A New String-to-Dependency Machine Translation Algorithm with a Target Dependency Language Model

Libin Shen
BBN Technologies
Cambridge, MA 02138, USA
lshen@bbn.com

Jinxi Xu
BBN Technologies
Cambridge, MA 02138, USA
jxu@bbn.com

Ralph Weischedel
BBN Technologies
Cambridge, MA 02138, USA
weisched@bbn.com

Abstract

In this paper, we propose a novel string-to-dependency algorithm for statistical machine translation. With this new framework, we employ a target dependency language model during decoding to exploit long distance word relations, which are unavailable with a traditional n-gram language model. Our experiments show that the string-to-dependency decoder achieves 1.48 point improvement in BLEU and 2.53 point improvement in TER compared to a standard hierarchical string-to-string system on the NIST 04 Chinese-English evaluation set.

1 Introduction

In recent years, hierarchical methods have been successfully applied to Statistical Machine Translation (Graehl and Knight, 2004; Chiang, 2005; Ding and Palmer, 2005; Quirk et al., 2005). In some language pairs, i.e. Chinese-to-English translation, state-of-the-art hierarchical systems show significant advantage over phrasal systems in MT accuracy. For example, Chiang (2007) showed that the Hiero system achieved about 1 to 3 point improvement in BLEU on the NIST 03/04/05 Chinese-English evaluation sets compared to a start-of-the-art phrasal system.

Our work extends the hierarchical MT approach. We propose a string-to-dependency model for MT, which employs rules that represent the source side as strings and the target side as dependency structures. We restrict the target side to the so called *well-formed* dependency structures, in order to cover a large set of non-constituent transfer rules (Marcu et al., 2006), and enable efficient decoding through dynamic programming. We incorporate a dependency language model during decoding, in order to exploit long-distance word relations which are unavailable with a traditional n-gram language model on target strings.

For comparison purposes, we replicated the Hiero decoder (Chiang, 2005) as our baseline. Our string-to-dependency decoder shows 1.48 point improvement in BLEU and 2.53 point improvement in TER on the NIST 04 Chinese-English MT evaluation set.

In the rest of this section, we will briefly discuss previous work on hierarchical MT and dependency representations, which motivated our research. In section 2, we introduce the model of string-to-dependency decoding. Section 3 illustrates of the use of dependency language models. In section 4, we describe the implementation details of our MT system. We discuss experimental results in section 5, compare to related work in section 6, and draw conclusions in section 7.

1.1 Hierarchical Machine Translation

Graehl and Knight (2004) proposed the use of target-tree-to-source-string transducers (*xRS*) to model translation. In *xRS* rules, the right-hand-side(*rhs*) of the target side is a tree with non-terminals(NTs), while the *rhs* of the source side is a string with NTs. Galley et al. (2006) extended this string-to-tree model by using Context-Free parse trees to represent the target side. A tree could represent multi-level transfer rules.

The Hiero decoder (Chiang, 2007) does not require explicit syntactic representation on either side of the rules. Both source and target are strings with *NTs*. Decoding is solved as chart parsing. Hiero can be viewed as a hierarchical string-to-string model.

Ding and Palmer (2005) and Quirk et al. (2005)

Proceedings of ACL-08: HLT, pages 577–585,
Columbus, Ohio, USA, June 2008.

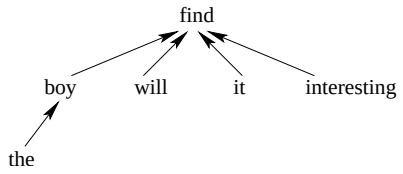

Figure 1: The dependency tree for sentence *the boy will find it interesting*

followed the tree-to-tree approach (Shieber and Schabes, 1990) for translation. In their models, dependency treelets are used to represent both the source and the target sides. Decoding is implemented as tree transduction preceded by source side dependency parsing. While tree-to-tree models can represent richer structural information, existing tree-to-tree models did not show advantage over string-to-tree models on translation accuracy due to a much larger search space.

One of the motivations of our work is to achieve desirable trade-off between model capability and search space through the use of the so called *well-formed* dependency structures in rule representation.

