A Dependency Treelet String Correspondence Model for Statistical Machine Translation

Deyi Xiong, Qun Liu and Shouxun Lin
Key Laboratory of Intelligent Information Processing
Institute of Computing Technology
Chinese Academy of Sciences
Beijing, China, 100080
{dyxiong, liuqun, sxlin}@ict.ac.cn

Abstract

This paper describes a novel model using dependency structures on the source side for syntax-based statistical machine translation: Dependency Treelet String Correspondence Model (DTSC). The DTSC model maps source dependency structures to target strings. In this model translation pairs of source treelets and target strings with their word alignments are learned automatically from the parsed and aligned corpus. The DTSC model allows source treelets and target strings with variables so that the model can generalize to handle dependency structures with the same head word but with different modifiers and arguments. Additionally, target strings can be also discontinuous by using gaps which are corresponding to the uncovered nodes which are not included in the source treelets. A chart-style decoding algorithm with two basic operations–substituting and attaching–is designed for the DTSC model. We argue that the DTSC model proposed here is capable of lexicalization, generalization, and handling discontinuous phrases which are very desirable for machine translation. We finally evaluate our current implementation of a simplified version of DTSC for statistical machine translation.

1 Introduction

Over the last several years, various statistical syntax-based models were proposed to extend traditional word/phrase based models in statistical machine translation (SMT) (Lin, 2004; Chiang, 2005; Ding et al., 2005; Quirk et al., 2005; Marcu et al., 2006; Liu et al., 2006). It is believed that these models can improve the quality of SMT significantly. Compared with phrase-based models, syntax-based models lead to better reordering and higher flexibility by introducing hierarchical structures and variables which make syntax-based models capable of hierarchical reordering and generalization. Due to these advantages, syntax-based approaches are becoming an active area of research in machine translation.

In this paper, we propose a novel model based on dependency structures: Dependency Treelet String Correspondence Model (DTSC). The DTSC model maps source dependency structures to target strings. It just needs a source language parser. In contrast to the work by Lin (2004) and by Quirk et al. (2005), the DTSC model does not need to generate target language dependency structures using source structures and word alignments. On the source side, we extract treelets which are any connected subgraphs and consistent with word alignments. While on the target side, we allow the aligned target sequences to be generalized and discontinuous by introducing variables and gaps. The variables on the target side are aligned to the corresponding variables of treelets, while gaps between words or variables are corresponding to the uncovered nodes which are not included by treelets. To complete the translation process, we design two basic operations for the decoding: substituting and attaching. Substituting is used to replace variable nodes which have been already translated, while attaching is used to attach uncov-

Proceedings of the Second Workshop on Statistical Machine Translation, pages 40–47,
Prague, June 2007.

ered nodes to treelets.

In the remainder of the paper, we first define dependency treelet string correspondence in section 2 and describe an algorithm for extracting DTSCs from the parsed and word-aligned corpus in section 3. Then we build our model based on DTSC in section 4. The decoding algorithm and related pruning strategies are introduced in section 5. We also specify the strategy to integrate phrases into our model in section 6. In section 7 we evaluate our current implementation of a simplified version of DTSC for statistical machine translation. And finally, we discuss related work and conclude.

2 Dependency Treelet String Correspondence

A dependency treelet string correspondence π is a triple $< D, S, A >$ which describes a translation pair $< D, S >$ and their alignment A, where D is the dependency treelet on the source side and S is the translation string on the target side. $< D, S >$ must be consistent with the word alignment M of the corresponding sentence pair

$$\forall(i, j) \in M, i \in D \leftrightarrow j \in S$$

A **treelet** is defined to be any connected subgraph, which is similar to the definition in (Quirk et al., 2005). Treelet is more representatively flexible than subtree which is widely used in models based on phrase structures (Marcu et al., 2006; Liu et al., 2006). The most important distinction between the treelet in (Quirk et al., 2005) and ours is that we allow variables at positions of subnodes. In our definition, the root node must be lexicalized but the subnodes can be replaced with a wild card. The target counterpart of a wildcard node in S is also replaced with a wild card. The wildcards introduced in this way generalize DTSC to match dependency structures with the same head word but with different modifiers or arguments.

