Improving a Statistical MT System with Automatically Learned Rewrite Patterns

Fei Xia
IBM T. J. Watson Research Center
P.O. Box 218, Yorktown Heights
NY 10598, USA
feixia@us.ibm.com

Michael McCord
IBM T. J. Watson Research Center
P.O. Box 218, Yorktown Heights
NY 10598, USA
mcmccord@us.ibm.com

Abstract

Current clump-based statistical MT systems have two limitations with respect to word ordering: First, they lack a mechanism for expressing and using generalization that accounts for reorderings of linguistic phrases. Second, the ordering of target words in such systems does not respect linguistic phrase boundaries. To address these limitations, we propose to use automatically learned rewrite patterns to preprocess the source sentences so that they have a word order similar to that of the target language. Our system is a hybrid one. The basic model is statistical, but we use broad-coverage rule-based parsers in two ways – during training for learning rewrite patterns, and at runtime for reordering the source sentences. Our experiments show 10% relative improvement in Bleu measure.

1 Introduction

We can contrast two major approaches to machine translation (MT), which we call here simply *statistical MT* and *syntax-based MT*.[1] A statistical MT (SMT) system (in our somewhat restricted sense) automatically learns all the model parameters from bilingual text. It does not need parsers, grammars, or a pre-existing translation lexicon. Recently the so-called *phrase-based* SMT models such as (Och et al., 1999; Marcu and Wong, 2002; Tillmann and Xia, 2003) have outperformed the word-based SMT models proposed by Brown et al. (1993). A *phrase* in such systems can be any contiguous text, and it is not necessarily a linguistic phrase (such as noun phrase). To avoid confusion, in this paper we shall call any contiguous piece of text a *clump*, and call a phrase-based SMT system a *clump-based system*. We reserve the name *phrase* for linguistic phrase only, as appears in parse trees.

The main advantage of clump-based systems over word-based systems is that the former can *memorize* the translation of large clumps, and therefore alleviate the problems of translation selection, function word insertion, and the ordering of target words within a clump. However, both kinds of systems lack a mechanism for expressing and using generalization that accounts for reorderings of linguistic phrases in the source and target languages.[2] Also, the ordering of clumps in clump-based systems does not respect linguistic phrase boundaries because the models are not aware of, and do not use, linguistic phrases.

In contrast, syntax-based systems such as LMT (McCord, 1989; McCord and Bernth, 1998) express generalizations explicitly; for instance, one can use the rewrite pattern "$Adj\ N \Rightarrow N\ Adj$ " to express the fact that the adjective modifies a noun from the left in the source language, but from the right in the target language. Another advantage of syntax-based systems is that the ordering of target words respect linguistic phrase boundaries, since the rewriting is performed with respect to a parse tree. However, syntax-based systems need parsers, translation lexicons and rewrite patterns, which often require much human effort to create.

Since the two kinds of systems have complementary strengths, a natural question is whether they can be combined to improve MT results. Och et al. (2004) applied various methods ranging from no syntax to deep syntax to rerank the topN candidates produced by a clump-based SMT system. Notably, their experimental results showed little improvement from syntax-based features.

In this paper, we outline our attempt at combining the merits of both types of MT systems. Instead of using syntax-based features in post-processing to select translations produced by a SMT, we use parsers and rewrite patterns to preprocess the source sentences so that the reordered source sentences have a word order similar to that of the target language. Such reordered source sentences, along with the original target sentences, are then sent to a clump-based SMT system (i.e., our baseline system) for training and testing. The rewrite patterns are learned from the same bilingual text. We found in an experiment for English-French that after such preprocessing, the translation performance improved by 10% in Bleu measure (Papineni et al., 2002).

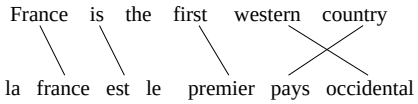
2 System Overview

In this section, we give a brief overview of the baseline SMT system and then outline a new approach.

[1] This dichotomy oversimplifies the situation, because there are so many possible hybrid versions of MT systems such as (Yamada and Knight, 2002).

[2] Alignment templates in (Och et al., 1999) can describe the reorderings of words or word classes, not the reorderings of linguistic phrases.

