Fast Translation Rule Matching for Syntax-based Statistical Machine Translation

Hui Zhang[1, 2] **Min Zhang**[1] **Haizhou Li**[1] **Chew Lim Tan**[2]
[1]Institute for Infocomm Research [2]National University of Singapore
zhangh1982@gmail.com {mzhang, hli}@i2r.a-star.edu.sg tancl@comp.nus.edu.sg

Abstract

In a linguistically-motivated syntax-based translation system, the entire translation process is normally carried out in two steps, translation rule matching and target sentence decoding using the matched rules. Both steps are very time-consuming due to the tremendous number of translation rules, the exhaustive search in translation rule matching and the complex nature of the translation task itself. In this paper, we propose a hyper-tree-based fast algorithm for translation rule matching. Experimental results on the NIST MT-2003 Chinese-English translation task show that our algorithm is at least 19 times faster in rule matching and is able to help to save 57% of overall translation time over previous methods when using large fragment translation rules.

1 Introduction

Recently linguistically-motivated syntax-based translation method has achieved great success in statistical machine translation (SMT) (Galley et al., 2004; Liu et al., 2006, 2007; Zhang et al., 2007, 2008a; Mi et al., 2008; Mi and Huang 2008; Zhang et al., 2009). It translates a source sentence to its target one in two steps by using structured translation rules. In the first step, which is called translation rule matching step, all the applicable[1] translation rules are extracted from the entire rule set by matching the source parse tree/forest. The second step is to decode the source sentence into its target one using the extracted translation rules. Both of the two steps are very time-consuming due to the exponential number of translation rules and the complex nature of machine translation as an NP-hard search problem (Knight, 1999). In the SMT research community, the second step has been well studied and many methods have been proposed to speed up the decoding process, such as node-based or span-based beam search with different pruning strategies (Liu et al., 2006; Zhang et al., 2008a, 2008b) and cube pruning (Huang and Chiang, 2007; Mi et al., 2008). However, the first step attracts less attention. The previous solution to this problem is to do exhaustive searching with heuristics on each tree/forest node or on each source span. This solution becomes computationally infeasible when it is applied to packed forests with loose pruning threshold or rule sets with large tree fragments of large rule height and width. This not only overloads the translation process but also compromises the translation performance since as shown in our experiments the large tree fragment rules are also very useful.

To solve the above issue, in this paper, we propose a hyper-tree-based fast algorithm for translation rule matching. Our solution includes two steps. In the first step, all the translation rules are re-organized using our proposed hyper-tree structure, which is a compact representation of the entire translation rule set, in order to make the common parts of translation rules shared as much as possible. This enables the common parts of different translation rules to be visited only once in rule matching. Please note that the first step can be easily done off-line very fast. As a result, it does not consume real translation time. In the second step, we design a recursive algorithm to traverse the hyper-tree structure and the input source forest in a top-down manner to do the rule matching between them. As we will show later, the hyper-tree structure and the recursive algorithm significantly improve the speed of the rule matching and the entire translation process compared with previous methods.

With the proposed algorithm, we are able to carry out experiments with very loose pruning

[1] Given a source structure (either a parse tree or a parse forest), a translation rule is applicable if and only if the left hand side of the translation rule exactly matches a tree fragment of the given source structure.

Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 1037–1045,
Singapore, 6-7 August 2009.

thresholds and larger tree fragment rules efficiently. Experimental results on the NIST MT-2003 Chinese-English translation task shows that our algorithm is 19 times faster in rule matching and is able to save 57% of overall translation time over previous methods when using large fragment translation rules with height up to 5. It also shows that the larger rules with height of up to 5 significantly outperforms the rules with height of up to 3 by around 1 BLEU score.

The rest of this paper is organized as follows. Section 2 introduces the syntax-based translation system that we are working on. Section 3 reviews the previous work. Section 4 explains our solution while section 5 reports the experimental results. Section 6 concludes the paper.

2 Syntax-based Translation

This section briefly introduces the forest/tree-based tree-to-string translation model which serves as the translation platform in this paper.