Another unique feature of our DTSC is that we allow target strings with gaps between words or wildcards. Since source treelets may not cover all subnodes, the uncovered subnodes will generate a gap as its counterpart on the target side. A sequence of continuous gaps will be merged to be one gap and gaps at the beginning and the end of S will be removed automatically.

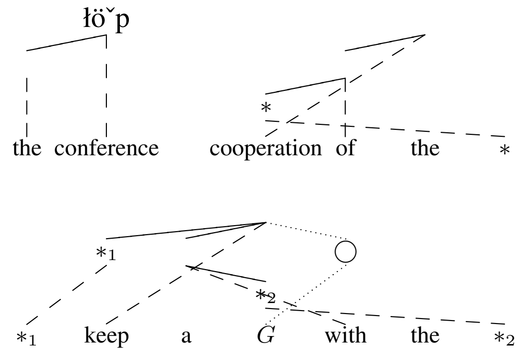

Figure 1: DTSC examples. Note that $*$ represents variable and G represents gap.

Gap can be considered as a special kind of variable whose counterpart on the source side is not present. This makes the model more flexible to match more partial dependency structures on the source side. If only variables can be used, the model has to match subtrees rather than treelets on the source side. Furthermore, the positions of variables on the target side are fixed so that some reorderings related with them can be recorded in DTSC. The positions of gaps on the target side, however, are not fixed until decoding. The presence of one gap and its position can not be finalized until attaching operation is performed. The introduction of gaps and the related attaching operation in decoding is the most important distinction between our model and the previous syntax-based models.

Figure 1 shows several different DTSCs automatically extracted from our training corpus. The top left DTSC is totally lexicalized, while the top right DTSC has one variable and the bottom has two variables and one gap. In the bottom DTSC, note that the node ◯ which is aligned to the gap G of the target string is an uncovered node and therefore not included in the treelet actually. Here we just want to show there is an uncovered node aligned with the gap G.

Each node at the source treelet has three attributes

1. The head word
2. The category, i.e. the part of speech of the head word
3. The node order which specifies the local order of the current node relative to its parent node.

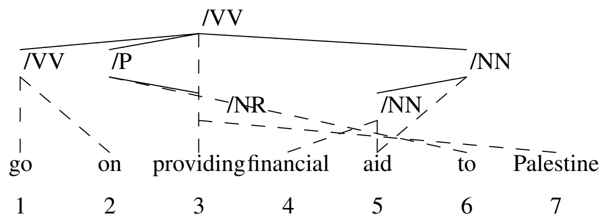

Figure 2: An example dependency tree and its alignments

Note that the node order is defined at the context of the extracted treelets but not the context of the original tree. For example, the attributes for the node ˇÐ in the bottom DTSC of Figure 1 are { , P, -1}. For two treelets, if and only if their structures are identical and each corresponding nodes share the same attributes, we say they are **matched**.

3 Extracting DTSCs

To extract DTSCs from the training corpus, firstly the corpus must be parsed on the source side and aligned at the word level. The source structures produced by the parser are unlabelled, ordered dependency trees with each word annotated with a part-of-speech. Figure 2 shows an example of dependency tree really used in our extractor.

When the source language dependency trees and word alignments between source and target languages are obtained, the DTSC extraction algorithm runs in two phases along the dependency trees and alignments. In the first step, the extractor annotates each node with specific attributes defined in section 3.1. These attributes are used in the second step which extracts all possible DTSCs rooted at each node recursively.

3.1 Node annotation

For each source dependency node n, we define three attributes: **word span**, **node span** and **crossed**. **Word span** is defined to be the target word sequence aligned with the head word of n, while **node span** is defined to be the closure of the union of node spans of all subnodes of n and its word span. These two attributes are similar to those introduced by Lin (Lin, 2004). The third attribute **crossed** is an indicator that has binary values. If the node span of n overlaps the word span of its parent node or the node span of its siblings, the **crossed** indicator of n is 1 and n is therefore a crossed node, otherwise the **crossed** indicator is 0 and n is a non-crossed node. Only non-crossed nodes can generate DTSCs because the target word sequence aligned with the whole subtree rooted at it does not overlap any other sequences and therefore can be extracted independently.