1.2 Dependency Trees

Dependency trees reveal long-distance relations between words. For a given sentence, each word has a parent word which it depends on, except for the *root* word.

Figure 1 shows an example of a dependency tree. Arrows point from the child to the parent. In this example, the word *find* is the root.

Dependency trees are simpler in form than CFG trees since there are no constituent labels. However, dependency relations directly model semantic structure of a sentence. As such, dependency trees are a desirable prior model of the target sentence.

1.3 Motivations for Well-Formed Dependency Structures

We restrict ourselves to the so-called *well-formed* target dependency structures based on the following considerations.

Dynamic Programming

In (Ding and Palmer, 2005; Quirk et al., 2005), there is no restriction on dependency treelets used in transfer rules except for the size limit. This may result in a high dimensionality in hypothesis representation and make it hard to employ shared structures for efficient dynamic programming.

In (Galley et al., 2004), rules contain NT slots and combination is only allowed at those slots. Therefore, the search space becomes much smaller. Furthermore, shared structures can be easily defined based on the labels of the slots.

In order to take advantage of dynamic programming, we fixed the positions onto which another another tree could be attached by specifying NTs in dependency trees.

Rule Coverage

Marcu et al. (2006) showed that many useful phrasal rules cannot be represented as hierarchical rules with the existing representation methods, even with composed transfer rules (Galley et al., 2006). For example, the following rule

- $<$(hong)$_{Chinese}$, (DT(the) JJ(red))$_{English}>$

is not a valid string-to-tree transfer rule since *the red* is a partial constituent.

A number of techniques have been proposed to improve rule coverage. (Marcu et al., 2006) and (Galley et al., 2006) introduced artificial constituent nodes dominating the phrase of interest. The binarization method used by Wang et al. (2007) can cover many non-constituent rules also, but not all of them. For example, it cannot handle the above example. DeNeefe et al. (2007) showed that the best results were obtained by combing these methods.

In this paper, we use *well-formed* dependency structures to handle the coverage of non-constituent rules. The use of dependency structures is due to the flexibility of dependency trees as a representation method which does not rely on constituents (Fox, 2002; Ding and Palmer, 2005; Quirk et al., 2005). The *well-formedness* of the dependency structures enables efficient decoding through dynamic programming.

2 String-to-Dependency Translation

2.1 Transfer Rules with Well-Formed Dependency Structures

A string-to-dependency grammar G is a 4-tuple $G =< \mathcal{R}, X, T_f, T_e >$, where $\mathcal{R}$ is a set of transfer rules. X is the only non-terminal, which is similar to the Hiero system (Chiang, 2007). T_f is a set of

terminals in the source language, and T_e is a set of terminals in the target language[1].

A string-to-dependency transfer rule $R \in \mathcal{R}$ is a 4-tuple $R =< S_f, S_e, D, A >$, where $S_f \in (T_f \cup \{X\})^+$ is a source string, $S_e \in (T_e \cup \{X\})^+$ is a target string, D represents the dependency structure for S_e, and A is the alignment between S_f and S_e. Non-terminal alignments in A must be one-to-one.

In order to exclude undesirable structures, we only allow S_e whose dependency structure D is *well-formed*, which we will define below. In addition, the same well-formedness requirement will be applied to partial decoding results. Thus, we will be able to employ shared structures to merge multiple partial results.

Based on the results in previous work (DeNeefe et al., 2007), we want to keep two kinds of dependency structures. In one kind, we keep dependency trees with a sub-root, where all the children of the sub-root are complete. We call them **fixed** dependency structures because the head is known or fixed. In the other, we keep dependency structures of sibling nodes of a common head, but the head itself is unspecified or floating. Each of the siblings must be a complete constituent. We call them **floating** dependency structures. Floating structures can represent many linguistically meaningful non-constituent structures: for example, like *the red*, a modifier of a noun. Only those two kinds of dependency structures are **well-formed** structures in our system.

Furthermore, we operate over *well-formed* structures in a bottom-up style in decoding. However, the description given above does not provide a clear definition on how to combine those two types of structures. In the rest of this section, we will provide formal definitions of *well-formed* structures and combinatory operations over them, so that we can easily manipulate *well-formed* structures in decoding. Formal definitions also allow us to easily extend the framework to incorporate a dependency language model in decoding. Examples will be provided along with the formal definitions.