2.1 Tree-to-string model

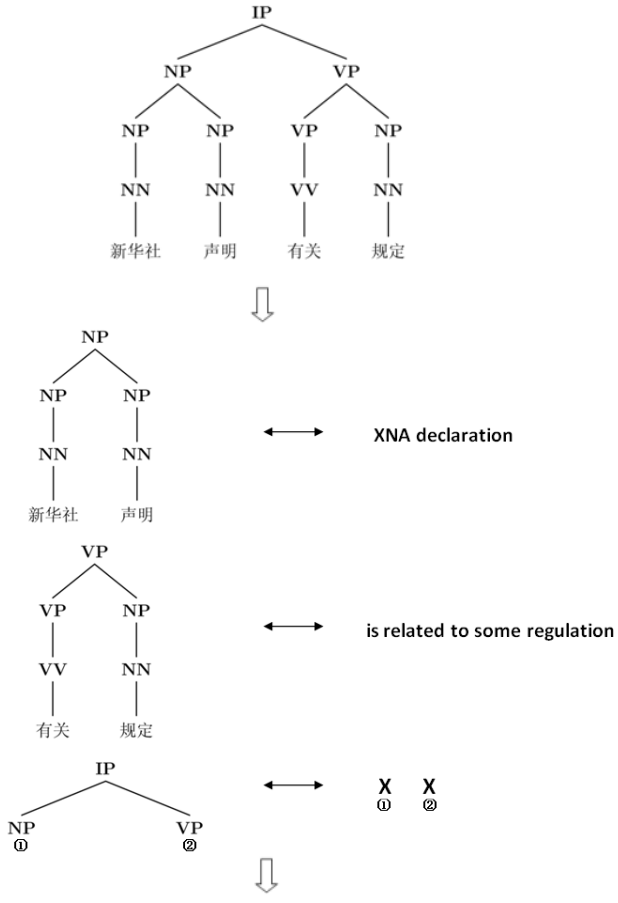

Figure 1. A tree-to-string translation process.

The tree-to-string model (Galley et al. 2004; Liu et al. 2006) views the translation as a structure mapping process, which first breaks the source syntax tree into many tree fragments and then maps each tree fragment into its corresponding target translation using translation rules, finally combines these target translations into a complete sentence. Fig. 1 illustrates this process. In real translation, the number of possible tree fragment segmentations for a given input tree is exponential in the number of tree nodes.

2.2 Forest-based translation

To overcome parse error for SMT, Mi and Huang (2008) propose forest-based translation by using a packed forest instead of a single syntax tree as the translation input. A packed forest (Tomita 1987; Klein and Manning, 2001; Huang and Chiang, 2005) is a compact representation of many possible parse trees of a sentence, which can be formally described as a triple $< V, E, S >$, where V is the set of non-terminal nodes, E is the set of hyper-edges and S is a sentence represented as an ordered word sequence. A hyper-edge in a packed forest is a group of edges in a tree which connects a father node to all its children nodes, representing a CFG-based parse rule. Fig. 2 is a packed forest incorporating two parse trees T1 and T2 of a sentence as shown in Fig. 3 and Fig. 4. Given a hyper-edge e, let h be its father node, then we say that e is attached to h.

A non-terminal node in a packed forest can be represented as "label [start, stop]", where "label" is its syntax category and "[start, stop]" is the range of words it covers. For example, the node in Fig. 5 pointed by the dark arrow is labelled as "NP[3,4]", where NP is its label and [3,4] means that it covers the span from the 3rd word to the 4th word. In forest-based translation, rule matching is much more complicated than the tree-based one.

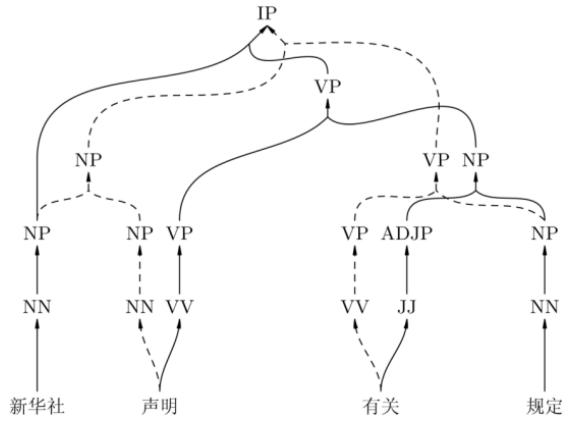

Figure 2. A packed forest

Zhang et al. (2009) reduce the tree sequence problem into tree problem by introducing virtual node and related forest conversion algorithms, so

the algorithm proposed in this paper is also applicable to the tree sequence-based models.

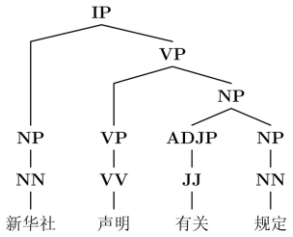

Figure 3. Tree 1 (T1)

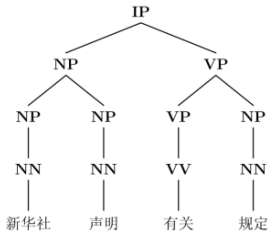

Figure 4. Tree 2 (T2)

3 Matching Methods in Previous Work

In this section, we discuss the two typical rule matching algorithms used in previous work.

3.1 Exhaustive search by tree fragments

This method generates all possible tree fragments rooted by each node in the source parse tree or forest, and then matches all the generated tree fragments against the source parts (left hand side) of translation rules to extract the useful rules (Zhang et al., 2008a).