For the dependency tree and its alignments shown in Figure 2, only the node ⁄ıŸ is a crossed node since its node span ([4,5]) overlaps the word span ([5,5]) of its parent node ˇñıù .

3.2 DTSCs extraction

The DTSC extraction algorithm (shown in Figure 3) runs recursively. For each non-crossed node, the algorithm generates all possible DTSCs rooted at it by combining DTSCs from some subsets of its direct subnodes. If one subnode n selected in the combination is a crossed node, all other nodes whose word/node spans overlap the node span of n must be also selected in this combination. This kind of combination is defined to be consistent with the word alignment because the DTSC generated by this combination is consistent with the word alignment. All DTSCs generated in this way will be returned to the last call and outputted. For each crossed node, the algorithm generates pseudo DTSCs[1] using DTSCs from all of its subnodes. These pseudo DTSCs will be returned to the last call but not outputted.

During the combination of DTSCs from subnodes into larger DTSCs, there are two major tasks. One task is to generate the treelet using treelets from subnodes and the current node. This is a basic tree generation operation. It is worth mentioning that some non-crossed nodes are to be replaced with a wild card so the algorithm can learn generalized DTSCs described in section 2. Currently, we replace any non-crossed node alone or together with their sibling non-crossed nodes. The second task is to combine target strings. The word sequences aligned with uncovered nodes will be replaced with a gap. The word sequences aligned with wildcard nodes will be replaced with a wild card.

If a non-crossed node n has m direct subnodes, all 2^m combinations will be considered. This will generate a very large number of DTSCs, which is

[1]Some words in the target string are aligned with nodes which are not included in the source treelet.

$DTSCExtractor(Dnode\ \ n)$
$\Re := \emptyset$ (DTSC container of n)
for each subnode k of n **do**
  $R := DTSCExtractor(k)$
  $L := L \bigcup R$
end for
if $n.crossed! = 1$ and there are no subnodes whose span overlaps the word span of n **then**
  Create a DTSC $\pi =< D, S, A >$ where the dependency treelet D only contains the node n (not including any children of it)
  output π
  for each combination c of n's subnodes **do**
    if c is consistent with the word alignment **then**
      Generate all DTSCs R by combining DTSCs (L) from the selected subnodes with the current node n
      $\Re := \Re \bigcup R$
    end if
  end for
  output $\Re$
  return $\Re$
else if $n.crossed == 1$ **then**
  Create pseudo DTSCs P by combining all DTSCs from n's all subnodes.
  $\Re := \Re \bigcup P$
  return $\Re$
end if

Figure 3: DTSC Extraction Algorithm.

undesirable for training and decoding. Therefore we filter DTSCs according to the following restrictions

1. If the number of direct subnodes of node n is larger than 6, we only consider combining one single subnode with n each time because in this case reorderings of subnodes are always monotone.

2. On the source side, the number of direct subnodes of each node is limited to be no greater than *ary-limit*; the height of treelet D is limited to be no greater than *depth-limit*.

3. On the target side, the length of S (including gaps and variables) is limited to be no greater than *len-limit*; the number of gaps in S is limited to be no greater than *gap-limit*.

4. During DTSC combination, the DTSCs from each subnode are sorted by size (in descending order). Only the top *comb-limit* DTSCs will be selected to generate larger DTSCs.

As an example, for the dependency tree and its alignments in Figure 2, all DTSCs extracted by the algorithm with parameters { *ary-limit* = 2, *depth-limit* = 2, *len-limit* = 3, *gap-limit* = 1, *comb-limit* = 20 } are shown in the table 1.

Treelet	String
(¸?žÃ/VV/0)	go on
(flØ˚À ®Ž/NR/0)	Palestine
(ž_/P/0)	to
(ž_/P/0 (flØ˚À ®Ž/NR/1))	to Palestine
(ž_/P/0 ($*$/1))	to $*$
(˘ñıù /NN/0 (/ıŸ /NN/-1))	financial aid
(Ž4łfi/VV/0)	providing
(Ž4łfi/VV/0 ($*$/1))	providing $*$
(Ž4łfi/VV/0 ($*$/-1))	providing G $*$
(Ž4łfi/VV/0 (¸?žÃ/VV/-1))	go on providing
(Ž4łfi/VV/0 ($*$/-1))	$*$ providing
(Ž4łfi/VV/0 ($*_1$/-1) ($*_2$/1))	providing $*_2$ $*_1$
(Ž4łfi/VV/0 ($*_1$/-1) ($*_2$/1))	$*_1$ providing $*_2$

Table 1: Examples of DTSCs extracted from Figure 2. Alignments are not shown here because they are self-evident.