Consider a sentence $S = w_1 w_2 ... w_n$. Let $d_1 d_2 ... d_n$ represent the parent word IDs for each word. For example, $d_4 = 2$ means that w_4 depends on w_2. If w_i is a root, we define $d_i = 0$.

[1]We ignore the left hand side here because there is only one non-terminal X. Of course, this formalism can be extended to have multiple NTs.

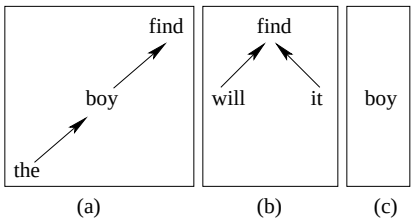

Figure 2: Fixed dependency structures

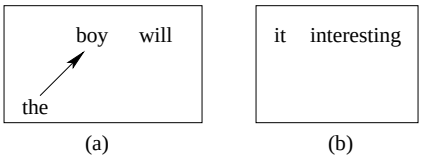

Figure 3: Floating dependency structures

Definition 1 *A dependency structure $d_{i..j}$ is* **fixed on head** *h, where $h \in [i, j]$, 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 $d_k \notin [i, j]$

In addition, we say the category of $d_{i..j}$ is $(-, h, -)$, where $-$ means this field is undefined.

Definition 2 *A dependency structure $d_i ... d_j$ 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 $k \notin C, d_k \in [i, j]$
- $\forall k \notin [i, j], d_k \notin [i, j]$

We say the category of $d_{i..j}$ is $(C, -, -)$ if $j < h$, or $(-, -, C)$ otherwise. A category is composed of the three fields (A, h, B), where h is used to represent the head, and A and B are designed to model left and right dependents of the head respectively.

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

Examples

We can represent dependency structures with graphs. Figure 2 shows examples of fixed structures, Figure 3 shows examples of floating structures, and Figure 4 shows ill-formed dependency structures.

It is easy to verify that the structures in Figures 2 and 3 are well-formed. 4(a) is ill-formed because

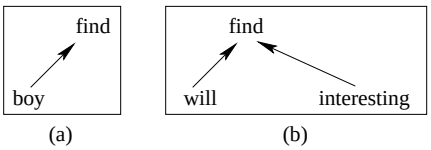

Figure 4: Ill-formed dependency structures

boy does not have its child word *the* in the tree. 4(b) is ill-formed because it is not a continuous segment.

As for the example *the red* mentioned above, it is a *well-formed* floating dependency structure.

2.2 Operations on Well-Formed Dependency Structures and Categories

One of the purposes of introducing *floating* dependency structures is that siblings having a common parent will become a well-defined entity, although they are not considered a constituent. We always build well-formed partial structures on the target side in decoding. Furthermore, we combine partial dependency structures in a way such that we can obtain all possible well-formed but no ill-formed dependency structures during bottom-up decoding.

The solution is to employ *categories* introduced above. Each well-formed dependency structure has a category. We can apply four combinatory operations over the categories. If we can combine two categories with a certain *category operation*, we can use a corresponding *tree operation* to combine two dependency structures. The category of the combined dependency structure is the result of the combinatory *category operations*.

We first introduce three meta category operations. Two of them are unary operations, *left raising* (LR) and *right raising* (RR), and one is the binary operation *unification* (UF).

First, the raising operations are used to turn a completed fixed structure into a floating structure. It is easy to verify the following theorem according to the definitions.

Theorem 1 *A fixed structure with category* $(-, h, -)$ *for span* $[i, j]$ *is also a floating structure with children* $\{h\}$ *if there are no outside words depending on word* h*.*

$$\forall k \notin [i, j], d_k \neq h. \quad (1)$$

Therefore we can always *raise* a fixed structure if we assume it is complete, i.e. (1) holds.