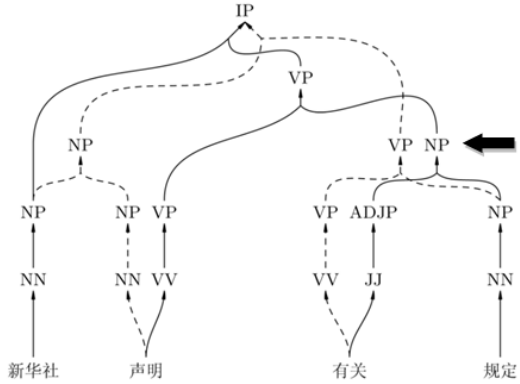

Figure 5. Node NP[3,4] in packed forest

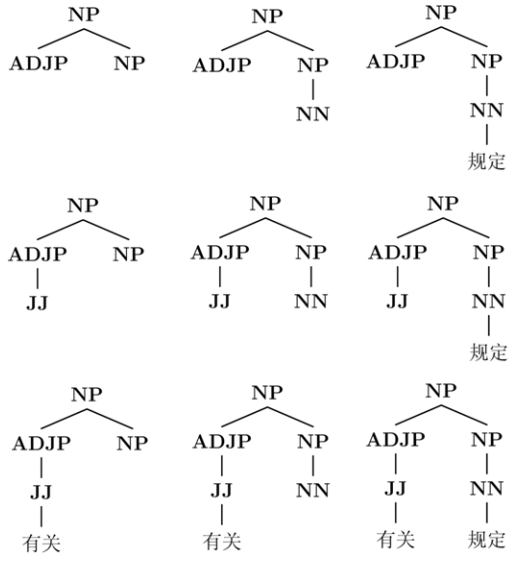

Figure 6. Candidate fragments on NP[3,4]

For example, if we want to extract useful rules for node NP[3,4] in Fig 5, we have to generate all the tree fragments rooted at node NP[3,4] as shown in Fig 6, and then query each fragment in the rule set. Let h be a node in the packed forest, $f(h)$ represents the number of possible tree fragments rooted at node h, then we have:

$$f(h) = \begin{cases} 0, & if\ h\ is\ a\ leaf\ node \\ \sum_{\substack{e\ is\ a\ hyper-edge \\ attached\ to\ h}} \prod_{\substack{c_i\ is\ the \\ i^{th}\ children \\ node\ in\ e}} (1 + f(c_i)), & otherwise \end{cases}$$

The above equation shows that the number of tree fragments is exponential to the span size, the height and the number of hyper-edges it covers. In a real system, one can use heuristics, e.g. the maximum number of nodes and the maximum height of fragment, to limit the number of possible fragments. However, these heuristics are very subjective and hard to optimize. In addition, they may filter out some "good" fragments.

3.2 Exhaustive search by rules

This method does not generate any source tree fragments. Instead, it does top-down recursive matching from each node one-by-one with each translation rule in the rule set (Mi and Huang 2008).

For example, given a translation rule with its left hand side as shown in Fig. 7, the rule matching between the given rule and the node IP[1,4] in Fig. 2 can be done as follows.

1. Decompose the left hand side of the translation rule as shown in Fig. 7 into a sequence of hyper-edges in top-down, left-to-right order as follows:

IP => NP VP; NP => NP NP; NP => NN;
NN => 声明

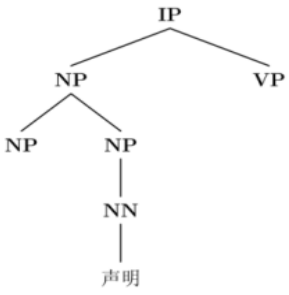

Figure 7. The left hand side of a rule

2. Pattern match these hyper-edges(rule) one-by-one in top-down left-to-right order from node IP[1,4]. If there is a continuous path in the forest matching all of these hyper-edges in order, then we can say that the rule is useful and matchable

with the tree fragment covered by the continuous path. The following illustrates the matching steps:

1. Match hyper-edge "IP => NP VP" with node IP[1,4]. There are two hyper-edges in the forest matching it: "IP[1,4] => NP[1,1] VP[2,4]" and "IP[1,4] => NP[1,2] VP [3,4]", which generates two candidate paths.

2. Since hyper-edge "NP => NP NP" fails to match NP[1,1], the path initiated with "IP[1,4] => NP[1,1] VP[2,4]" is pruned out.

3. Since there is a hyper-edge "NP[1,2] => NP[1,1] NP[2,2]" matching "NP => NP NP" on NP[1,2], then continue for further matching.

4. Since "NP=>NN" on NP[2,2] matches "NP[2,2] => NN[2,2]", then continue for further matching.

5. "NN=>声明" on NN[2,2] matches "NN[2,2] =>声明" and it is the last hyper-edge in the input rules. Finally, there is one continuous path successfully matching the left hand side of the input rule.

This method is able to avoid the exponential problem of the first method as described in the previous subsection. However, it has to do one-by-one pattern matching for each rule on each node. When the rule set is very large (indeed it is very large in the forest-based model even with a small training set), it becomes very slow, and even much slower than the first method.