4 The Model

Given an input dependency tree, the decoder generates translations for each dependency node in bottom-up order. For each node, our algorithm will search all **matched** DTSCs automatically learned from the training corpus by the way mentioned in section 3. When the root node is traversed, the translating is finished. This complicated procedure involves a large number of sequences of applications of DTSC rules. Each sequence of applications of DTSC rules can derive a translation.

We define a derivation δ as a sequence of applications of DTSC rules, and let $c(\delta)$ and $e(\delta)$ be the source dependency tree and the target yield of δ, respectively. The score of δ is defined to be the product of the score of the DTSC rules used in the translation, and timed by other feature functions:

$$\S(\delta) = \prod_i \S(i) \cdot p_{lm}(e)^{\lambda}_{lm} \cdot exp(-\lambda_{ap} A(\delta)) \quad (1)$$

where $\S(i)$ is the score of the ith application of DTSC rules, $p_{lm}(e)$ is the language model score, and $exp(-\lambda_{ap} A(\delta))$ is the attachment penalty, where $A(\delta)$ calculates the total number of attachments occurring in the derivation δ. The attachment penalty gives some control over the selection of DTSC rules which makes the model prefer rules with more nodes covered and therefore less attaching operations involved.

For the score of DTSC rule π, we define it as follows:

$$\S(\pi) = \prod_j f_j(\pi)^{\lambda_j} \quad (2)$$

where the f_j are feature functions defined on DTSC rules. Currently, we used features proved to be effective in phrase-based SMT, which are:

1. The translation probability $p(D|S)$.
2. The inverse translation probability $p(S|D)$.
3. The lexical translation probability $p_{lex}(D|S)$ which is computed over the words that occur on the source and target sides of a DTSC rule by the IBM model 1.
4. The inverse lexical translation probability $p_{lex}(S|D)$ which is computed over the words that occur on the source and target sides of a DTSC rule by the IBM model 1.
5. The word penalty wp.
6. The DTSC penalty dp which allows the model to favor longer or shorter derivations.

It is worth mentioning how to integrate the N-gram language mode into our DTSC model. During decoding, we have to encounter many partial translations with gaps and variables. For these translations, firstly we only calculate the language model scores for word sequences in the translations. Later we update the scores when gaps are removed or specified by attachments or variables are substituted. Each updating involves merging two neighbor substrings s_l (left) and s_r (right) into one bigger string s. Let the sequence of $n-1$ (n is the order of N-gram language model used) rightmost words of s_l be s_l^r and the sequence of $n-1$ leftmost words of s_r be s_r^l. we have:

$$\begin{aligned} LM(s) = LM(s_l) + LM(s_r) + LM(s_l^r s_r^l) \\ -LM(s_l^r) - LM(s_r^l) \end{aligned} \quad (3)$$

where LM is the logarithm of the language model probability. We only need to compute the increment of the language model score:

$$\triangle_{LM} = LM(s_l^r s_r^l) - LM(s_l^r) - LM(s_r^l) \quad (4)$$

for each node n of the input tree T, in bottom-up order **do**
  Get all **matched** DTSCs rooted at n
  for each **matched** DTSC π **do**
    for each wildcard node n^* in π **do**
      Substitute the corresponding wildcard on the target side with translations from the stack of n^*
    end for
    for each uncovered node $n^@$ by π **do**
      Attach the translations from the stack of $n^@$ to the target side at the attaching point
    end for
  end for
end for

Figure 4: Chart-style Decoding Algorithm for the DTSC Model.

Melamed (2004) also used a similar way to integrate the language model.

