Efficient Top-Down BTG Parsing for Machine Translation Preordering

Tetsuji Nakagawa
Google Japan Inc.
tnaka@google.com

Abstract

We present an efficient incremental top-down parsing method for preordering based on Bracketing Transduction Grammar (BTG). The BTG-based preordering framework (Neubig et al., 2012) can be applied to any language using only parallel text, but has the problem of computational efficiency. Our top-down parsing algorithm allows us to use the early update technique easily for the latent variable structured Perceptron algorithm with beam search, and solves the problem.

Experimental results showed that the top-down method is more than 10 times faster than a method using the CYK algorithm. A phrase-based machine translation system with the top-down method had statistically significantly higher BLEU scores for 7 language pairs without relying on supervised syntactic parsers, compared to baseline systems using existing preordering methods.

1 Introduction

The difference of the word order between source and target languages is one of major problems in phrase-based statistical machine translation. In order to cope with the issue, many approaches have been studied. Distortion models consider word reordering in decoding time using such as distance (Koehn et al., 2003) and lexical information (Tillman, 2004). Another direction is to use more complex translation models such as hierarchical models (Chiang, 2007). However, these approaches suffer from the long-distance reordering issue and computational complexity.

Preordering (reordering-as-preprocessing) (Xia and McCord, 2004; Collins et al., 2005) is another approach for tackling the problem, which modifies the word order of an input sentence in a source language to have the word order in a target language (Figure 1(a)).

Various methods for preordering have been studied, and a method based on Bracketing Transduction Grammar (BTG) was proposed by Neubig et al. (2012). It reorders source sentences by handling sentence structures as latent variables. The method can be applied to any language using only parallel text. However, the method has the problem of computational efficiency.

In this paper, we propose an efficient incremental top-down BTG parsing method which can be applied to preordering. Model parameters can be learned using latent variable Perceptron with the early update technique (Collins and Roark, 2004), since the parsing method provides an easy way for checking the reachability of each parser state to valid final states. We also try to use forced-decoding instead of word alignment based on Expectation Maximization (EM) algorithms in order to create better training data for preordering. In experiments, preordering using the top-down parsing algorithm was faster and gave higher BLEU scores than BTG-based preordering using the CYK algorithm. Compared to existing preordering methods, our method had better or comparable BLEU scores without using supervised parsers.

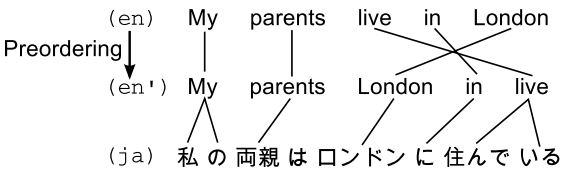
2 Previous Work

2.1 Preordering for Machine Translation

Many preordering methods which use syntactic parse trees have been proposed, because syntactic information is useful for determining the word order in a target language, and it can be used to restrict the search space against all the possible permutations. Preordering methods using manually created rules on parse trees have been studied (Collins et al., 2005; Xu et al., 2009), but

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 208–218, Beijing, China, July 26-31, 2015.

(a) Preordering for English-Japanese translation.



(b) BTG tree representing the preordering.

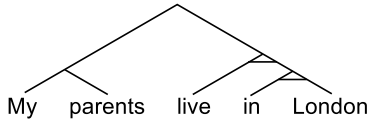


Figure 1: An example of preordering.

linguistic knowledge for a language pair is necessary to create such rules. Preordering methods which automatically create reordering rules or utilize statistical classifiers have also been studied (Xia and McCord, 2004; Li et al., 2007; Genzel, 2010; Visweswariah et al., 2010; Yang et al., 2012; Miceli Barone and Attardi, 2013; Lerner and Petrov, 2013; Jehl et al., 2014). These methods rely on source-side parse trees and cannot be applied to languages where no syntactic parsers are available.

There are preordering methods that do not need parse trees. They are usually trained only on automatically word-aligned parallel text. It is possible to mine parallel text from the Web (Uszkoreit et al., 2010; Antonova and Misyurev, 2011), and the preordering systems can be trained without manually annotated language resources. Tromble and Eisner (2009) studied preordering based on a Linear Ordering Problem by defining a pairwise preference matrix. Khalilov and Sima'an (2010) proposed a method which swaps adjacent two words using a maximum entropy model. Visweswariah et al. (2011) regarded the preordering problem as a Traveling Salesman Problem (TSP) and applied TSP solvers for obtaining reordered words. These methods do not consider sentence structures.