4 The Proposed Hyper-tree-based Rule Matching Algorithm

In this section, we first explain the motivation why we re-organize the translation rule sets, and then elaborate how to re-organize the translation rules using our proposed hyper-tree structure. Finally we discuss the top-down rule matching algorithm between forest and hyper-tree.

4.1 Motivation

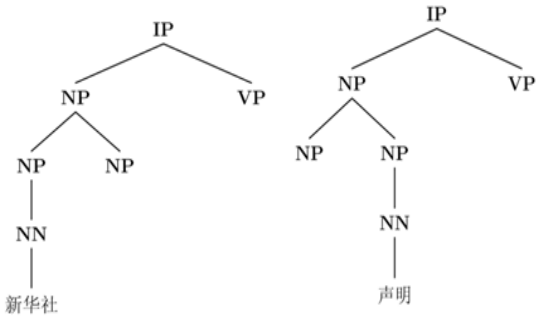

Figure 8. Two rules' left hand side

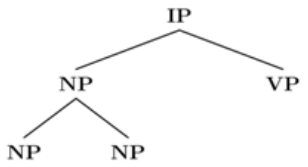

Figure 9. Common part of the two rules' left hand sides in Figure 8

Fig. 9 shows the common part of the left hand sides of two translation rules as shown in Fig. 8. In previous rule matching algorithm, the common parts are matched as many times as they appear in the rule set, which reduces the rule matching speed significantly. This motivates us to propose the hyper-tree structure and the rule matching algorithm to make the common parts shared by multiple translation rules to be visited only once in the entire rule matching process.

4.2 Hyper-node, hyper-path and hyper-tree

A hyper-tree is a compact representation of a group of tree translation rules with common parts shared. It consists of a set of hyper-nodes with edges connecting different hyper-nodes into a big tree. A hyper-tree is constructed from the translation rule sets in two steps:

1) Convert each tree translation rule into a hyper-path;
2) Construct the hyper-tree by incrementally adding each individual hyper-path into the hyper-tree.

A tree rule can be converted into a hyper-path without losing information. Fig. 10 demonstrates the conversion process:

1) We first fill the rule tree with virtual nodes to make all its leaves have the same depth to the root;
2) We then group all the nodes in the same tree level to form a single hyper-node, where we use a comma as a delimiter to separate the tree nodes with different father nodes;
3) A hyper-path is a set of hyper-nodes linked in a top-down manner.

The commas and virtual nodes ε are introduced to help to recover the original tree from the hyper-path. Given a tree node in a hyper-node, if there are n commas before it, then its father node is the $(n+1)^{th}$ tree node in the father hyper-node. If we could find father node for each node in hyper-nodes, then it is straightforward to recover the original tree from the hyper-path by just adding the edges between original father and children nodes except the virtual node ε.

After converting each tree rule into a hyper-path, we can organize the entire rule set into a big hyper-tree as shown in Figure 11. The concept of hyper-path and hyper-tree could be viewed as an extension of the "prefix merging" ideas for CFG rules (Klein and Manning 2001).

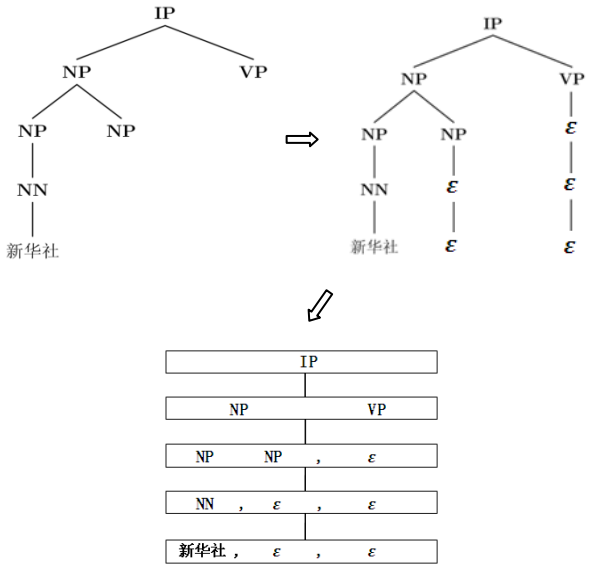

Figure 10. Convert tree to hyper-path

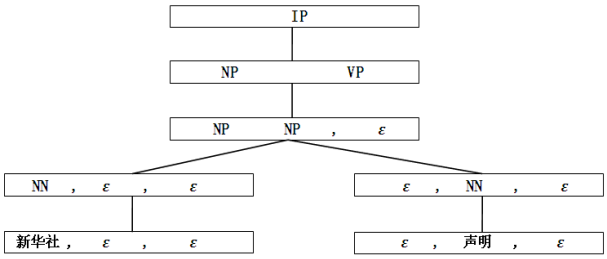

Figure 11. A hyper-tree example

Algorithm 1 shows how to organize the rule set into a big hyper-tree. The general process is that for each rule we convert it into a hyper-path and then add the hyper-path into a hyper-tree incrementally. However, there are many different hyper-trees generated given a big rule set. We then introduce a TOP label as the root node to link all the individual hyper-trees to a single big hyper-tree. Algorithm 2 shows the process of adding a hyper-path into a hyper-tree. Given a hyper-path, we do a top-down matching between the hyper-tree and the input hyper-path from root hyper-node until a leaf hyper-node is reached or there is no matching hyper-node at some level found. Then we add the remaining unmatchable part of the input hyper-path as the descendants of the last matchable hyper-node.