5 Decoding

Our decoding algorithm is similar to the bottom-up chart parsing. The distinction is that the input is a tree rather than a string and therefore the chart is indexed by nodes of the tree rather than spans of the string. Also, several other tree-based decoding algorithms introduced by Eisner (2003), Quirk et al. (2005) and Liu et al. (2006) can be classified as the chart-style parsing algorithm too.

Our decoding algorithm is shown in Figure 4. Given an input dependency tree, firstly we generate the bottom-up order by postorder transversal. This order guarantees that any subnodes of node n have been translated before node n is done. For each node n in the bottom-up order, all **matched** DTSCs rooted at n are found, and a stack is also built for it to store the candidate translations. A DTSC π is said to **match** the input dependency subtree T rooted at n if and only if there is a treelet rooted at n that **matches** [2] the treelet of π on the source side.

For each matched DTSC π, two operations will be performed on it. The first one is **substituting** which replaces a wildcard node with the corresponding translated node. The second one is **attaching** which attaches an uncovered node to π. The two operations are shown in Figure 5. For each wildcard node n^*, translations from the stack of it will be selected to replace the corresponding wildcard on the

[2]The words, categories and orders of each corresponding nodes are matched. Please refer to the definition of **matched** in section 2.

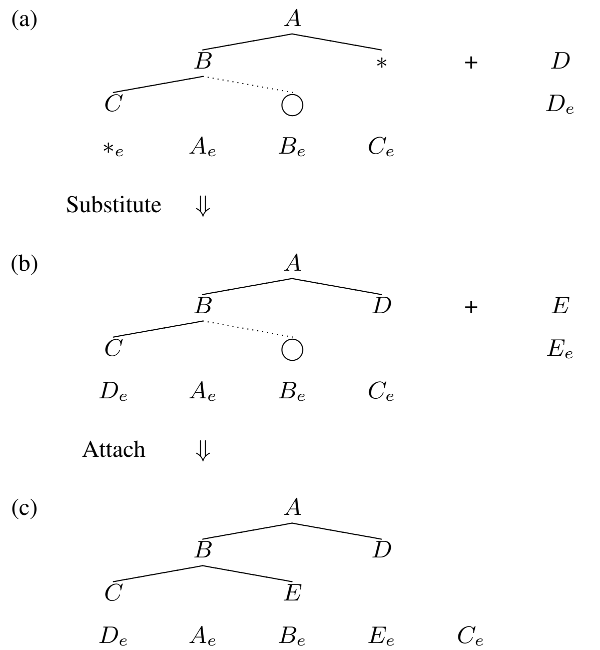

Figure 5: Substituting and attaching operations for decoding. X_e is the translation of X. Node that $*$ is a wildcard node to be substituted and node ◯ is an uncovered node to be attached.

target side and the scores of new translations will be calculated according to our model. For each uncovered node $n^@$, firstly we determine where translations from the stack of $n^@$ should be attached on the target side. There are several different mechanisms for choosing attaching points. Currently, we implement a heuristic way: on the source side, we find the node $n_p^@$ which is the nearest neighbor of $n^@$ from its parent and sibling nodes, then the attaching point is the left/right of the counterpart of $n_p^@$ on the target side according to their relative order. As an example, see the uncovered node ◯ in Figure 5. The nearest node to it is node B. Since node ◯ is at the right of node B, the attaching point is the right of B_e. One can search all possible points using an ordering model. And this ordering model can also use information from gaps on the target side. We believe this ordering model can improve the performance and let it be one of directions for our future research.

Note that the gaps on the target side are not necessarily attaching points in our current attaching mechanism. If they are not attaching point, they will be removed automatically.

The search space of the decoding algorithm is very large, therefore some pruning techniques have to be used. To speed up the decoder, the following pruning strategies are adopted.

1. **Stack pruning.** We use three pruning ways. The first one is recombination which converts the search to dynamic programming. When two translations in the same stack have the same w leftmost/rightmost words, where w depends on the order of the language model, they will be recombined by discarding the translation with lower score. The second one is the threshold pruning which discards translations that have a score worse than *stack-threshold* times the best score in the same stack. The last one is the histogram pruning which only keeps the top *stack-limit* best translations for each stack.

2. **Node pruning.** For each node, we only keep the top *node-limit* matched DTSCs rooted at that node, as ranked by the size of source treelets.

