations (Dyer et al., 2008), and weighted alignment matrices instead of 1-best alignments (Liu et al., 2009).

Along the same direction, we propose a structure called *dependency forest*, which encodes exponentially many dependency trees compactly, for dependency-based translation systems. In this paper, we develop two new algorithms for extracting string-to-dependency rules and for training dependency language models, respectively. We show that using the rules and dependency language models learned from dependency forests leads to consistent and significant improvements over that of using 1-best trees on the NIST 2004/2005/2006 Chinese-English test sets.

2 Background

Figure 1 shows a dependency tree of an English sentence *he saw a boy with a telescope*. Arrows point from the child to the parent, which is often referred to as the head of the child. For example, in Figure 1, *saw* is the head of *he*. A dependency tree is more compact than its constituent counterpart because there is no need to build a large superstructure over a sentence.

Shen et al. (2008) propose a novel string-to-dependency translation model that features two important advantages. First, they define that a string-to-dependency rule must have a *well-formed* dependency structure on the target side, which makes efficient dynamic programming possible and manages to retain most useful non-constituent rules. A well-formed structure can be either *fixed* or *floating* . A fixed structure is a

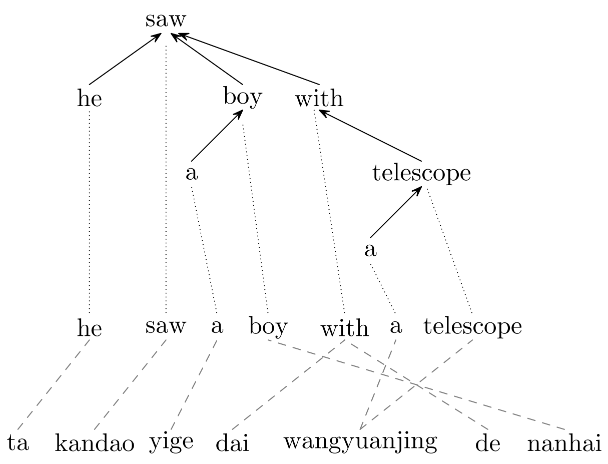

Figure 1: A training example for tree-based rule extraction.

dependency tree with all the children complete. Floating structures consist of sibling nodes of a common head, but the head itself is unspecified or floating. For example, Figure 2(a) and Figure 2(b) are two fixed structures while Figure 2(c) is a floating one.

Formally, for a given sentence $w_{1:l} = w_1 \dots w_l$, $d_1 \dots d_l$ represent the parent word IDs for each word. If w_i is a root, we define $d_i = 0$.

Definition 1. *A dependency structure $d_{i..j}$ is* **fixed on head** *h, where $h \notin [i, j]$, or* **fixed** *for short, if and only if it meets the following conditions*

- $d_h \notin [i, j]$
- $\forall k \in [i, j]$ and $k \neq h, d_k \in [i, j]$
- $\forall k \notin [i, j]$, $d_k = h$ or $d_k \notin [i, j]$

Definition 2. *A dependency structure $d_{i..j}$ is* **floating with children** *C, for a non-empty set C $\subseteq \{i, ..., j\}$, or* **floating** *for short, if and only if it meets the following conditions*

- $\exists h \notin [i, j], s.t. \forall k \in C, d_k = h$
- $\forall k \in [i, j]$ and $k \notin C, d_k \in [i, j]$
- $\forall k \notin [i, j]$, $d_k \notin [i, j]$

A dependency structure is **well-formed** if and only if it is either **fixed** or **floating**.

2.1 Tree-based Rule Extraction

Figure 1 shows a training example consisting of an English dependency tree, its Chinese translation,

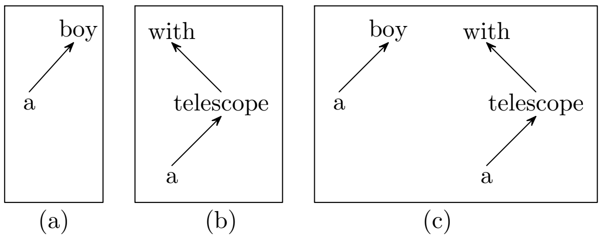

Figure 2: Well-formed dependency structures corresponding to Figure 1. (a) and (b) are fixed and (c) is floating.