(a) word alignment



(b) some of the clump pairs

France => france, France => la france
is => est, first => premier
is the first => est le premier
first western country => premier pays occidental
western country => pays occidental

Figure 1: The training stage of the baseline system

2.1 The baseline SMT system

Our baseline system is a clump-based system as described in (Tillmann and Xia, 2003). During training, words in every sentence pair in the training data are aligned by a word aligner, and clump pairs are then extracted with some adjacency constraints. Clump pairs from all the sentence pairs plus their counts form a *clump dictionary*. Figure 1 shows an (English, French) sentence pair, the word alignment provided by a word aligner, and some of the extracted clump pairs.[3]

During decoding, the decoder segments the source sentence into multiple clumps, and translates each clump with the clump dictionary learned at the training time. All the possible translations are ranked using a scoring formula that contains a block unigram model and a word-based trigram language model, where a *block* is the same as a (source clump, target clump) pair. The translations with the highest scores are produced as system output. In order to account for the different word order in source and target language, the decoder allows source clumps to be translated in non-monotonic order.

Figure 2 illustrates the decoding stage of the system: (a) is a source sentence to be translated. (b) is a portion of the clump dictionary that is relevant to the sentence. Given such a dictionary, the decoder tries different ways to segment the source sentence. One possible segmentation is shown in (c). Next, the decoder replaces each source segment (i.e., clump) with the corresponding target clump(s) using the clump dictionary. Given the four source clumps in (c), the decoder could try all 4! possible permutations, resulting in different word order in the target side. Four of such translations are shown in (d), where the bars (|) mark clump boundaries. Among these translations, only the last one is correct.

This example demonstrates two limitations of the baseline system with respect to word ordering. First, the parse trees for the two source sentences in Figures 1 and 2 are identical except for the lexical

[3]One can use any word aligner that he prefers, and the aligner does not have to find links for all the words. For instance, the French words *la* and *le* (both meaning *the* in English) are not linked to any English word.

(a) sentence to be translated

He is the first international student

(b) part of the clump dictionary

is => est, the => le, first => premier
is the first => est le premier, he => il
student ⇒ étudiant
international => international

(c) one possible segmentation

He	is the first	international	student

(d) possible translations

il	est le premier	international	étudiant.
est le premier	il	international	étudiant.
il	international	est le premier	étudiant.
il	est le premier	étudiant	international.

Figure 2: The decoding stage of the baseline system

items at leaf nodes. Because the system does not have a mechanism to express the generation such as "$Adj_1\ Adj_2\ N \Rightarrow Adj_1\ N\ Adj_2$", it cannot benefit from the similarity of the parse trees. Second, when a clump (such as "*is the first*" in this example) is not a linguistic phrase, swapping it with other clumps often results in ungrammatical sentences, such as the second and third translations in Figure 2(d). Because the baseline system is not aware of linguistic phrases, it cannot avoid trying such ungrammatical translations. Not only does this practice tremendously slow down the system, but it also runs the risk that the language model might prefer such ungrammatical translations.

2.2 The new approach

To address these limitations, we propose to learn rewrite patterns from the training data, and then use them to reorder source sentences before decoding, so the decoder can translate source clumps in monotonic order. This approach has the following steps:

(T) At training time :

- **(T1)** Learn rewrite patterns: first parse sentences and align phrases, and then extract rewrite patterns.
- **(T2)** Reorder the source sentences using the rewrite patterns.
- **(T3)** Train the baseline system with the reordered source sentences and the original target sentences to get the clump dictionary.

(D) At decoding time :

- **(D1)** Reorder the test sentences with the rewrite patterns.

(D2) Translate the reordered sentences with the baseline decoder in *monotonic* order.

Among the five steps, Steps (T3) and (D2) are identical to the training step and the decoding step in the baseline system; Step (T2) is the same as Step (D1), except that one is on training data, the other is on test data. Therefore, we shall explain only Steps (T1) and (T2) in the next section.

This approach requires a parser for the source language, as the rewrite patterns are applied to source parse trees. A parser for the target language is optional, because, as we shall show in Section 3.2, the parse trees for target sentences are used only to provide an additional constraint for pattern extraction.