Please note that in Fig. 10 and Fig. 11, we ignore the target side (right hand side) of translation rules for easy discussion. Indeed, we can easily represent all the complete translation rules (not only left hand side) in Fig. 11 by simply adding the corresponding rule target sides into each hyper-node as done by line 5 of Algorithm 1.

Any hyper-path from the root to any hyper-node (not necessarily be a leaf of the hyper-tree) in a hyper-tree can represent a tree fragment. As a result, the hyper-tree in Fig. 11 can represent up to 6 candidate tree fragments. It is easy to understand that the maximum number of tree fragments that a hyper-tree can represent is equal to the number of hyper-nodes in it except the root. It is worth noting that a hyper-node in a hyper-tree without any target side rule attached means there is no translation rule corresponding to the tree fragment represented by the hyper-path from the root to the current hyper-node. The compact representation of the rule set by hyper-tree enables a fast algorithm to do translation rule matching.

Algorithm 1. *Compile rule set into hyper-tree*
Input: *rule set*
Output: *hyper-tree*

1. Initialize hyper-tree as a TOP node
2. **for** each rule in rule set **do**
3. Convert the left hand side tree to a hyper-path p
4. Add hyper-path p into hyper-tree
5. Add rule's right hand side to the leaf hyper-node of a hyper-path in the hyper-tree
6. **end for**

Algorithm 2. *Add hyper-path into hyper-tree*
Input: *hyper-path p and hyper-tree t*
Notation:
h: the height of hyper-path p
p(i) : the hyper-node of ith level (top-down) of p
TN: the hyper-node in hyper-tree
Output: *updated hyper-tree t*

1. Initialize TN as TOP
2. **for** i := 1 to h **do**
3. **if** there is a child c of TN has the same label as p(i) **then**
4. TN := c
5. **else**
6. Add a child c to TN, label c as p(i)
7. TN := c

4.3 Translation rule matching between forest and hyper-tree

Given the source parse forest and the translation rules represented in the hyper-tree structure, here we present a fast matching algorithm to extract so-called useful translation rules from the entire rule set in a top-down manner for each node of the forest.

As shown in Algorithm 3, the general process of the matching algorithm is as follows:

Algorithm 3. *Rule matching on one node*
Input: *hyper-tree T, forest F, and node n*
Notation:
FP: a pair <FNS, TN>, FNS is the frontier nodes of matched tree fragment,
TN is the hyper-tree node matching it
SFP: the queue of FP
Output: *Available rules on node n*

```
1.  if there is no child c of TOP having the same label as n
    then
2.      Return failure.
3.  else
4.      Initialize FP as <{n},c> and put it into SFP
5.      for each FP in SFP do
6.          SFP ← PropagateNextLevel(FP.FNS, FP.TN)
7.      for each FP in SFP do
8.          if the rule set attached to FP.TN is not empty
            then
9.              Add FP to result
```

Algorithm 4. *PropagateNextLevel*
Input: *Frontier node sequence FNS, hyper-tree node TN*
Notation:
CT: a child node of TN
the number of node sequence (separated by comma, see Fig 11) in CT is equal to the number of node in TN.
CT(i) : the ith node sequence in hyper-node CT
FNS(i): the ith node in FNS
TFNS: the temporary set of frontier node sequence
RFNS: the result set of frontier node sequence
FP: a pair of frontier node sequence and hyper-tree node
RFP: the result set of FP
Output: *RFP*

```
1.  for each child hyper-node CT of TN do
2.      for i:= 1 to the number of node sequence in CT do
3.          empty TFNS
4.          if CT(i) == ε then
5.              Add FNS(i) to TFNS.
6.          else
7.              for each hyper-edge e attached to FNS(i) do
8.                  if e.children match CT(i) then
9.                      Add e.children to TFNS
10.         if TFNS is empty then
11.             empty RFNS
12.             break
13.         else if i == 1 then
14.             RFNS := TFNS
15.         else
16.             RFNS := RFNS ⊗ TFNS
17.     for each FNS in RFNS do
18.         add <FNS, CT > into RFP
```

1) For each node *n* of the source forest if no child node of TOP in hyper-tree has the same label with it, it means that no rule matches any tree fragments rooted at the node *n* (i.e., no useful rules to be used for the node *n*) (line 1-2)

2) Otherwise, we match the sub-forest starting from the node *n* against a sub-hyper-tree starting from the matchable child node of TOP layer by layer in a top-down manner. There may be many possible tree fragments rooted at node *n* and each of them may have multiple useful translation rules. In our implementation, we maintain a data structure of FP = <FNS, TN> to record the currently matched tree fragment of forest and its corresponding hyper-tree node in the rule set, where FNS is the frontier node set of the current tree fragment and TN is the hyper-tree node. The data structure FP is used to help extract useful translation rules and is also used for further matching of larger tree fragments. Finally, all the FPs for the node *n* are kept in a queue. During the search, the queue size is dynamically increased. The matching algorithm terminates when all the FPs have been visited (line 5-6 and Algorithm 4).