and the word alignments between them. To facilitate identifying the correspondence between the English and Chinese words, we also gives the English sentence. Extracting string-to-dependency rules from aligned string-dependency pairs is similar to extracting SCFG (Chiang, 2007) except that the target side of a rule is a well-formed structure. For example, we can first extract a string-to-dependency rule that is consistent with the word alignment (Och and Ney, 2004):

$$\textit{with } ((a) \textit{ telescope}) \rightarrow \textit{dai wangyuanjing de}$$

Then a smaller rule

$$(a) \textit{ telescope} \rightarrow \textit{wangyuanjing}$$

can be subtracted to obtain a rule with one non-terminal:

$$\textit{with } (X_1) \rightarrow \textit{dai } X_1 \textit{ de}$$

where X is a non-terminal and the subscript indicates the correspondence between non-terminals on the source and target sides.

2.2 Tree-based Dependency Language Model

As dependency relations directly model the semantics structure of a sentence, Shen et al. (2008) introduce *dependency language model* to better account for the generation of target sentences. Compared with the conventional n-gram language models, dependency language model excels at capturing non-local dependencies between words (e.g., *saw ... with* in Figure 1). Given a dependency tree, its dependency language model probability is a product of three sub-models defined between headwords and their dependants. For example, the probability of the tree in Figure 1 can

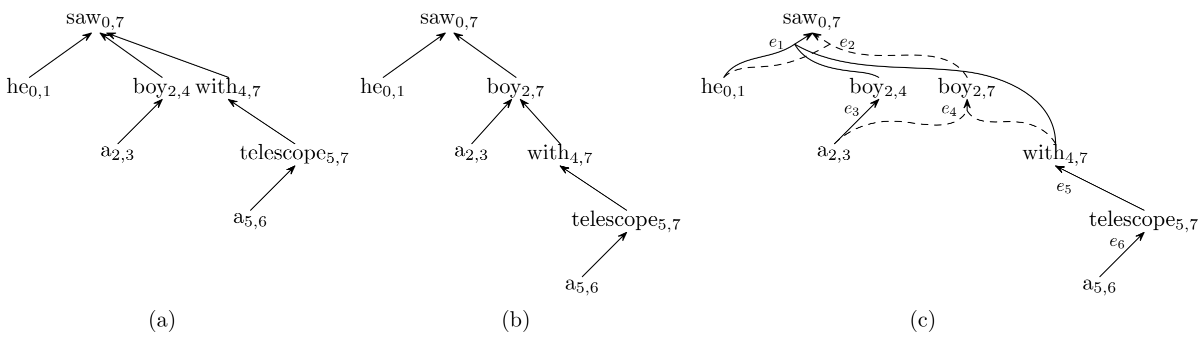

Figure 3: (a) the dependency tree in Figure 1, (b) another dependency tree for the same sentence, and (c) a dependency forest compactly represents the two trees.

be calculated as:

$$\begin{aligned} Prob &= P_T(saw) \\ &\quad \times P_L(he|saw\text{-as-head}) \\ &\quad \times P_R(boy|saw\text{-as-head}) \\ &\quad \times P_R(with|boy, saw\text{-as-head}) \\ &\quad \times P_L(a|boy\text{-as-head}) \\ &\quad \times P_R(telescope|with\text{-as-head}) \\ &\quad \times P_L(a|telescope\text{-as-head}) \end{aligned}$$

where $P_T(x)$ is the probability of word x being the root of a dependency tree. P_L and P_R are the generative probabilities of left and right sides respectively.

As the string-to-tree system relies on 1-best trees for parameter estimation, the quality of rule table and dependency language model might be affected by parsing errors and therefore ultimately results in translation mistakes.

3 Dependency Forest

We propose to encode multiple dependency trees in a compact representation called dependency forest, which offers an elegant solution to the problem of parsing error propagation.

Figures 3(a) and 3(b) show two dependency trees for the example English sentence in Figure 1. The prepositional phrase *with a telescope* could either depend on *saw* or *boy*. Figure 3(c) is a dependency forest compactly represents the two trees by sharing common nodes and edges.

Each **node** in a dependency forest is a word. To distinguish among nodes, we attach a **span** to each node. For example, in Figure 1, the span of the first *a* is $(2, 3)$ because it is the third word in the sentence. As the fourth word *boy* dominates the node $a_{2,3}$, it can be referred to as $boy_{2,4}$. Note that the position of *boy* itself is taken into consideration. Similarly, the word *boy* in Figure 3(b) can be represented as $boy_{2,7}$.