3. **Operation pruning.** For each operation, substituting and attaching, the decoding will generate a large number of partial translations[3] for the current node. We only keep the top *operation-limit* partial translations each time according to their scores.

6 Integrating Phrases

Although syntax-based models are good at dealing with hierarchical reordering, but at the local level, translating idioms and similar complicated expressions can be a problem. However, phrase-based models are good at dealing with these translations. Therefore, integrating phrases into the syntax-based models can improve the performance (Marcu et al., 2006; Liu et al., 2006). Since our DTSC model is based on dependency structures and lexicalized naturally, DTSCs are more similar to phrases than other translation units based on phrase structures. This means that phrases will be easier to be integrated into our model.

The way to integrate phrases is quite straightforward: if there is a treelet rooted at the current node,

[3]There are wildcard nodes or uncovered nodes to be handled.

of which the word sequence is continuous and identical to the source of some phrase, then a phrase-style DTSC will be generated which uses the target string of the phrase as its own target. The procedure is finished during decoding. In our experiments, integrating phrases improves the performance greatly.

7 Current Implementation

To test our idea, we implemented the dependency treelet string correspondence model in a Chinese-English machine translation system. The current implementation in this system is actually a simplified version of the DTSC model introduced above. In this version, we used a simple heuristic way for the operation of attaching rather than a sophisticated statistical model which can learn ordering information from the training corpus. Since dependency structures are more o flattened p compared with phrasal structures, there are many subnodes which will not be covered even by generalized matched DTSCs. This means the attaching operation is very common during decoding. Therefore better attaching model which calculates the best point for attaching , we believe, will improve the performance greatly and is a major goal for our future research.

To obtain the dependency structures of the source side, one can parse the source sentences with a dependency parser or parse them with a phrasal structure parser and then convert the phrasal structures into dependency structures. In our experiments we used a Chinese parser implemented by Xiong et al. (2005) which generates phrasal structures. The parser was trained on articles 1-270 of Penn Chinese Treebank version 1.0 and achieved 79.4% (F1 measure). We then converted the phrasal structure trees into dependency trees using the way introduced by Xia (1999).

To obtain the word alignments, we use the way of Koehn et al. (2005). After running GIZA++ (Och and Ney, 2000) in both directions, we apply the o grow-diag-final p refinement rule on the intersection alignments for each sentence pair.

The training corpus consists of 31, 149 sentence pairs with 823K Chinese words and 927K English words. For the language model, we used SRI Language Modeling Toolkit (Stolcke, 2002) to train a trigram model with modified Kneser-Ney smoothing on the 31, 149 English sentences. We selected 580 short sentences of length at most 50 characters from the 2002 NIST MT Evaluation test set as our development corpus and used it to tune λs by maximizing the BLEU score (Och, 2003), and used the 2005 NIST MT Evaluation test set as our test corpus.

Systems	BLEU-4
PB	20.88 ± 0.87
DTSC	20.20 ± 0.81
DTSC + phrases	21.46 ± 0.83

Table 2: BLEU-4 scores for our system and a phrase-based system.

From the training corpus, we learned 2, 729, 964 distinct DTSCs with the configuration { *ary-limit* = 4, *depth-limit* = 4, *len-limit* = 15, *gap-limit* = 2, *comb-limit* = 20 }. Among them, 160,694 DTSCs are used for the test set. To run our decoder on the development and test set, we set *stack-thrshold* = 0.0001, *stack-limit* = 100, *node-limit* = 100, *operation-limit* = 20.

We also ran a phrase-based system (PB) with a distortion reordering model (Xiong et al., 2006) on the same corpus. The results are shown in table 2. For all BLEU scores, we also show the 95% confidence intervals computed using Zhang's significant tester (Zhang et al., 2004) which was modified to conform to NIST's definition of the BLEU brevity penalty. The BLEU score of our current system with the DTSC model is lower than that of the phrase-based system. However, with phrases integrated, the performance is improved greatly, and the new BLEU score is higher than that of the phrase-based SMT. This difference is significant according to Zhang's tester. This result can be improved further using a better parser (Quirk et al., 2006) or using a statistical attaching model.