DeNero and Uszkoreit (2011) presented a preordering method which builds a monolingual parsing model and a tree reordering model from parallel text. Neubig et al. (2012) proposed to train a discriminative BTG parser for preordering directly from word-aligned parallel text by handling underlying parse trees with latent variables. This method is explained in detail in the next subsection. These two methods can use sentence structures for designing feature functions to score permutations.

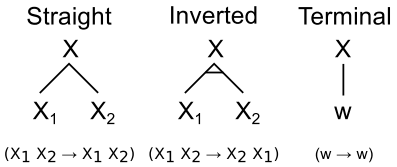


Figure 2: Bracketing transduction grammar.

2.2 BTG-based Preordering

Neubig et al. (2012) proposed a BTG-based preordering method. Bracketing Transduction Grammar (BTG) (Wu, 1997) is a binary synchronous context-free grammar with only one non-terminal symbol, and has three types of rules (Figure 2): *Straight* which keeps the order of child nodes, *Inverted* which reverses the order, and *Terminal* which generates a terminal symbol.[1]

BTG can express word reordering. For example, the word reordering in Figure 1(a) can be represented with the BTG parse tree in Figure 1(b).[2] Therefore, the task to reorder an input source sentence can be solved as a BTG parsing task to find an appropriate BTG tree.

In order to find the best BTG tree among all the possible ones, a score function is defined. Let $\Phi(m)$ denote the vector of feature functions for the BTG tree node m, and Λ denote the vector of feature weights. Then, for a given source sentence x, the best BTG tree $\hat{z}$ and the reordered sentence x' can be obtained as follows:

$$\hat{z} = \operatorname*{argmax}_{z \in Z(x)} \sum_{m \in Nodes(z)} \Lambda \cdot \Phi(m), \quad (1)$$

$$x' = Proj(\hat{z}), \quad (2)$$

where $Z(x)$ is the set of all the possible BTG trees for x, $Nodes(z)$ is the set of all the nodes in the tree z, and $Proj(z)$ is the function which generates a reordered sentence from the BTG tree z.

The method was shown to improve translation performance. However, it has a problem of processing speed. The CYK algorithm, whose computational complexity is $O(n^3)$ for a sen-

[1] Although *Terminal* produces a pair of source and target words in the original BTG (Wu, 1997), the target-side words are ignored here because both the input and the output of preordering systems are in the source language. In (Wu, 1997), (DeNero and Uszkoreit, 2011) and (Neubig et al., 2012), *Terminal* can produce multiple words. Here, we produce only one word.

[2] There may be more than one BTG tree which represents the same word reordering (e.g., the word reordering $C_3B_2A_1$ to $A_1B_2C_3$ has two possible BTG trees), and there are permutations which cannot be represented with BTG (e.g., $B_2D_4A_1C_3$ to $A_1B_2C_3D_4$, which is called the 2413 pattern).

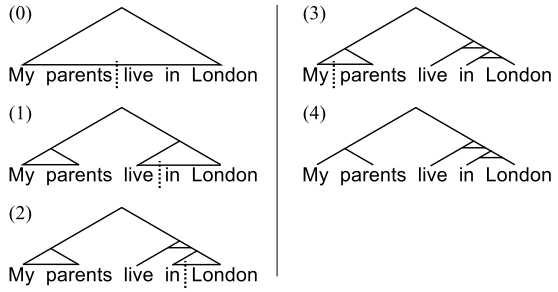


Figure 3: Top-down BTG parsing.

(0) $\langle [[0,5)], [], 0 \rangle$
(1) $\langle [[0,2),[2,5)], [(2,S)], v_1 \rangle$
(2) $\langle [[0,2),[3,5)], [(2,S),(3,I)], v_2 \rangle$
(3) $\langle [[0,2)], [(2,S),(3,I),(4,I)], v_3 \rangle$
(4) $\langle [], [(2,S),(3,I),(4,I),(1,S)], v_4 \rangle$

Table 1: Parser states in top-down parsing.

tence of length n, is used to find the best parse tree. Furthermore, due to the use of a complex loss function, the complexity at training time is $O(n^5)$ (Neubig et al., 2012). Since the computational cost is prohibitive, some techniques like cube pruning and cube growing have been applied (Neubig et al., 2012; Na and Lee, 2013). In this study, we propose a top-down parsing algorithm in order to achieve fast BTG-based preordering.

3 Preordering with Incremental Top-Down BTG Parsing

3.1 Parsing Algorithm

We explain an incremental top-down BTG parsing algorithm using Figure 3, which illustrates how a parse tree is built for the example sentence in Figure 1. At the beginning, a tree (span) which covers all the words in the sentence is considered. Then, a span which covers more than one word is split in each step, and the node type (*Straight* or *Inverted*) for the splitting point is determined. The algorithm terminates after $(n-1)$ iterations for a sentence with n words, because there are $(n-1)$ positions which can be split.