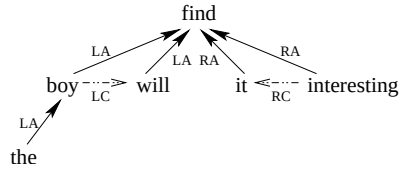

Figure 5: A dependency tree with flexible combination

Definition 3 *Meta Category Operations*

- $\text{LR}((-, h, -)) = (\{h\}, -, -)$
- $\text{RR}((-, h, -)) = (-, -, \{h\})$
- $\text{UF}((A_1, h_1, B_1), (A_2, h_2, B_2)) = \text{NORM}((A_1 \sqcup A_2, h_1 \sqcup h_2, B_1 \sqcup B_2))$

Unification is well-defined if and only if we can unify all three elements and the result is a valid fixed or floating category. For example, we can unify a fixed structure with a floating structure or two floating structures in the same direction, but we cannot unify two fixed structures.

$$h_1 \sqcup h_2 = \begin{cases} h_1 & \text{if } h_2 = - \\ h_2 & \text{if } h_1 = - \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$A_1 \sqcup A_2 = \begin{cases} A_1 & \text{if } A_2 = - \\ A_2 & \text{if } A_1 = - \\ A_1 \cup A_2 & \text{otherwise} \end{cases}$$

$$\text{NORM}((A, h, B)) = \begin{cases} (-, h, -) & \text{if } h \neq - \\ (A, -, -) & \text{if } h = -, B = - \\ (-, -, B) & \text{if } h = -, A = - \\ \text{undefined} & \text{otherwise} \end{cases}$$

Next we introduce the four tree operations on *dependency structures*. Instead of providing the formal definition, we use figures to illustrate these operations to make it easy to understand. Figure 1 shows a traditional dependency tree. Figure 5 shows the four operations to combine partial dependency structures, which are *left adjoining* (LA), *right adjoining* (RA), *left concatenation* (LC) and *right concatenation* (RC).

Child and parent subtrees can be combined with *adjoining* which is similar to the traditional dependency formalism. We can either adjoin a fixed structure or a floating structure to the head of a fixed structure.

Complete siblings can be combined via *concatenation*. We can concatenate two fixed structures, one fixed structure with one floating structure, or two floating structures in the same direction. The flexibility of the order of operation allows us to take ad-

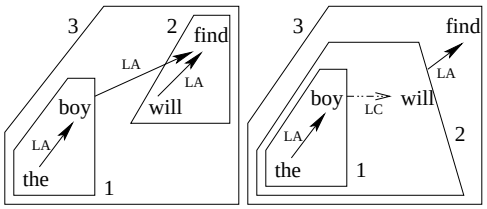

Figure 6: Operations over well-formed structures

vantage of various translation fragments encoded in transfer rules.

Figure 6 shows alternative ways of applying operations on *well-formed* structures to build larger structures in a bottom-up style. Numbers represent the order of operation.

We use the same names for the operations on categories for the sake of convenience. We can easily use the meta category operations to define the four combinatory operations. The definition of the operations in the left direction is as follows. Those in the right direction are similar.

Definition 4 *Combinatory category operations*

$$\begin{aligned}
& \text{LA}((A_1,-,-),(-,h_2,-)) \\
= \ & \text{UF}((A_1,-,-),(-,h_2,-)) \\
& \text{LA}((-,h_1,-),(-,h_2,-)) \\
= \ & \text{UF}(\text{LR}((-,h_1,-)),(-,h_2,-)) \\
& \text{LC}((A_1,-,-),(A_2,-,-)) \\
= \ & \text{UF}((A_1,-,-),(A_2,-,-)) \\
& \text{LC}((A_1,-,-),(-,h_2,-)) \\
= \ & \text{UF}((A_1,-,-),\text{LR}((-,h_2,-))) \\
& \text{LC}((-,h_1,-),(A_2,-,-)) \\
= \ & \text{UF}(\text{LR}((-,h_1,-)),(A_2,-,-)) \\
& \text{LC}((-,h_1,-),(-,h_2,-)) \\
= \ & \text{UF}(\text{LR}((-,h_1,-)),\text{LR}((-,h_2,-)))
\end{aligned}$$

It is easy to verify the *soundness* and *completeness* of category operations based on one-to-one mapping of the conditions in the definitions of corresponding operations on dependency structures and on categories.