8 Related Work

The DTSC model is different from previous work based on dependency grammars by Eisner (2003), Lin (2004), Quirk et al. (2005), Ding et al. (2005) since they all deduce dependency structures on the target side. Among them, the most similar work is (Quirk et al., 2005). But there are still several major differences beyond the one mentioned above. Our treelets allow variables at any non-crossed nodes and target strings allow gaps, which are not available in (Quirk et al., 2005). Our language model is calculated during decoding while Quirk's language model is computed after decoding because of the complexity of their decoding.

The DTSC model is also quite distinct from previous tree-string models by Marcu et al. (2006) and Liu et al. (2006). Firstly, their models are based on phrase structure grammars. Secondly, subtrees instead of treelets are extracted in their models. Thirdly, it seems to be more difficult to integrate phrases into their models. And finally, our model allow gaps on the target side, which is an advantage shared by (Melamed, 2004) and (Simard, 2005).

9 Conclusions and Future Work

We presented a novel syntax-based model using dependency trees on the source side–dependency treelet string correspondence model–for statistical machine translation. We described an algorithm to learn DTSCs automatically from the training corpus and a chart-style algorithm for decoding.

Currently, we implemented a simple version of the DTSC model. We believe that our performance can be improved greatly using a more sophisticated mechanism for determining attaching points. Therefore the most important future work should be to design a better attaching model. Furthermore, we plan to use larger corpora for training and n-best dependency trees for decoding, which both are helpful for the improvement of translation quality.

Acknowledgements

This work was supported by National Natural Science Foundation of China, Contract No. 60603095 and 60573188.

References

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of ACL*.

Yuan Ding and Martha Palmer. 2005. Machine Translation Using Probabilistic Synchronous Dependency Insertion Grammars. In *Proceedings of ACL*.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of ACL*.

Philipp Koehn, Amittai Axelrod, Alexandra Birch Mayne, Chris Callison-Burch, Miles Osborne and David Talbot. 2005. Edinburgh System Description for the 2005 IWSLT Speech Translation Evaluation. In *International Workshop on Spoken Language Translation*.

Daniel Marcu, Wei Wang, Abdessamad Echihabi, and Kevin Knight. 2006. SPMT: Statistical Machine Translation with Syntactified Target Language Phraases. In *Proceedings of EMNLP*.

I. Dan Melamed. 2004. Algorithms for Syntax-Aware Statistical Machine Translation. In *Proceedings of the Conference on Theoretical and Methodological Issues in Machine Translation (TMI)*, Baltimore, MD.

Dekang Lin. 2004. A path-based transfer model for machine translation. In *Proceedings of COLING*.

Yang Liu, Qun Liu, and Shouxun Lin. 2006. Tree-to-String Alignment Template for Statistical Machine Translation. In *Proceedings of ACL*.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*.

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

Chris Quirk, Arul Menezes and Colin Cherry. 2005. Dependency Treelet Translation: Syntactically Informed Phrasal SMT. In *Proceedings of ACL*.

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

Michel Simard, Nicola Cancedda, Bruno Cavestro, Marc Dymetman, Eric Gaussier, Cyril Goutte, Kenji Yamada. 2005. Translating with non-contiguous phrases. In *Proceedings of HLT-EMNLP*.

Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *Proceedings of International Conference on Spoken Language Processing*, volume 2, pages 901-904.

Fei Xia. 1999. Automatic Grammar Generation from Two Different Perspectives. PhD thesis, University of Pennsylvania.

Deyi Xiong, Qun Liu, and Shouxun Lin. 2006. Maximum Entropy Based Phrase Reordering Model for Statistical Machine Translation. In *Proceedings of COLING-ACL*, Sydney, Australia.

Deyi Xiong, Shuanglong Li, Qun Liu, Shouxun Lin, Yueliang Qian. 2005. Parsing the Penn Chinese Treebank with Semantic Knowledge. In *Proceedings of IJCNLP*, Jeju Island, Korea.

Ying Zhang, Stephan Vogel, and Alex Waibel. 2004. Interpreting BLEU/NIST scores: How much improvement do we need to have a better system? In *Proceedings of LREC*, pages 2051&¶2054.