We consider that the incremental parser has a *parser state* in each step, and define the state as a triple $\langle P, C, v \rangle$. P is a stack of unresolved spans. A span denoted by $[p, q)$ covers the words $x_p \cdots x_{q-1}$ for an input word sequence $x = x_0 \cdots x_{|x|-1}$. C is a list of past parser actions. A parser action denoted by (r, o) represents the action to split a span at the position between x_{r-1} and x_r with the node type $o \in \{\mathrm{S}, \mathrm{I}\}$, where S and I indicate *Straight* and *Inverted* respectively. v is the score of the state, which is the sum of the scores for the nodes constructed so far. Parsing starts with the initial state $\langle [[0, |x|)], [], 0 \rangle$, because there is one span covering all the words at the beginning. In each step, a span is popped from the top of the stack, and a splitting point in the span and its node type are determined. The new spans generated by the split are pushed onto the stack if their lengths are greater than 1, and the action is added to the list. On termination, the parser has the final state $\langle [], [c_0, \cdots, c_{|x|-2}], v \rangle$, because the stack is empty and there are $(|x|-1)$ actions in total. The parse tree can be obtained from the list of actions. Table 1 shows the parser state for each step in Figure 3.

Input: Sentence x, feature weights Λ, beam width k.
Output: BTG parse tree.
$S_0 \leftarrow \{\langle [[0, |x|)], [], 0 \rangle\}$ // Initial state.
for $i := 1, \cdots, |x|-1$ **do**
  $S \leftarrow \{\}$ // Set of the next states.
  foreach $s \in S_{i-1}$ **do**
    $S \leftarrow S \cup \tau_{x,\Lambda}(s)$ // Generate next states.
  $S_i \leftarrow Top_k(S)$ // Select k-best states.
$\hat{s} = \mathrm{argmax}_{s \in S_{|x|-1}} Score(s)$
return $Tree(\hat{s})$

function $\tau_{x,\Lambda}(\langle P, C, v \rangle)$
  $[p, q) \leftarrow P.pop()$
  $S \leftarrow \{\}$
  for $r := p+1, \cdots, q$ **do**
    $P' \leftarrow P$
    if $r - p > 1$ **then**
      $P'.push([p, r))$
    if $q - r > 1$ **then**
      $P'.push([r, q))$
    $v^{\mathrm{S}} \leftarrow v + \Lambda \cdot \Phi(x, C, p, q, r, \mathrm{S})$
    $v^{\mathrm{I}} \leftarrow v + \Lambda \cdot \Phi(x, C, p, q, r, \mathrm{I})$
    $C^{\mathrm{S}} \leftarrow C;\ C^{\mathrm{S}}.append((r, \mathrm{S}))$
    $C^{\mathrm{I}} \leftarrow C;\ C^{\mathrm{I}}.append((r, \mathrm{I}))$
    $S \leftarrow S \cup \{\langle P', C^{\mathrm{S}}, v^{\mathrm{S}} \rangle, \langle P', C^{\mathrm{I}}, v^{\mathrm{I}} \rangle\}$
  return S

Figure 4: Top-down BTG parsing with beam search.

The top-down parsing method can be used with beam search as shown in Figure 4. $\tau_{x,\Lambda}(s)$ is a function which returns the set of all the possible next states for the state s. $Top_k(S)$ returns the top k states from S in terms of their scores, $Score(s)$ returns the score of the state s, and $Tree(s)$ returns the BTG parse tree constructed from s. $\Phi(x, C, p, q, r, o)$ is the feature vector for the node created by splitting the span $[p, q)$ at r with the node type o, and is explained in Section 3.3.

3.2 Learning Algorithm

Model parameters Λ are estimated from training examples. We assume that each training example

consists of a sentence x and its word order in a target language $y = y_0 \cdots y_{|x|-1}$, where y_i is the position of x_i in the target language. For example, the example sentence in Figure 1(a) will have $y = 0, 1, 4, 3, 2$. y can have ambiguities. Multiple words can be reordered to the same position on the target side. The words whose target positions are unknown are indicated by position -1, and we consider such words can appear at any position.[3] For example, the word alignment in Figure 5 gives the target side word positions $y = -1, 2, 1, 0, 0$.