Theorem 2 (soundness and completeness) *Suppose X and Y are well-formed dependency structures. OP(cat(X), cat(Y)) is well-defined for a given operation OP if and only if OP(X,Y) is well-defined. Furthermore,*

$$\text{cat}(\text{OP}(X,Y)) = \text{OP}(\text{cat}(X),\text{cat}(Y))$$

Suppose we have a dependency tree for *a red apple*, where both *a* and *red* depend on *apple*. There are two ways to compute the category of this string from the bottom up.

$$\begin{aligned}
& \text{cat}(D_{a_red_apple}) \\
= \ & \text{LA}(\text{cat}(D_a),\text{LA}(\text{cat}(D_{red}),\text{cat}(D_{apple}))) \\
= \ & \text{LA}(\text{LC}(\text{cat}(D_a),\text{cat}(D_{red})),\text{cat}(D_{apple}))
\end{aligned}$$

Based on Theorem 2, it follows that combinatory operation of categories has the *confluence* property, since the result dependency structure is determined.

Corollary 1 (confluence) *The category of a well-formed dependency tree does not depend on the order of category calculation.*

With categories, we can easily track the types of dependency structures and constrain operations in decoding. For example, we have a rule with dependency structure $find \leftarrow X$, where X *right adjoins* to $find$. Suppose we have two floating structures[2],

$$\begin{aligned}
\text{cat}(X_1) &= (\{he, will\},-,-) \\
\text{cat}(X_2) &= (-,-,\{it, interesting\})
\end{aligned}$$

We can replace X by X_2, but not by X_1 based on the definition of category operations.

2.3 Rule Extraction

Now we explain how we get the string-to-dependency rules from training data. The procedure is similar to (Chiang, 2007) except that we maintain tree structures on the target side, instead of strings.

Given sentence-aligned bi-lingual training data, we first use GIZA++ (Och and Ney, 2003) to generate word level alignment. We use a statistical CFG parser to parse the English side of the training data, and extract dependency trees with Magerman's rules (1995). Then we use heuristic rules to extract transfer rules recursively based on the GIZA alignment and the target dependency trees. The rule extraction procedure is as follows.

1. Initialization:
   All the 4-tuples $(P_f^{i,j}, P_e^{m,n}, D, A)$ are *valid phrase alignments*, where source phrase $P_f^{i,j}$ is

[2]Here we use words instead of word indexes in categories to make the example easy to understand.

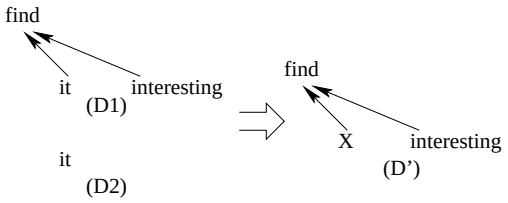

Figure 7: Replacing *it* with X in D_1

aligned to target phrase $P_e^{m,n}$ under alignment[3] A, and D, the dependency structure for $P_e^{m,n}$, is *well-formed*. All *valid phrase templates* are *valid rules templates*.

2. Inference:
   Let $(P_f^{i,j}, P_e^{m,n}, D_1, A)$ be a *valid rule template*, and $(P_f^{p,q}, P_e^{s,t}, D_2, A)$ a *valid phrase alignment*, where $[p,q] \subset [i,j]$, $[s,t] \subset [m,n]$, D_2 is a sub-structure of D_1, and at least one word in $P_f^{i,j}$ but not in $P_f^{p,q}$ is aligned.
   We create a new *valid rule template* (P'_f, P'_e, D', A), where we obtain P'_f by replacing $P_f^{p,q}$ with label X in $P_f^{i,j}$, and obtain P'_e by replacing $P_e^{s,t}$ with X in $P_e^{m,n}$. Furthermore, We obtain D' by replacing sub-structure D_2 with X in D_1[4]. An example is shown in Figure 7.

Among all valid rule templates, we collect those that contain at most two NTs and at most seven elements in the source as transfer rules in our system.

2.4 Decoding

Following previous work on hierarchical MT (Chiang, 2005; Galley et al., 2006), we solve decoding as chart parsing. We view target dependency as the hidden structure of source fragments.