The nodes in a dependency forest are connected by **hyperedges**. While an edge in a dependency tree only points from a dependent to its head, a hyperedge groups all the dependants that have a common head. For example, in Figure 3(c), the hyperedge

$$e_1\text{: } \langle (he_{0,1}, boy_{2,4}, with_{4,7}), saw_{0,7} \rangle$$

denotes that $he_{0,1}$, $boy_{2,4}$, and $with_{4,7}$ are dependants (from left to right) of $saw_{0,7}$.

More formally, a *dependency forest* is a pair $\langle V, E \rangle$, where V is a set of nodes, and E is a set of hyperedges. For a given sentence $w_{1:l} = w_1 \dots w_l$, each node $v \in V$ is in the form of $w_{i,j}$, which denotes that w dominates the substring from positions i through j (i.e., $w_{i+1} \dots w_j$). Each hyperedge $e \in E$ is a pair $\langle tails(e), head(e) \rangle$, where $head(e) \in V$ is the head and $tails(e) \in V$ are its dependants.

A dependency forest has a structure of a *hypergraph* such as packed forest (Klein and Manning, 2001; Huang and Chiang, 2005). However, while each hyperedge in a packed forest naturally treats the corresponding PCFG rule probability as its weight, it is challenging to make dependency forest to be a weighted hypergraph because dependency parsers usually only output a score, which can be either positive or negative, for each edge in a dependency tree rather than a hyperedge in a

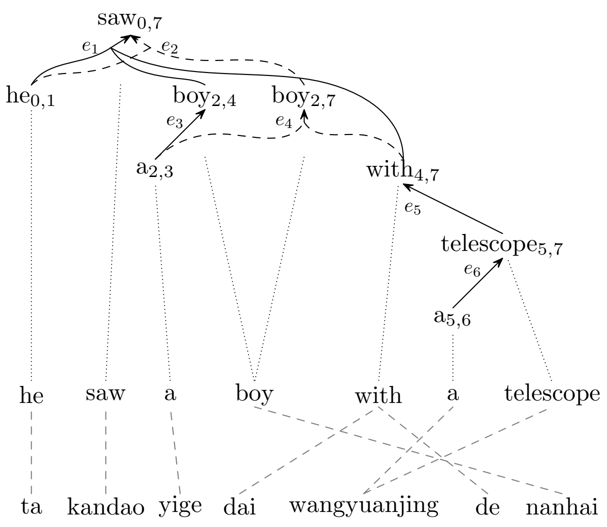

Figure 4: A training example for forest-based rule extraction.

dependency forest. For example, in Figure 3(a), the scores for the edges *he* $\rightarrow$ *saw*, *boy* $\rightarrow$ *saw*, and *with* $\rightarrow$ *saw* could be 13, 22, and -12, respectively.

To assign a probability to each hyperedge, we can first obtain a positive number for a hyperedge using the scores of the corresponding edges:[1]

$$c(e) = exp\left(\frac{\sum_{v \in tails(e)} s\big(v, head(e)\big)}{|tails(e)|}\right) \qquad (1)$$

where $c(e)$ is the count of a hyperedge e, $head(e)$ is a head, $tails(e)$ is a set of dependants of the head, v is one dependant, and $s(v, head(e))$ is the score of an edge from v to $head(e)$. For example, the count of the hyperedge e_1 in Figure 3(c) is

$$c(e_1) = exp\left(\frac{13 + 22 - 12}{3}\right) \qquad (2)$$

Then, the probability of a hyperedge can be obtained by normalizing the count among all hyperedges with the same head collected from a training corpus:

$$p(e) = \frac{c(e)}{\sum_{e':head(e')=head(e)} c(e')} \qquad (3)$$

Therefore, we obtain a weighted dependency forest in which each hyperedge has a probability.

[1]It is difficult to assign a probability to each hyperedge. The current method is arbitrary, and we will improve it in the future.