Statistical syntactic parsers are usually trained on tree-annotated corpora. However, corpora annotated with BTG parse trees are unavailable, and only the gold standard permutation y is available. Neubig et al. (2012) proposed to train BTG parsers for preordering by regarding BTG trees behind word reordering as latent variables, and we use latent variable Perceptron (Sun et al., 2009) together with beam search. In latent variable Perceptron, among the examples whose latent variables are compatible with a gold standard label, the one with the highest score is picked up as a positive example. Such an approach was used for parsing with multiple correct actions (Goldberg and Elhadad, 2010; Sartorio et al., 2013).

Figure 6 describes the training algorithm.[4] $\Phi(x, s)$ is the feature vector for all the nodes in the partial parse tree at the state s, and $\tau_{x,\Lambda,y}(s)$ is the set of all the next states for the state s. The algorithm adopts the early update technique (Collins and Roark, 2004) which terminates incremental parsing if a correct state falls off the beam, and there is no possibility to obtain a correct output. Huang et al. (2012) proposed the violation-fixing Perceptron framework which is guaranteed to converge even if inexact search is used, and also showed that early update is a special case of the framework. We define that a parser state is *valid* if the state can reach a final state whose BTG parse tree is compatible with y. Since this is a latent variable setting in which multiple states can reach correct final states, early update occurs when all the valid states fall off the beam (Ma et al., 2013; Yu et al., 2013). In order to use early update, we need to check the validity of each parser state. We extend the parser state to the four tuple $\langle P, A, v, w \rangle$, where $w \in \{\text{true}, \text{false}\}$ is the validity of the state. We remove training examples which cannot be represented with BTG beforehand and set w of the initial state to true. The function $Valid(s)$ in Figure 6 returns the validity of state s. One advantage of the top-down parsing algorithm is that it is easy to track the validity of each state. The validity of a state can be calculated using the following property, and we can implement the function $\tau_{x,\Lambda,y}(s)$ by modifying the function $\tau_{x,\Lambda}(s)$ in Figure 4.

Lemma 1. *When a valid state s, which has $[p, q)$ in the top of the stack, transitions to a state s' by the action (r, o), s' is also valid if and only if the following condition holds:*

$$\begin{aligned} &\forall i \in \{p, \cdots, r-1\}\, y_i = -1 \quad \vee \\ &\forall i \in \{r, \cdots, q-1\}\, y_i = -1 \quad \vee \\ &\Big(o = \mathrm{S} \wedge \max_{\substack{i=p,\cdots,r-1 \\ y_i \neq -1}} y_i \leq \min_{\substack{i=r,\cdots,q-1 \\ y_i \neq -1}} y_i\Big) \quad \vee \\ &\Big(o = \mathrm{I} \wedge \max_{\substack{i=r,\cdots,q-1 \\ y_i \neq -1}} y_i \leq \min_{\substack{i=p,\cdots,r-1 \\ y_i \neq -1}} y_i\Big). \end{aligned} \tag{3}$$

Proof. Let π_i denote the position of x_i after reordering by BTG parsing. If Condition (3) does not hold, there are i and j which satisfy $\pi_i < \pi_j \wedge y_i > y_j \wedge y_i \neq -1 \wedge y_j \neq -1$, and π_i and π_j are not compatible with y. Therefore, s' is valid only if Condition (3) holds.

When Condition (3) holds, a valid permutation can be obtained if the spans $[p, r)$ and $[r, q)$ are BTG-parsable. They are BTG-parsable as shown below. Let us assume that y does not have ambiguities. The class of the permutations which can be represented by BTG is known as *separable permutations* in combinatorics. It can be proven (Bose et al., 1998) that a permutation is a separable permutation if and only if it contains neither the 2413 nor the 3142 patterns. Since s is valid, y is a separable permutation. y does not contain the 2413 nor the 3142 patterns, and any subsequence of y also does not contain the patterns. Thus, $[p, r)$ and $[r, q)$ are separable permutations. The above argument holds even if y has ambiguities (duplicated positions or unaligned words). In such a case, we can always make a word order y' which specializes y and has no ambiguities (e.g., $y' = 2, 1.0, 0.0, 0.1, 1.1$ for $y = -1, 1, 0, 0, 1$), because s is valid, and there is at least one BTG parse tree which licenses y. Any subsequence in

[3] In (Neubig et al., 2012), the positions of such words were fixed by heuristics. In this study, the positions are not fixed, and all the possibilities are considered by latent variables.

[4] Although the simple Perceptron algorithm is used for explanation, we actually used the Passive Aggressive algorithm (Crammer et al., 2006) with the parameter averaging technique (Freund and Schapire, 1999).