The parser scans all source cells in a bottom-up style, and checks matched transfer rules according to the source side. Once there is a completed rule, we build a larger dependency structure by substituting component dependency structures for corresponding NTs in the target dependency structure of rules.

Hypothesis dependency structures are organized in a shared forest, or AND-OR structures. An AND-structure represents an application of a rule over component OR-structures, and an OR-structure represents a set of alternative AND-structures with the same *state*. A **state** means a n-tuple that characterizes the information that will be inquired by up-level AND-structures.

Supposing we use a traditional tri-gram language model in decoding, we need to specify the leftmost two words and the rightmost two words in a state. Since we only have a single NT X in the formalism described above, we do not need to add the NT label in states. However, we need to specify one of the three types of the dependency structure: fixed, floating on the left side, or floating on the right side. This information is encoded in the category of the dependency structure.

In the next section, we will explain how to extend categories and states to exploit a dependency language model during decoding.

3 Dependency Language Model

For the dependency tree in Figure 1, we calculate the probability of the tree as follows

$$\begin{aligned} Prob &= P_T(find) \\ &\times P_L(will|find\text{-as-head}) \\ &\times P_L(boy|will, find\text{-as-head}) \\ &\times P_L(the|boy\text{-as-head}) \\ &\times P_R(it|find\text{-as-head}) \\ &\times P_R(interesting|it, find\text{-as-head}) \end{aligned}$$

Here $P_T(x)$ is the probability that word x is the root of a dependency tree. P_L and P_R are left and right side generative probabilities respectively. Let w_h be the head, and $w_{L_1}w_{L_2}...w_{L_n}$ be the children on the left side from the nearest to the farthest. Suppose we use a tri-gram dependency LM,

$$\begin{aligned} & P_L(w_{L_1}w_{L_2}...w_{L_n}|w_h\text{-as-head}) \\ = \; & P_L(w_{L_1}|w_h\text{-as-head}) \\ & \times P_L(w_{L_2}|w_{L_1}, w_h\text{-as-head}) \\ & \times ... \times P_L(w_{L_n}|w_{L_{n-1}}, w_{L_{n-2}}) \end{aligned} \quad (2)$$

w_h-as-head represents w_h used as the head, and it is different from w_h in the dependency language model. The right side probability is similar.

In order to calculate the dependency language model score, or depLM score for short, on the fly for

[3] By $P_f^{i,j}$ aligned to $P_e^{m,n}$, we mean all words in $P_f^{i,j}$ are either aligned to words in $P_e^{m,n}$ or unaligned, and vice versa. Furthermore, at least one word in $P_f^{i,j}$ is aligned to a word in $P_e^{m,n}$.

[4] If D_2 is a $floating$ structure, we need to merge several dependency links into one.

partial hypotheses in a bottom-up decoding, we need to save more information in categories and states.

We use a 5-tuple (LF, LN, h, RN, RF) to represent the category of a dependency structure. h represents the head. LF and RF represent the farthest two children on the left and right sides respectively. Similarly, LN and RN represent the nearest two children on the left and right sides respectively. The three types of categories are as follows.

- fixed: $(LF, -, h, -, RF)$
- floating left: $(LF, LN, -, -, -)$
- floating right: $(-, -, -, RN, RF)$

Similar operations as described in Section 2.2 are used to keep track of the head and boundary child nodes which are then used to compute depLM scores in decoding. Due to the limit of space, we skip the details here.

4 Implementation Details

Features

1. Probability of the source side given the target side of a rule
2. Probability of the target side given the source side of a rule
3. Word alignment probability
4. Number of target words
5. Number of concatenation rules used
6. Language model score
7. Dependency language model score
8. Discount on ill-formed dependency structures

We have eight features in our system. The values of the first four features are accumulated on the rules used in a translation. Following (Chiang, 2005), we also use concatenation rules like $X \rightarrow XX$ for backup. The 5th feature counts the number of concatenation rules used in a translation. In our system, we allow substitutions of dependency structures with unmatched categories, but there is a discount for such substitutions.

Weight Optimization

We tune the weights with several rounds of decoding-optimization. Following (Och, 2003), the k-best results are accumulated as the input of the optimizer. Powell's method is used for optimization with 20 random starting points around the weight vector of the last iteration.