Algorithm 1 Forest-based Initial Phrase Extraction

```
    Input: a source sentence $\psi$, a forest $F$, an alignment $a$, and $k$
    Output: minimal initial phrase set $\mathcal{R}$
 1: for each node $v \in V$ in a bottom-up order do
 2:     for each hyperedge $e \in E$ and $head(e) = v$ do
 3:         $W \leftarrow \emptyset$
 4:         fixs $\leftarrow$ EnumFixed$(v, modifiers(e))$
 5:         floatings $\leftarrow$ EnumFloating$(modifiers(e))$
 6:         add structures fixs, floatings to $W$
 7:         for each $\omega \in W$ do
 8:             if $\omega$ is consistent with $a$ then
 9:                 generate a rule $r$
10:                 $\mathcal{R}$.append$(r)$
11:     keep $k$-best dependency structures for $v$
```

4 Forest-based Rule Extraction

In tree-based rule extraction, one just needs to first enumerate all bilingual phrases that are consistent with word alignment and then check whether the dependency structures over the target phrases are well-formed. However, this algorithm fails to work in the forest scenario because there are usually exponentially many well-formed structures over a target phrase.

The GHKM algorithm (Galley et al., 2004), which is originally developed for extracting tree-to-string rules from 1-best trees, has been successfully extended to packed forests recently (Mi and Huang, 2008). The algorithm distinguishes between minimal and composed rules. Although there are exponentially many composed rules, the number of minimal rules extracted from each node is rather limited (e.g., one or zero). Therefore, one can obtain promising composed rules by combining minimal rules.

Unfortunately, the GHKM algorithm cannot be applied to extracting string-to-dependency rules from dependency forests. This is because the GHKM algorithm requires a complete subtree to exist in a rule while neither fixed nor floating dependency structures ensure that all dependants of a head are included. For example, the floating structure shown in Figure 2(c) actually contains two trees.

Alternatively, our algorithm searches for well-formed structures for each node in a bottom-up style. Algorithm 1 shows the algorithm for extracting initial phrases, that is, rules without non-terminals from dependency forests. The algorithm maintains k-best well-formed structures for each node (line 11). The well-formed structures of a head can be constructed from those of its dependants. For example, in Figure 4, as the fixed structure rooted at *telescope*$_{5,7}$ is

$$(a)\ \textit{telescope}$$

we can obtain a fixed structure rooted for the node *with*$_{4,7}$ by attaching the fixed structure of its dependant to the node (*EnumFixed* in line 4). Figure 2(b) shows the resulting fixed structure.

Similarly, the floating structure for the node *saw*$_{0,7}$ can be obtained by concatenating the fixed structures of its dependants *boy*$_{2,4}$ and *with*$_{4,7}$ (*EnumFloating* in line 5). Figure 2(c) shows the resulting fixed structure. The algorithm is similar to Wang et al. (2007), which binarize each constituent node to create some intermediate nodes that correspond to the floating structures.

Therefore, we can find k-best fixed and floating structures for a node in a dependency forest by manipulating the fixed structures of its dependants. Then we can extract string-to-dependency rules if the dependency structures are consistent with the word alignment.

How to judge a well-formed structure extracted from a node is better than others? We follow Mi and Huang (2008) to assign a **fractional count** to each well-formed structure. Given a tree fragment t, we use the inside-outside algorithm to compute its posterior probability:

$$\begin{aligned} \alpha\beta(t) &= \alpha(root(t)) \times \prod_{e \in t} p(e) \\ &\quad \times \prod_{v \in leaves(t)} \beta(v) \end{aligned} \tag{4}$$

where $root(t)$ is the root of the tree, e is an edge, $leaves(t)$ is a set of leaves of the tree, $\alpha(\cdot)$ is outside probability, and $\beta(\cdot)$ is inside probability.

For example, the subtree rooted at *boy*$_{2,7}$ in Figure 4 has the following posterior probability:

$$\begin{aligned} &\alpha(\textit{boy}_{2,7}) \times p(e_4) \times p(e_5) \\ &\times p(e_6) \times \beta(\textit{a}_{2,3}) \times \beta(\textit{a}_{5,6}) \end{aligned} \tag{5}$$

Now the fractional count of the subtree t is

$$c(t) = \frac{\alpha\beta(t)}{\alpha\beta(TOP)} \tag{6}$$

where TOP denotes the root node of the forest.