3) In the final queue, each element (FP) of the queue contains the frontier node sequence of the matched tree fragment and its corresponding hyper-tree node. If the target side of a rule in the hyper-tree node is not empty, we just output the frontier nodes of the matched tree fragment, its root node *n* and all the useful translation rules for later translation process.

Algorithm 4 describes the detailed process of how to propagate the matching process down to the next level. <FNS, TN> is the current level frontier node sequence and hyper-tree node. Given a child hyper-node CT of TN (line 1), we try to find the group of next level frontier node sequence to match it (line 2-18). As shown in Fig 11, a hyper-node consists of a sequence of node sequence with comma as delimiter. For the i^{th} node sequence CT(i) in CT, If CT(i) is ε, that means FNS(i) is a leaf/frontier node in the matched tree fragment and thus no need to propagate to the next level (line 4-5). Otherwise, we try each hyper-edge e of FNS(i) to see whether its children match CT(i), and put the children of the matched hyper-edge into a temp set TFNS (line 7-9). If the temp set is empty, that means the current matching fails and no further expansion needs (line 10-12). Otherwise, we integrate current matched children into the final group of frontier node sequence (line 13-16) by Descartes Product (⊗). Finally, we construct all the <FNS, TN> pair for next level matching (line 17-18).

It would be interesting to study the time complexity of our Algorithm 3 and 4. Suppose the maximum number of children of each hyper-node in hyper-tree is N (line 1), the maximum number of node sequence in CT is M (line 2), the maximum number of hyper-edge in each node in packed forest is K (line 7), the maximum number of hyper-edge with same children representation in each node in packed forest is C (i.e. the maximum size of TFNS in line 16, and the maximum complexity of the Descartes Product in line 16

would be C^M), then the time complexity upper-bound of Algorithm 4 is $O(NM(K+C^M))$. For Algorithm 3, its time complexity is $O(RNM(K+C^M))$, where R is the maximum number of tree fragment matched in each node.

5 Experiment

5.1 Experimental settings

We carry out experiment on Chinese-English NIST evaluation tasks. We use FBIS corpus (250K sentence pairs) as training data with the source side parsed by a modified Charniak parser (Charniak 2000) which can output a packed forest. The Charniak Parser is trained on CTB5, tuned on 301-325 portion, with F1 score of 80.85% on 271-300 portion. We use GIZA++ (Och and Ney, 2003) to do *m-to-n* word-alignment and adopt heuristic "grow-diag-final-and" to do refinement. A 4-gram language model is trained on Gigaword 3 Xinhua portion by SRILM toolkit (Stolcke, 2002) with Kneser-Ney smoothing. We use NIST 2002 as development set and NIST 2003 as test set. The feature weights are tuned by the modified Koehn's MER (Och, 2003, Koehn, 2007) trainer. We use case-sensitive BLEU-4 (Papineni et al., 2002) to measure the quality of translation result. Zhang et al. 2004's implementation is used to do significant test.

Following (Mi and Huang 2008), we use viterbi algorithm to prune the forest. Instead of using a static pruning threshold (Mi and Huang 2008), we set the threshold as the distance of the probabilities of the n^{th} best tree and the 1^{st} best tree. It means the pruned forest is able to at least keep all the top n best trees. However, because of the sharing nature of the packed forest, it may still contain a large number of additional trees. Our statistic shows that when we set the threshold as the 100^{th} best tree, the average number of all possible trees in the forest is $1.2*10^5$ after pruning.

In our experiments, we compare our algorithm with the two traditional algorithms as discussed in section 3. For the "Exhaustive search by tree" algorithm, we use a bottom-up dynamic programming algorithm to generate all the candidate tree fragments rooted at each node. For the "Exhaustive search by rule" algorithm, we group all rules with the same left hand side in order to remove the duplicated matching for the same left hand side rules. All these settings aim for fair comparison.

5.2 Accuracy, speed vs. rule heights

We first compare the three algorithms' performance by setting the maximum rule height from 1 to 5. We set the forest pruning threshold to the 100^{th} best parse tree.