3 Learning and Applying Rewrite Patterns

There has been much work on learning rewrite patterns from parallel data, such as (Kaji et al., 1992; Matsumoto et al., 1993; Wu, 1996; Watanabe et al., 2000; Meyers et al., 2000; Lavoie et al., 2001; Menezes and Richardson, 2001) Most of these previous efforts use the learned patterns directly in a syntax-based MT system. In contrast, we use the patterns to preprocess the source sentences.

3.1 Definition of *rewrite patterns*

We define a rewrite pattern to be a quintuple:

(*SrcRule, TgtRule, SrcHeadPos, TgtHeadPos, ChildAlign*):

The five components are as follows:

(1) *SrcRule* is a context-free rule of the form

$$l(X) \rightarrow l(X_1) \ ... \ l(X_m),$$

where $l(X)$ is the label of a node X in a parse tree, and each X_i is a child of X in the parse tree. The set $\{X_i\}$ must include the head child of X.[4] The *label* $l(X)$ of a node X can be any information associated with X. For instance, it can be a syntactic tag (such as NP for noun phrase), a function tag (such as SBJ for subject), the head word of X, or any combination of these.

(2) *TgtRule* is of the same format as *SrcRule*, and let us represent it as $l(Y) \rightarrow l(Y_1) \ ... \ l(Y_n)$.

(3) *SrcHeadPos* is an integer giving the position of X's head child in the child sequence $X_1 \ ... \ X_m$.

(4) Similarly *TgtHeadPos* is the position of Y's head child in the child sequence $Y_1 \ ... \ Y_n$.

(5) *ChildAlign* is a correspondence between the $\{X_i\}$ and the $\{Y_j\}$. We require the correspondence to be injective, but not necessarily bijective, because of possible insertions and deletions of elements such as function words.

[4] We require the head child to be included in a rewrite pattern because quite often the ordering of dependents is with respect to the head child.

To simplify the notation in this paper, we drop the *SrcHeadPos*, *TgtHeadPos*, and *ChildAlign* from the notation if there is no confusion. For instance, the rewrite pattern

$$(NP \rightarrow Adj \ N, \quad NP \rightarrow N \ Adj, \\ 0, 1, \{0:1, 1:0\})$$

is now simply written as:

$$(NP \rightarrow Adj \ N) \ \Rightarrow \ (NP \rightarrow N \ Adj)$$

Because the parent label can often be predicted from the label of its head child, we may further drop the parent labels, and write the pattern as $Adj \ N \Rightarrow N \ Adj$.

We call a rewrite pattern *lexicalized* if at least one source node label X_i includes the head word. Lexicalized rewrite patterns are used to handle the cases where the ordering of a source child sequence in the target language depends on the head word of some child in the sequence. For instance, for most adjectives, "*Adj N*" in English becomes "*N Adj*" in French. This can be expressed as an unlexicalized pattern "$Adj \ N \Rightarrow N \ Adj$". However, for some adjectives such as *good*, "*Adj N*" in English often remains the same order in French. This can be expressed by a lexicalized pattern "Adj(good) $N \Rightarrow Adj$(bon) N".

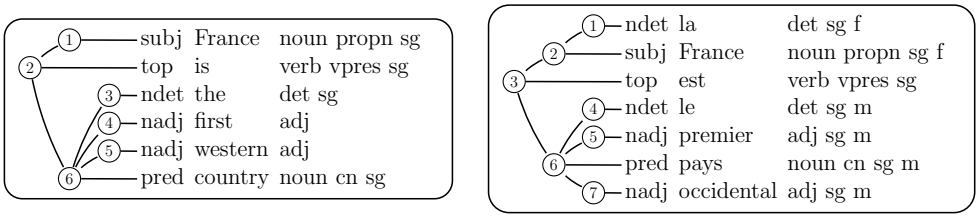
3.2 Learning rewrite patterns

The rewrite patterns are generated from bilingual text in four steps as described below.