Rescoring

We rescore 1000-best translations (Huang and Chiang, 2005) by replacing the 3-gram LM score with the 5-gram LM score computed offline.

5 Experiments

We carried out experiments on three models.

- baseline: replication of the Hiero system.
- filtered: a string-to-string MT system as in baseline. However, we only keep the transfer rules whose target side can be generated by a well-formed dependency structure.
- str-dep: a string-to-dependency system with a dependency LM.

We take the replicated Hiero system as our baseline because it is the closest to our string-to-dependency model. They have similar rule extraction and decoding algorithms. Both systems use only one non-terminal label in rules. The major difference is in the representation of target structures. We use dependency structures instead of strings; thus, the comparison will show the contribution of using dependency information in decoding.

All models are tuned on BLEU (Papineni et al., 2001), and evaluated on both BLEU and Translation Error Rate (TER) (Snover et al., 2006) so that we could detect over-tuning on one metric.

We used part of the NIST 2006 Chinese-English large track data as well as some LDC corpora collected for the DARPA GALE program (LDC2005E83, LDC2006E34 and LDC2006G05) as our bilingual training data. It contains about 178M/191M words in source/target. Hierarchical rules were extracted from a subset which has about 35M/41M words[5], and the rest of the training data were used to extract phrasal rules as in (Och, 2003; Chiang, 2005). The English side of this subset was also used to train a 3-gram dependency LM. Traditional 3-gram and 5-gram LMs were trained on a corpus of 6G words composed of the LDC Gigaword corpus and text downloaded from Web (Bulyko et al., 2007). We tuned the weights on NIST MT05 and tested on MT04.

[5]It includes eight corpora: LDC2002E18, LDC2003E07, LDC2004T08_HK_News, LDC2005E83, LDC2005T06, LDC2005T10, LDC2006E34, and LDC2006G05

Model	#Rules
baseline	140M
filtered	26M
str-dep	27M

Table 1: Number of transfer rules

Model	BLEU%		TER%	
	lower	mixed	lower	mixed
Decoding (3-gram LM)				
baseline	38.18	35.77	58.91	56.60
filtered	37.92	35.48	57.80	55.43
str-dep	39.52	37.25	56.27	54.07
Rescoring (5-gram LM)				
baseline	40.53	38.26	56.35	54.15
filtered	40.49	38.26	55.57	53.47
str-dep	41.60	39.47	55.06	52.96

Table 2: BLEU and TER scores on the test set.

Table 1 shows the number of transfer rules extracted from the training data for the tuning and test sets. The constraint of well-formed dependency structures greatly reduced the size of the rule set. Although the rule size increased a little bit after incorporating dependency structures in rules, the size of string-to-dependency rule set is less than 20% of the baseline rule set size.

Table 2 shows the BLEU and TER scores on MT04. On decoding output, the string-to-dependency system achieved 1.48 point improvement in BLEU and 2.53 point improvement in TER compared to the baseline hierarchical string-to-string system. After 5-gram rescoring, it achieved 1.21 point improvement in BLEU and 1.19 improvement in TER. The *filtered* model does not show improvement on BLEU. The filtered string-to-string rules can be viewed the string projection of string-to-dependency rules. It means that just using dependency structure does not provide an improvement on performance. However, dependency structures allow the use of a dependency LM which gives rise to significant improvement.

6 Discussion

The *well-formed* dependency structures defined here are similar to the data structures in previous work on mono-lingual parsing (Eisner and Satta, 1999; McDonald et al., 2005). However, here we have *fixed* structures growing on both sides to exploit various translation fragments learned in the training data, while the operations in mono-lingual parsing were designed to avoid artificial ambiguity of derivation.

Charniak et al. (2003) described a two-step string-to-CFG-tree translation model which employed a syntax-based language model to select the best translation from a target parse forest built in the first step. Only translation probability $P(F|E)$ was employed in the construction of the target forest due to the complexity of the syntax-based LM. Since our dependency LM models structures over target words directly based on dependency trees, we can build a single-step system. This dependency LM can also be used in hierarchical MT systems using lexicalized CFG trees.