As a well-formed structure might be non-constituent, we approximate the fractional count by taking that of the minimal constituent tree fragment that contains the well-formed structure. Finally, the fractional counts of well-formed structures can be used to compute the relative frequencies of the rules having them on the target side (Mi and Huang, 2008):

$$\phi(r|lhs(r)) = \frac{c(r)}{\sum_{r':lhs(r')=lhs(r)} c(r')} \tag{7}$$

$$\phi(r|rhs(r)) = \frac{c(r)}{\sum_{r':rhs(r')=rhs(r)} c(r')} \tag{8}$$

Often, our approach extracts a large amount of rules from training corpus as we usually retain exponentially many well-formed structures over a target phrase. To maintain a reasonable rule table size, we discard any rule that has a fractional count lower that a threshold t.

5 Forest-based Dependency Language Model Training

Dependency language model plays an important role in string-to-dependency system. Shen et al. (2008) show that string-to-dependency system achieves 1.48 point improvement in BLEU along with dependency language model, while no improvement without it. However, the string-to-dependency system still commits to using dependency language model from noisy 1-best trees. We now turn to dependency forest for it encodes multiple dependency trees.

To train a dependency language model from a dependency forest, we need to collect all heads and their dependants. This can be easily done by enumerating all hyperedges. Similarly, we use the inside-outside algorithm to compute the posterior probability of each hyperedge e,

$$\begin{aligned} \alpha\beta(e) &= \alpha(head(e)) \times p(e) \\ &\quad \times \prod_{v \in tailes(e)} \beta(v) \end{aligned} \tag{9}$$

For example, the posterior probability of the hyperedge e_2 in Figure 4 is calculated as

$$\begin{aligned} \alpha\beta(e_2) &= \alpha(\textit{saw}_{0,7}) \times p(e_2) \\ &\quad \times \beta(\textit{he}_{0,1}) \times \beta(\textit{boy}_{2,7}) \end{aligned} \tag{10}$$

Rule	DepLM	NIST 2004	NIST 2005	NIST 2006	time
tree	tree	33.97	30.21	30.73	19.6
tree	forest	34.42*	31.06*	31.37*	24.1
forest	tree	34.60*	31.16*	31.45*	21.7
forest	forest	**35.33****	**31.57****	**32.19****	28.5

Table 1: BLEU scores and average decoding time (second/sentence) on the Chinese-English test sets. The baseline system (row 2) used the rule table and dependency language model learned both from 1-best dependency trees. We use " *" and "**" to denote a result is better than baseline significantly at $p < 0.05$ and $p < 0.01$, respectively.

Then, we can obtain the fractional count of a hyperedge e,

$$c(e) = \frac{\alpha\beta(e)}{\alpha\beta(TOP)} \tag{11}$$

Each n-gram (e.g., "boy-as-head a") is assigned the same fractional count of the hyperedge it belongs to.

We also tried training dependency language model as in (Shen et al., 2008), which means all hyperedges were on equal footing without regarding probabilities. However, the performance is about 0.8 point lower in BLEU. One possbile reason is that hyperedges with probabilities could distinguish high quality structures better.

6 Experiments

6.1 Results on the Chinese-English Task

We used the FBIS corpus (6.9M Chinese words + 8.9M English words) as our bilingual training corpus. We ran GIZA++ (Och and Ney, 2000) to obtain word alignments. We trained a 4-gram language model on the Xinhua portion of GIGAWORD corpus using the SRI Language Modeling Toolkit (Stolcke, 2002) with modified Kneser-Ney smoothing (Kneser and Ney, 1995). We optimized feature weights using the minimum error rate training algorithm (Och and Ney, 2002) on the NIST 2002 test set. We evaluated the translation quality using case-insensitive BLEU metric (Papineni et al., 2002) on the NIST 2004/2005/2006 test sets.

To obtain dependency trees and forests, we parsed the English sentences of the FBIS corpus using a shift-reduce dependency parser that enables beam search (Huang et al., 2009). We only retained the best well-formed structure for each node when extracting string-to-tree rules from dependency forests (i.e., $k = 1$). We trained two 3-gram depLMs (one from trees and another from forests) on English side of FBIS corpus plus 2M sentence pairs from other LDC corpus.

Rules	Size	New Rules
tree	7.2M	-
forest	7.6M	16.86%

Table 2: Statistics of rules. The last column shows the ratio of rules extracted from non 1-best parses being used in 1-best derivations.

After extracting rules and training depLMs, we ran our replication of string-to-dependency system (Shen et al., 2008) to translate the development and test sets.