3.2.1 Parsing the sentences

The first step is to parse source and target sentences, though, as mentioned before, a parser for the target language is optional to this work. In our experiments, we use Slot Grammar (SG) parsers. SG is a dependency-oriented system (McCord, 1980; McCord, 1990; McCord, 1993). Figure 3 shows simplified forms of the SG parse trees for the sentence pair in Figure 1. In this display form, there is one line per node (and per head word). On each line, one sees (1) the node label (which is usually the sentence word number of the head word), along with arcs to the node's (surface) daughters, (2) the head word, and (3) the features of the node. The first feature listed is always a part of speech. Of course an SG *noun phrase* is just a phrase whose POS feature is noun, and similarly for other parts of speech. The full SG parse data structures contain not only the surface information displayed here, but also deep structure information that includes word senses along with logical argument structure and remote relations.

Currently SGs exist for English, Spanish, French, Italian, Portuguese and German. The French SG parser, used here, is actually part of a single grammar for the four Romance languages, with switches for the differences, and this is developed by Esmé Manandise. There is an SG shell that contains many



Word alignment: (1→2, 2→3, 3→4, 4→5, 5→7, 6→6)
Phrase alignment: (1→2, 2→3, 3→4, 4→5, 5→7, 6→6)

Figure 3: Parse trees for source and target sentences, with word alignment and (equal) phrase alignment

language-universal rules and makes up a large percentage of the rules for any specific language grammar. Most of SG is "hand-coded", because of the emphasis on generalizations and language universals.

3.2.2 Aligning phrases

Previous work on phrase alignment such as (Matsumoto et al., 1993; Imamura, 2001) starts with word alignment and then aligns phrases using heuristic rules. We adopt a similar approach: We first align source and target words using a word aligner trained from the parallel data. Then for each source phrase S and each target phrase T, we calculate the percentage of words in the two phrases that are linked together using the formula in Eq. (1). Next we align S to the T with the highest $Score(S,T)$. Here $\#Links(S,T)$ is the total number of source words in S and target words in T that are linked together, and $Span(S)$ and $Span(T)$ are the numbers of words in S and T, respectively.

$$Score(S,T) = \frac{\#Links(S,T)}{Span(S) + Span(T)} \qquad (1)$$

For our SG displays, a word alignment can be shown as the list of pairs of words aligned, where each word is identified by the node number of the phrase headed by the word (which in turn is usually the word's position number in the sentence). A phrase alignment can be shown similarly, using phrase labels. In Figure 3, we see the word alignment provided by a word aligner, along with the resulting phrase alignment (which comes out with the same representation in this case).

3.2.3 Extracting rewrite patterns

Given a parse tree pair (SrcTree, TgtTree) and a phrase alignment Al, where Al is represented as a set of (source node, target node) pairs, we extract all the rewrite patterns $X_1 \ldots X_m \Rightarrow Y_1 \ldots Y_n$ that satisfy all of the following conditions:

- X_i are siblings in SrcTree, and their relative ordering in $X_1 \ldots X_m$ is the same as their ordering in SrcTree. Similar conditions hold for Y_i with respect to TgtTree.
- The parent node X of X_i aligns to the parent node Y of Y_i in Al. Each source child in $\{X_i\}$ must align to a unique target child in $\{Y_j\}$ and vice versa, except for children that dominate only function words.[5]
- $\{X_i\}$ must contain the head child of X in SrcTree, and $\{Y_j\}$ must contain the head child of Y in TgtTree, and the two head children are aligned in Al.
- For any aligned child pair (X_i, Y_j), they are either both lexicalized or both unlexicalized in the pattern.

Table 1: Some extracted rewrite patterns from the sentence pair in Figure 3

Unlexicalized rules:
1: $NP_0\ V\ NP_1 \Rightarrow NP_0\ V\ NP_1$
2: $N \Rightarrow Det\ N$
3: $Adj_1\ Adj_2\ N \Rightarrow Adj_1\ N\ Adj_2$
4: $Adj\ N \Rightarrow Adj\ N$
5: $Adj\ N \Rightarrow N\ Adj$
Lexicalized rules:
6: $Adj(first)\ N \Rightarrow Adj(premier)\ N$
7: $Adj\ N(country) \Rightarrow Adj\ N(pays)$
8: $Adj(western)\ N \Rightarrow N\ Adj(occidental)$
9: $Adj\ N(country) \Rightarrow N(pays)\ Adj$