The use of a dependency LM in MT is similar to the use of a structured LM in ASR (Xu et al., 2002), which was also designed to exploit long-distance relations. The depLM is used in a bottom-up style, while SLM is employed in a left-to-right style.

7 Conclusions and Future Work

In this paper, we propose a novel string-to-dependency algorithm for statistical machine translation. For comparison purposes, we replicated the Hiero system as described in (Chiang, 2005). Our string-to-dependency system generates 80% fewer rules, and achieves 1.48 point improvement in BLEU and 2.53 point improvement in TER on the decoding output on the NIST 04 Chinese-English evaluation set.

Dependency structures provide a desirable platform to employ linguistic knowledge in MT. In the future, we will continue our research in this direction to carry out translation with deeper features, for example, propositional structures (Palmer et al., 2005). We believe that the *fixed* and *floating* structures proposed in this paper can be extended to model predicates and arguments.

Acknowledgments

This work was supported by DARPA/IPTO Contract No. HR0011-06-C-0022 under the GALE program. We are grateful to Roger Bock, Ivan Bulyko, Mike Kayser, John Makhoul, Spyros Matsoukas, Antti-Veikko Rosti, Rich Schwartz and Bing Zhang for their help in running the experiments and constructive comments to improve this paper.

References

I. Bulyko, S. Matsoukas, R. Schwartz, L. Nguyen, and J. Makhoul. 2007. Language model adaptation in machine translation from speech. In *Proceedings of the 32nd IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

E. Charniak, K. Knight, and K. Yamada. 2003. Syntax-based language models for statistical machine translation. In *Proceedings of MT Summit IX*.

D. Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics (ACL)*.

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

S. DeNeefe, K. Knight, W. Wang, and D. Marcu. 2007. What can syntax-based mt learn from phrase-based mt? In *Proceedings of the 2007 Conference of Empirical Methods in Natural Language Processing*.

Y. Ding and M. Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 541–548, Ann Arbor, Michigan, June.

J. Eisner and G. Satta. 1999. Effi cient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*.

H. Fox. 2002. Phrasal cohesion and statistical machine translation. In *Proceedings of the 2002 Conference of Empirical Methods in Natural Language Processing*.

M. Galley, M. Hopkins, K. Knight, and D. Marcu. 2004. What's in a translation rule? In *Proceedings of the 2004 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

M. Galley, J. Graehl, K. Knight, D. Marcu, S. DeNeefea, W. Wang, and I. Thayer. 2006. Scalable inference and training of context-rich syntactic models. In *COLING-ACL '06: Proceedings of 44th Annual Meeting of the Association for Computational Linguistics and 21st Int. Conf. on Computational Linguistics*.

J. Graehl and K. Knight. 2004. Training tree transducers. In *Proceedings of the 2004 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

L. Huang and D. Chiang. 2005. Better k-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies*.

D. Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*.

D. Marcu, W. Wang, A. Echihabi, and K. Knight. 2006. SPMT: Statistical machine translation with syntactifi ed target language phraases. In *Proceedings of the 2006 Conference of Empirical Methods in Natural Language Processing*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics (ACL)*.

F. J. Och and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1).

F. J. Och. 2003. Minimum error rate training for statistical machine translation. In Erhard W. Hinrichs and Dan Roth, editors, *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 160–167, Sapporo, Japan, July.

M. Palmer, D. Gildea, and P. Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1).

K. Papineni, S. Roukos, and T. Ward. 2001. Bleu: a method for automatic evaluation of machine translation. IBM Research Report, RC22176.

C. Quirk, A. Menezes, and C. Cherry. 2005. Dependency treelet translation: Syntactically informed phrasal SMT. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 271–279, Ann Arbor, Michigan, June.

S. Shieber and Y. Schabes. 1990. Synchronous tree adjoining grammars. In *Proceedings of COLING '90: The 13th Int. Conf. on Computational Linguistics*.

M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of Association for Machine Translation in the Americas*.

W. Wang, K. Knight, and D. Marcu. 2007. Binarizing syntax trees to improve syntax-based machine translation accuracy. In *Proceedings of the 2007 Conference of Empirical Methods in Natural Language Processing*.

P. Xu, C. Chelba, and F. Jelinek. 2002. A study on richer syntactic dependencies for structured language modeling. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*.