Table 1 shows the BLEU scores on the test sets. The first column "Rule" indicates where the string-to-dependency rules are learned from: 1-best dependency trees or dependency forests. Similarly, the second column "DepLM" also distinguish between the two sources for training dependency language models. The baseline system used the rule table and dependency language model both learned from 1-best dependency trees. We find that adding the rule table and dependency language models obtained from dependency forests improves string-to-dependency translation consistently and significantly, ranging from +1.3 to +1.4 BLEU points. In addition, using the rule table and dependency language model trained from forest only increases decoding time insignificantly.

How many rules extracted from non 1-best

Rule	DepLM	BLEU
tree	tree	22.31
tree	forest	22.73*
forest	tree	22.80*
forest	forest	**23.12****

Table 3: BLEU scores on the Korean-Chinese test set.

parses are used by the decoder? Table 2 shows the number of rules filtered on the test set. We observe that the rule table size hardly increases. One possible reason is that we only keep the best dependency structure for each node. The last row shows that 16.86% of the rules used in 1-best derivations are extracted from non 1-best parses in the forests, indicating that some useful rules cannot be extracted from 1-best parses.

6.2 Results on the Korean-Chinese Task

To examine the efficacy of our approach on different language pairs, we carried out an experiment on Korean-Chinese translation. The training corpus contains about 8.2M Korean words and 7.3M Chinese words. The Chinese sentences were used to train a 5-gram language model as well as a 3-gram dependency language model. Both the development and test sets consist of 1,006 sentences with single reference. Table 3 shows the BLEU scores on the test set. Again, our forest-based approach achieves significant improvement over the baseline ($p < 0.01$).

6.3 Effect of K-best

We investigated the effect of different k-best structures for each node on translation quality (BLEU scores on the NIST 2005 set) and the rule table size (filtered for the tuning and test sets), as shown in Figure 5. To save time, we extracted rules just from the first 30K sentence pairs of the FBIS corpus. We trained a language model and depLMs on the English sentences. We used 10 different k: 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10. Obviously, the higher the k is, the more rules are extracted. When k=10, the number of rules used on the tuning and test sets was 1,299,290 and the BLEU score was 20.88. Generally, both the number of rules and the BLEU score went up with

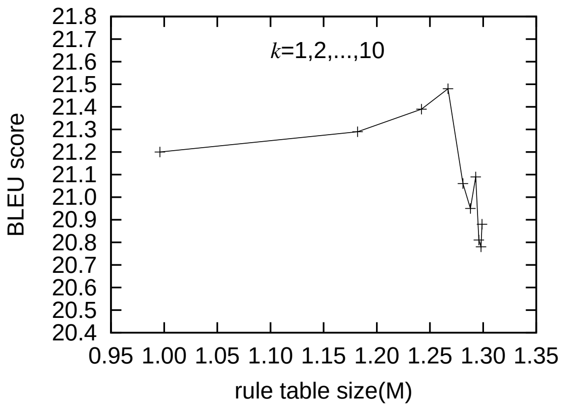

Figure 5: Effect of k-best on rule table size and translation quality.

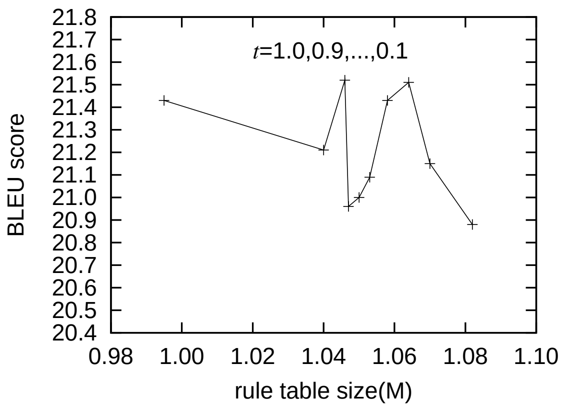

Figure 6: Effect of pruning threshold on rule table size and translation quality.

the increase of k. However, this trend did not hold within the range [4,10]. We conjecture that when retaining more dependency structures for each node, low quality structures would be introduced, resulting in much rules of low quality.

An interesting finding is that the rule table grew rapidly when k is in range [1,4], while gradually within the range [4,10]. One possible reason is that there are limited different dependency structures in the spans with a maximal length of 10, which the target side of rules cover.