The number of rewrite patterns extracted from a tree node pair is exponential with respect to the number of aligned children. To deal with that, we generate only patterns whose source rule length is no more than a threshold, which is set to five in our experiment. Table 1 lists a few patterns extracted from the sentence pair in Figure 3. The top five are unlexicalized, and the bottom four are lexicalized. Among them, Patterns 6 and 7 are lexicalized versions of Pattern 4; similarly, Patterns 8 and 9 are lexicalized version of Pattern 5. Some patterns (such as Patterns 4 and 5) are in conflict as they share the same source rule, but have different target rules. Such conflict is addressed next.

[5] This exception is meant to allow patterns such as $N \Rightarrow Det\ N$ (e.g., "*France*" in English becomes "*la France*" in French), where the French determiner does not align to any word on the English side.

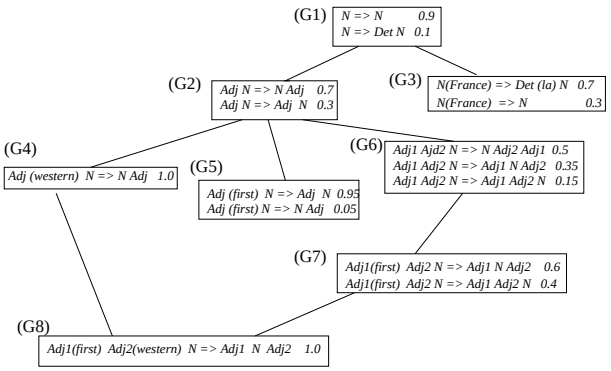


Figure 4: A sample hierarchy for pattern groups

3.2.4 Organizing rewrite patterns

After running the previous steps on training data and merging identical patterns, the number of resulting patterns can easily reach millions as many of the patterns are lexicalized. Furthermore, there will be many conflicting patterns. Therefore, the rewrite patterns need to be organized and filtered before they can be applied to source parse trees as in Steps (T2) and (D1). Existing filtering algorithms such as (Imamura et al., 2003) are not applicable here because rewrite patterns in our approach are not used directly for translation.

To apply a rewrite pattern to a source parse tree at a tree node, the source rule of the pattern has to match the children of the tree node. Therefore, patterns should be organized according to their source rules. Some source rules are clearly related. We say that a rule $X \Rightarrow X_1 \ldots X_m$ is more *specific* than another rule $X' \Rightarrow X'_1 \ldots X'_{m'}$ if the former rule is a lexicalized version of the latter rule, or if X and X' are the same, and $X_1 \ldots X_m$ is a superstring of $X'_1 \ldots X'_{m'}$. For instance, a lexicalized rule such as "$NP \rightarrow Adj(first)\ N$" is more specific than its unlexicalized version "$NP \rightarrow Adj\ N$", and the rule "$NP \rightarrow Det\ Adj\ N$" is more specific than the rule "$NP \rightarrow Adj\ N$". Given two rules such as "$NP \rightarrow Adj\ N$" and "$NP \rightarrow N\ PP$", neither is more specific than the other; therefore this specificity relation is not a total relation.

We organize patterns in two steps: First, we group patterns according to the source rules; that is, two patterns belong to the same group if and only if they have exactly the same source rule. We also normalize the counts for patterns in the same group. For each group, we find the pattern with the highest count, and call it *the default pattern* for the source rule. Second, for each group pair (G_i, G_j), we add a link from G_i to G_j if and only if the source rule for G_j is more specific than the source rule for G_i, and there is no other group coming between G_i and G_j. The resulting structure is a network, as a group can have more than one parent.

Figure 4 shows a small hierarchy, which has eight pattern groups.[6] The further away a group is from the root node of the hierarchy, the more specific its source rule is. We allow a group to "overwrite" the decisions made by its parent groups. For instance, given this hierarchy and a noun phrase of the form "*Adj N*" in the source language, if we know nothing else, we should apply the patterns in Group $G2$, which are likely to swap *Adj* and *N*. However, if we know that the head word of *Adj* is *first*, we should apply the patterns in Group $G5$ which are likely to keep the word order unchanged.