Table 1 compares the speed of the three algorithms. It clearly shows that the speed of both of the two traditional algorithms increases dramatically while the speed of our hyper-tree based algorithm is almost linear to the tree height. In the case of rule height of 5, the hyper-tree algorithm is at least 19 times (9.329/0.486) faster than the two traditional algorithms and saves 8.843(9.329 - 0.486) seconds in rule matching for each sentence on average, which contributes 57% (8.843/(9.329 + 6.21)) speed improvement to the overall translation.

H	Rule Matching			D
	Exhaustive by tree	Exhaustive by rule	Hyper-tree-based	
1	0.043	0.077	0.083	2.96
2	0.047	0.920	0.173	3.56
3	0.237	9.572	0.358	4.02
4	2.300	48.90	0.450	5.27
5	9.329	90.80	0.486	6.21

Table 1. Speed in seconds per sentence vs. rule height; "H" is rule height, "D" represents the decoding time after rule matching

Height	BLEU
1	0.1646
2	0.2498
3	0.2824
4	0.2874
5	**0.2925**
Moses	0.2625

Table 2. BLEU vs. rule height

Table 2 reports the BLEU score with different rule heights, where Moses, a state-of-the-art phrase-based SMT system, serves as the baseline system. It shows the BLEU score consistently improves as the rule height increases. In addition, one can see that the rules with maximum height of 5 are able to outperform the rules with maximum height of 3 by 1 BLEU score ($p<0.05$) and significantly outperforms Moses by 3 BLEU score ($p<0.01$). To our knowledge, this is the first time to report the performance of rules up to height of 5 for forest-based translation model.

We also study the distribution of the rules used in the 1-best translation output. The results are shown in Table 3; we could see something interesting that is as the rule height increases, the total number of rules with that height decreases, while the percentage of partial-lexicalized increases dramatically. And one thing needs to note is the percentage of partial-lexicalized rules with height of 1 is 0, since there is no partial-lexicalized rule with height of 1 in the rule set (the father node of a word is a pos tag node).

H	Total	Rule Type Percentage (%)		
		F	P	U
1	9814	76.58	0	23.42
2	5289	44.99	46.40	8.60
3	3925	18.39	77.25	4.35
4	1810	7.90	87.68	4.41
5	511	6.46	90.50	3.04

Table 3. statistics of rules used in the 1-best translation output, "F" means full-lexicalized, "P" means partial-lexicalized, "U" means unlexiclaizd.

5.3 Speed vs. forest pruning threshold

This section studies the impact of the forest pruning threshold on the rule matching speed when setting the maximum rule height to 5.

Threshold	Rule Matching		
	Exhaustive by tree	Exhaustive by rule	Hyper-tree-based
1	1.2	23.66	0.171
10	3.1	36.42	0.234
50	5.7	66.20	0.405
100	9.3	90.80	0.486
200	27.3	104.86	0.598
500	133.6	148.54	0.873

Table 4. Speed in seconds per sentence vs. forest pruning threshold

In Table 4, we can see that our hyper-tree based algorithm is the fastest among the three algorithms in all pruning threshold settings and even 150 times faster than both of the two traditional algorithms with threshold of 500th best. Table 5 shows the average number of parse trees embedded in a packed forest with different pruning thresholds per sentence. We can see that the number of trees increases exponentially when the pruning threshold increases linearly. When the threshold is 500th best, the average number of trees per sentence is $1.49*10^9$. However, even in this extreme case, the hyper-tree based algorithm is still capable of completing rule matching within 1 second.

Threshold	Number of Trees
1	1
10	32
50	5922
100	128860
200	$2.75*10^6$
500	$1.49*10^9$

Table 5. Average number of trees in packed forest with different pruning threshold.

5.4 Hyper-tree compression rate

As we describe in section 4.2, theoretically the number of tree fragments that a hyper-tree can represent is equal to the number of hyper-nodes in it. However, in real rule set, there is no guarantee that each tree fragment in the hyper-tree has corresponding translation rules. To gain insights into how effective the compact representation of the hyper-tree and how many hyper-nodes without translation rules, we define the compression rate as follows.

$$compression\ rate\ = \frac{the\ number\ tree\ fragments\ having\ trans\ rules}{the\ total\ number\ of\ tree\ fragments}$$

Table 6 reports the different statistics on the rule sets with different maximum rule heights ranging from 1 to 5. The reported statistics are the number of rules, the number of unique left hand side (since there may be more than one rules having the same left hand side), the number of hyper-nodes and the compression rate.

H	n_rules	n_LHS	n_nodes	c_rate
1	21588	10779	10779	100%
2	141632	51807	51903	99.8%
3	$1.73*10^6$	491268	494919	99.2%
4	$8.65*10^6$	2052731	2083296	98.5%
5	$1.89*10^7$	3966742	4043824	98.1%

Table 6. Statistics of rule set and hyper-tree. "H" is rule height, "n_rules" is the number of rules, "n_LHS" is the number of unique left hand side, "n_nodes" is the number of hyper-nodes in hyper-tree and "c_rate" is the compression rate.