6.4 Effect of Pruning Threshold

Figure 6 shows the effect of pruning threshold on translation quality and the rule table size. We retained 10-best dependency structures for each node in dependency forests. We used 10 different pruning thresholds: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0. Intuitively, the higher the pruning threshold is, the less rules are extracted. When t=0.1, the number of rules used on the tuning and test sets was 1,081,841 and the BLEU score was 20.68.

Lots of rules are pruned when the pruning threshold increases from 0.0 to 0.3 (around 20%). After pruning away these rules, we achieved 0.6 point improvement in BLEU. However, when we filtered more rules, the BLEU score went down.

Figures 5 and 6 show that using two parameters that have to be hand-tuned achieves a small improvement at the expense of an additional complexity. To simplify the approach, we only keep the best dependency structure for each node without pruning any rule.

7 Related Works

While Mi and Huang (2008) and we both use forests for rule extraction, there remain two major differences. Firstly, Mi and Huang (2008) use a packed forest, while we use a dependency forest. Packed forest is a natural weighted hypergraph (Klein and Manning, 2001; Huang and Chiang, 2005), for each hyperedge treats the corresponding PCFG rule probability as its weight. However, it is challenging to make dependency forest to be a weighted hypergraph because dependency parsers usually only output a score for each edge in a dependency tree rather than a hyperedge in a dependency forest. Secondly, The GHKM algorithm (Galley et al., 2004), which is originally developed for extracting tree-to-string rules from 1-best trees, has been successfully extended to packed forests recently (Mi and Huang, 2008). Unfortunately, the GHKM algorithm cannot be applied to extracting string-to-dependency rules from dependency forests, because the GHKM algorithm requires a complete subtree to exist in a rule while neither fixed nor floating dependency structures ensure that all dependants of a head are included.

8 Conclusion and Future Work

In this paper, we have proposed to use dependency forests instead of 1-best parses to extract string-to-dependency tree rules and train dependency language models. Our experiments show that our approach improves translation quality significantly over a state-of-the-art string-to-dependency system on various language pairs and test sets. We believe that dependency forest can also be used to improve the dependency treelet system (Quirk et al., 2005) that takes 1-best trees as input.

Acknowledgement

The authors were supported by SK Telecom C&I Business, and National Natural Science Foundation of China, Contracts 60736014 and 60903138. We thank the anonymous reviewers for their insightful comments. We are also grateful to Wenbin Jiang for his invaluable help in dependency forest.

References

Chiang, David. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, pages 201–228.

Ding, Yuan and Martha Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of ACL*.

Dyer, Christopher, Smaranda Muresan, and Philip Resnik. 2008. Generalizing word lattice translation. In *Proceedings of ACL*.

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

Huang, Liang and David Chiang. 2005. Better k-best parsing. In *Proceedings of IWPT*.

Huang, Liang, Wenbin Jiang, and Qun Liu. 2009. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proceedings of EMNLP*.

Klein, Dan and Christopher D. Manning. 2001. Parsing and hypergraphs. In *Proceedings of IWPT*.

Kneser, R. and H. Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of Acoustics, Speech, and Signal*.

Liu, Yang, Tian Xia, Xinyan Xiao, and Qun Liu. 2009. Weighted alignment matrices for statistical machine translation. In *Proceedings of EMNLP*.

Mi, Haitao and Liang Huang. 2008. Forest-based translation rule extraction. In *Proceedings of EMNLP*.

Och, Franz J. and Hermann Ney. 2000. Improved statistical alignment models. In *Proceedings of ACL*.

Och, Franz J. and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of ACL*.

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

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of ACL*.

Quirk, Chris and Simon Corston-Oliver. 2006. The impact of parsing quality on syntactically-informed statistical machine translation. In *Proceedings of EMNLP*.

Quirk, Chris, Arul Menezes, and Colin Cherry. 2005. Dependency treelet translation: syntactically informed phrasal smt. In *Proceedings of ACL*.

Shen, Libin, Jinxi Xu, and Ralph Weischedel. 2008. A new string-to-dependency machine translation algorithm with a target dependency language model. In *Proceedings of ACL*.

Stolcke, Andreas. 2002. Srilm - an extensible language modeling toolkit. In *Proceedings of ICSLP*.

Wang, Wei, Kevin Knight, and Daniel Marcu. 2007. Binarizing syntax trees to improve syntax-based machine translation accuracy. In *Proceedings of EMNLP*.