Now that we have put all the patterns into a hierarchy, we can quantify the usefulness of a pattern group G by measuring the degree of *agreement* between this G and G's parent groups. The more they agree, the less useful G is. The exact formula for measuring the "agreement" depends on exactly how rewrite patterns are used. Once a formula is selected, one can filter out patterns or pattern groups whose usefulness measure is below a threshold.

3.3 Applying rewrite patterns

There are many ways of applying rewrite patterns to parse trees. Currently we adopt the following greedy algorithm: given a parse tree $Tree$ and rewrite patterns stored in a hierarchy, we apply the patterns to $Tree$ and obtain a new tree $Tree'$ by visiting each internal node $Node$ of $Tree$ and applying the most specific applicable pattern to $Node$ to reorder the children of $Node$. Because at each internal node we apply at most one rewrite pattern and the pattern will only change the relative ordering of the node's children, traversal order is immaterial.

Given an internal node N in a parse tree, let us represent the node and its children N_i as a lexicalized context-free rule

$$N \rightarrow N_1(w_1)\ \ldots\ N_m(w_m)$$

where w_i is the head word of N_i. A pattern group is said to be *applicable* to the node N if the lexicalized context-free rule for N is identical to, or more specific than, the source rule of the pattern group. For instance, the lexicalized context-free rule for node 6 of the English tree in Figure 3 is "*NP → Det(the) Adj(first) Adj(western) N(country)*", which is more specific than the source rules for pattern group G1, G2, G6, and G7 in Figure 4, so all the four groups are applicable to node 6.

If there is more than one applicable group, we choose the one with the longest and most lexicalized source rule. Once we have selected the pattern group, we apply the default pattern in that group to the node. In this case, we apply the pattern "*Adj1(first) Adj2 N ⇒ Adj1 N Adj2*" in $G7$ to node 6. Similarly, we apply the pattern "*N(France) ⇒ Det(la) N(france)*" in Group $G3$ to node 2. The new parse tree after applying rewrite patterns is shown

[6] In this figure, we omit target node' head words unless the head word is a spontaneous word such as *la* in the first pattern of Group $G3$.

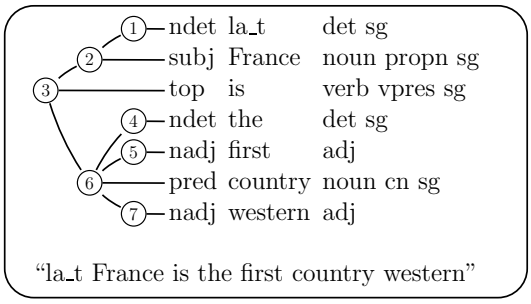


Figure 5: Results of applying rewrite patterns

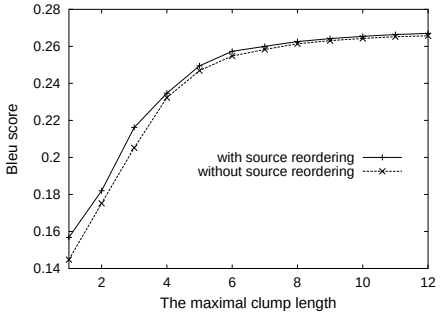


Figure 6: The MT results on TestSet1

in Figure 5.[7] Note the similarity between the new parse tree and the target parse tree in Figure 3. The new sentence that is read off from the new parse tree is a sequence of English words in French word order, and it is used for training or decoding.

4 Experiments

For evaluation, we use the 90-million-word English-French Canadian Hansard corpus as the training data. From the training data, we extract 2.9 million rewrite patterns that occur at least once. After organizing and filtering the patterns as described in Section 3.2.4, the number of patterns reduces to 56 thousand, and 1042 of them are unlexicalized. When applying them to the source parse trees in training data, on average each source tree triggers only 1.4 patterns. The most commonly used patterns are the ones that reorder a noun and its modifiers.