Table 6 shows that in all the five cases, the compression rates of hyper-tree are all more than

98%. It means that almost all the tree fragments embedded in the hyper-tree have corresponding translation rules. As a result, we are able to use almost only one hyper-edge (i.e. only the frontier nodes of a tree fragment without any internal nodes) to represent all the rules with the same left hand side. This suggests that our hyper-tree is particularly effective in representing the tree translation rules compactly. It also shows that there are a lot of common parts among different translation rules.

All the experiments reported in this section convincingly demonstrate the effectiveness of our proposed hyper-tree representation of translation rules and the hyper-tree-based rule matching algorithm.

6 Conclusion

In this paper, we propose the concept of hyper-tree for compact rule representation and a hyper-tree-based fast algorithm for translation rule matching in a forest-based translation system. We compare our algorithm with two previous widely-used rule matching algorithms. Experimental results on the NIST Chinese-English MT 2003 evaluation data set show the rules with maximum rule height of 5 outperform those with height 3 by 1.0 BLEU and outperform MOSES by 3.0 BLEU. In the same test cases, our algorithm is at least 19 times faster than the two traditional algorithms, and contributes 57% speed improvement to the overall translation. We also show that in a more challenging setting (forest containing $1.49*10^9$ trees on average) our algorithm is 150 times faster than the two traditional algorithms. Finally, we show that the hyper-tree structure has more than 98% compression rate. It means the compact representation by the hyper-tree is very effective for translation rules.

References

Eugene Charniak. 2000. *A maximum-entropy inspired parser*. NAACL-00.

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

Liang Huang and David Chiang. 2005. *Better k-best Parsing*. IWPT-05.

Liang Huang and David Chiang. 2007. *Forest rescoring: Faster decoding with integrated language models*. ACL-07. 144–151

Dan Klein and Christopher D. Manning. 2001. *Parsing and Hypergraphs*. IWPT-2001.

Dan Klein and Christopher D. Manning. 2001. *Parsing with Treebank Grammars: Empirical Bounds, Theoretical Models, and the Structure of the Penn Treebank*. ACL - 2001. 338-345.

Kevin Knight. 1999. *Decoding Complexity in Word-Replacement Translation Models*. CL: J99-4005.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin and Evan Herbst. 2007. *Moses: Open Source Toolkit for Statistical Machine Translation.* ACL-07. 177-180. (poster)

Yang Liu, Qun Liu and Shouxun Lin. 2006. *Tree-to-String Alignment Template for Statistical Machine Translation.* COLING-ACL-06. 609-616.

Yang Liu, Yun Huang, Qun Liu and Shouxun Lin. 2007. *Forest-to-String Statistical Translation Rules.* ACL-07. 704-711.

Haitao Mi, Liang Huang, and Qun Liu. 2008. *Forest-based translation*. ACL-HLT-08. 192-199.

Haitao Mi and Liang Huang. 2008. *Forest-based Translation Rule Extraction*. EMNLP-08. 206-214

Franz J. Och. 2003. *Minimum error rate training in statistical machine translation.* ACL-03. 160-167.

Franz Josef Och and Hermann Ney. 2003. *A Systematic Comparison of Various Statistical Alignment Models*. Computational Linguistics. 29(1) 19-51

Kishore Papineni, Salim Roukos, ToddWard and Wei-Jing Zhu. 2002. *BLEU: a method for automatic evaluation of machine translation.* ACL-02.311-318.

Andreas Stolcke. 2002. *SRILM - an extensible language modeling toolkit.* ICSLP-02. 901-904.

Masaru Tomita. 1987. *An Efficient Augmented-Context-Free Parsing Algorithm*. Computational Linguistics 13(1-2): 31-46.

Hui Zhang, Min Zhang, Haizhou Li, Aiti Aw and Chew Lim Tan. 2009. *Forest-based Tree Sequence to String Translation Model.* ACL-IJCNLP-09.

Min Zhang, Hongfei Jiang, Ai Ti Aw, Jun Sun, Sheng Li and Chew Lim Tan. 2007. *A Tree-to-Tree Alignment-based Model for Statistical Machine Translation*. MT-Summit-07. 535-542.

Min Zhang, Hongfei Jiang, Aiti Aw, Haizhou Li, Chew Lim Tan, Sheng Li. 2008a. *A Tree Sequence Alignment-based Tree-to-Tree Translation Model.* ACL-HLT-08. 559-567.

Min Zhang, Hongfei Jiang, Haizhou Li, Aiti Aw, Sheng Li. 2008b. *Grammar Comparison Study for Translational Equivalence Modeling and Statistical Machine Translation.* COLING-08. 1097-1104.

Ying Zhang, Stephan Vogel, Alex Waibel. 2004. *Interpreting BLEU/NIST scores: How much improvement do we need to have a better system?* LREC-04

The corresponding authors of this paper are Hui Zhang (zhangh1982@gmail.com) and Min Zhang (mzhang@i2r.a-star.edu.sg)