To evaluate the impact of source reordering on MT performance, we train the baseline system on the same Canadian Hansard corpus. For testing, we use two test sets. The first one, TestSet1, contains 3971 sentences from the same Hansard corpus, which are *not* included in the training data. The second test set, TestSet2, contains 500 sentences from various news articles. The average sentence lengths for the two test sets are 21.7 words and 28.8 words, respectively. The Bleu scores with *one* reference translation are shown in Figure 6 and 7.[8] In both figures,

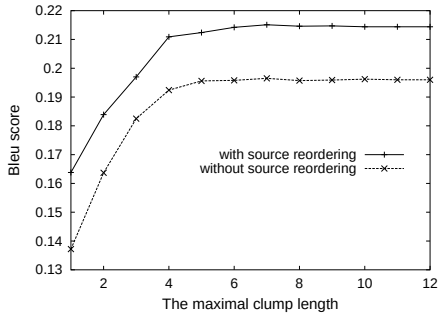


Figure 7: The MT results on TestSet2

X-axis is the maximal clump length that the MT system memorizes during the training time; i.e., the value of n as a clump is simply an n-gram. The top curve shows the Bleu scores when both training and test sentences are reordered using rewrite patterns; and the bottom curve illustrates the baseline results, where the same system is trained and tested on the original sentences without reordering. The 95% confidence intervals for TestSet1 ranges from $\pm$ 0.005 to $\pm$ 0.011; the 95% confidence intervals for TestSet2 ranges from $\pm$ 0.012 to $\pm$ 0.017 as TestSet2 contains less number of sentences than TestSet1.

Several conclusions can be drawn from the two figures. First, clump-based systems benefit from memorizing n-grams, but the improvement quickly levels off because there are fewer long n-grams that appear in both training and test data. Second, the curves in Figure 7 become flat once n reaches 4. In contrast, the curves in 6 keep going up even after n reaches 8. This indicates that TestSet1 (the Hansard test set) is much closer to the training data than is TestSet2 (the new-domain test set), which is not surprising given that TestSet1 comes from the same corpus as the training data. Third, the improvement provided by reordering source sentences is statistically significant for TestSet2 no matter what the value of n is, but the improvement is no longer statistically significant for TestSet1 once n reaches 4. This is because the main benefit of using rewrite patterns is to produce the correct target word order for *unseen* source word sequences. Since TestSet1 is very close to the training data and there are fewer unseen source word sequences, the benefit of reordering diminishes.

For the experimental results in Figure 6 and 7, the decoder translates the source clumps in monotonic order. To test the effect of reordering on the *target* side, we rerun the experiment on TestSet2 where we allow the decoder to translate the source clump in a non-monotonic order.[9] The Bleu score with

[7] In the new parse tree, we attach _t to any word inserted by a rewrite rule; for instance, "la_t" is a made-up token that comes from the word "la" in the target language.

[8] Clearly the Bleu scores would be higher if more references were used. For instance, in a separate experiment conducted by our colleague Yaser Al-onaizan, the Bleu score of one system output is 0.44 when four references are used. When the same output is evaluated against each of the four reference translations, the average 1-reference Bleu score is 0.25.

[9] Because the number of source clump permutations is exponential, in this experiment, we allow only certain kinds of clump reordering. Even with this restriction, the decoding

Table 2: The effect of non-monotonic decoding

	non-monotonic decoding	monotonic decoding
rewrite patterns not used	0.187	0.196
rewrite patterns used	0.185	0.215

one reference is shown in Table 2, where the maximal clump length is set to 8, and 95% confidence intervals are about ± 0.016. This result indicates that letting the decoder *reorder* clumps using the word-based trigram language model alone hurts the performance.

5 Conclusion

This paper addresses two limitations of clump-based SMTs: First, the systems lack a mechanism for expressing and using generalization that accounts for reorderings of linguistic phrases. Second, the ordering of clumps in such systems does not respect linguistic phrase boundaries. To alleviate these problems, we have used automatically learned rewrite patterns to preprocess the source sentences so that they have a word order similar to that of the target language and the decoder can translate the sentences in monotonic order. We have used broad-coverage rule-based parsers both in the discovery of the rewrite patterns, and in the runtime sentence reordering by applying those patterns to parse trees. Our experiments show 10% relative improvement in Bleu measure for a more realistic test set.

There are several directions for future work. First, we would like to try our approach on other language pairs with more word-order difference. Second, we want to study how parsing accuracy affects the reordering results and eventually the MT results. Third, we are hoping to exploit the possibility of using rewrite patterns directly in the decoder.

References

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19(2):263–311.

Kenji Imamura, Eiichiro Sumita, and Yuji Matsumoto. 2003. Feedback Cleaning of Machine Translation Rules Using Automatic Evaluation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, pages 447–454.

Kenji Imamura. 2001. Hierarchical Phrase Alignment Harmonized with Parsing. In *Proc. of the 6th Natural Language Processing Pacific Rim Symposium (NLPRS 2001)*, pages 377–384.

Hiroyuki Kaji, Yuuko Kida, and Yasutsugu Morimoto. 1992. Learning Translation Templates From Bilingual Text. In *Proc. of the 14th International Conference on Computational Linguistics (COLING 1992)*, pages 672–678.

Benoit Lavoie, Michael White, and Tanya Korelsky. 2001. Inducing Lexico-Structural Transfer Rules from Parsed Bi-texts. In *Proc. of the Workshop on Data-driven Machine Translation in conjunction with ACL 2001.*

Daniel Marcu and William Wong. 2002. A Phrased-Based, Joint Probability Model for Statistical Machine Translation. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 133–139.

Yuji Matsumoto, Hiroyuki Ishimoto, and Takehito Utsuro. 1993. Structural Matching of Parallel Texts. In *Proc of the 31st Annual Meeting of the Association for Computational Linguistics (ACL 1993)*, pages 23–30.

Michael C. McCord and Arendse Bernth. 1998. The LMT Transformational System. In *Proc. of the Third Conference of the Association for Machine Translation in the Americas (AMTA 1998)*, pages 344–355.

Michael McCord. 1980. Slot Grammars. *Computational Linguistics*, 6(1):31–43.

Michael McCord. 1989. Design of LMT: A Prolog-based Machine Translation System. *Computational Linguistics*, 15:33–52.

Michael C. McCord. 1990. Slot Grammar: A system for simpler construction of practical natural language grammars. In R. Studer, editor, *Natural Language and Logic: International Scientific Symposium, Lecture Notes in Computer Science*, pages 118–145. Springer Verlag, Berlin.

Michael C. McCord. 1993. Heuristics for broad-coverage natural language parsing. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 127–132. Morgan-Kaufmann.

Arul Menezes and Stephen D. Richardson. 2001. A best-first alignment algorithm for automatic extraction of transfer mappings from bilingual corpora. In *Proc. of the Workshop on Data-Driven Machine Translation in conjunction with ACL 2001.*

Adam Meyers, Michiko Kosaka, and Ralph Grishman. 2000. Chart-based transfer rule application in machine translation. In *Proc. of the 18th International Conference on Computational Linguistics (COLING 2000).*

Franz-Josef Och, Christoph Tillmann, and Hermann Ney. 1999. Improved Alignment Models for Statistical Machine Translation. In *Proc. of the Joint Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC 1999)*, pages 20–28.

Franz Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. 2004. A Smorgasbord of Features for Statistical Machine Translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL 2004).*

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proc. of the 40th Annual Conf. of the Association for Computational Linguistics (ACL 2002)*, pages 311–318.

Christoph Tillmann and Fei Xia. 2003. A Phrase-Based Unigram Model for Statistical Machine Translation. In *Proc. of the third Human Language Technology Conference (HLT/NAACL 2003).*

Hideo Watanabe, Sado Kurohashi, and Eiji Aramak. 2000. Finding Structural Correspondences from Bilingual Parsed Corpus for Corpus-based Translation. In *Proc. of the 18th International Conference on Computational Linguistics (COLING 2000).*

Dekai Wu. 1996. A Polynominal-Time Algorithm for Statistical Machine Translation. In *Proc of the 34th Annual Meeting of the Association for Computational Linguistics (ACL 1996).*

Kenji Yamada and Kevin Knight. 2002. A Decoder for Syntax-based Statistical MT. In *Proc. of the 40th Annual Conf. of the Association for Computational Linguistics (ACL 2002)*, pages 303–310.

time is 140 times slower than the monotonic decoding.