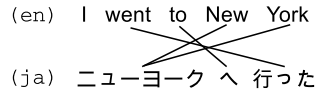


Figure 5: An example of word reordering with ambiguities.

y' is a separable permutation, and $[p, r)$ and $[r, q)$ are separable permutations. Therefore, s' is valid if Condition (3) holds. □

For dependency parsing and constituent parsing, incremental bottom-up parsing methods have been studied (Yamada and Matsumoto, 2003; Nivre, 2004; Goldberg and Elhadad, 2010; Sagae and Lavie, 2005). Our top-down approach is contrastive to the bottom-up approaches. In the bottom-up approaches, spans which cover individual words are considered at the beginning, then they are merged into larger spans in each step, and a span which covers all the words is obtained at the end. In the top-down approach, a span which covers all the words is considered at the beginning, then spans are split into smaller spans in each step, and spans which cover individual words are obtained at the end. The top-down BTG parsing method has the advantage that the validity of parser states can be easily tracked.

The computational complexity of the top-down parsing algorithm is $O(kn^2)$ for sentence length n and beam width k, because in Line 5 of Figure 4, which is repeated at most $k(n-1)$ times, at most $2(n-1)$ parser states are generated, and their scores are calculated. The learning algorithm uses the same decoding algorithm as in the parsing phase, and has the same time complexity. Note that the validity of a parser state can be calculated in $O(1)$ by pre-calculating $\min_{i=p,\cdots,r \wedge y_i \neq -1} y_i$, $\max_{i=p,\cdots,r \wedge y_i \neq -1} y_i$, $\min_{i=r,\cdots,q-1 \wedge y_i \neq -1} y_i$, and $\max_{i=r,\cdots,q-1 \wedge y_i \neq -1} y_i$ for all r for the span $[p, q)$ when it is popped from the stack.

3.3 Features

We assume that each word x_i in a sentence has three attributes: word surface form x_i^{w}, part-of-speech (POS) tag x_i^{p} and word class x_i^{c} (Section 4.1 explains how x_i^{p} and x_i^{c} are obtained).

Table 2 lists the features generated for the node which is created by splitting the span $[p, q)$ with the action (r, o). o' is the node type of the parent node, $d \in \{\mathrm{left}, \mathrm{right}\}$ indicates whether this node is the left-hand-side or the right-hand-side child of the parent node, and $Balance(p, q, r)$ returns a value among $\{\text{'<'}, \text{'='}, \text{'>'}\}$ according to the relation of the lengths of $[p, r)$ and $[r, q)$. The baseline feature templates are those used by Neubig et al. (2012), and the additional feature templates are extended features that we introduce in this study. The top-down parser is fast, and allows us to use a larger number of features.

Input: Training data $\{\langle x^l, y^l \rangle\}_{l=0}^{L-1}$,
number of iterations T, beam width k.
Output: Feature weights Λ.

```
1: Λ ← 0
2: for t := 0, ··· , T − 1 do
3:   for l := 0, ··· , L − 1 do
4:     S_0 ← {⟨[[0, |x^l|)], [], 0, true⟩}
5:     for i := 1, ··· , |x^l| − 1 do
6:       S ← {}
7:       foreach s ∈ S_{i−1} do
8:         S ← S ∪ τ_{x^l,Λ,y^l}(s)
9:       S_i ← Top_k(S)
10:      ŝ ← argmax_{s∈S} Score(s)
11:      s* ← argmax_{s∈S∧Valid(s)} Score(s)
12:      if s* ∉ S_i then
13:        break                    // Early update.
14:    if ŝ ≠ s* then
15:      Λ ← Λ + Φ(x^l, s*) − Φ(x^l, ŝ)
16: return Λ
```

Figure 6: A training algorithm for latent variable Perceptron with beam search.

In order to make the feature generation efficient, the attributes of all the words are converted to their 64-bit hash values beforehand, and concatenating the attributes is executed not as string manipulation but as faster integer calculation to generate a hash value by merging two hash values. The hash values are used as feature names. Therefore, when accessing feature weights stored in a hash table using the feature names as keys, the keys can be used as their hash values. This technique is different from the hashing trick (Ganchev and Dredze, 2008) which directly uses hash values as indices, and no noticeable differences in accuracy were observed by using this technique.

3.4 Training Data for Preordering

As described in Section 3.2, each training example has y which represents correct word positions after reordering. However, only word alignment data is generally available, and we need to convert it to y. Let A_i denote the set of indices of the target-side words which are aligned to the source-side word x_i. We define an order relation between two words:

$$x_i \leq x_j \quad \Leftrightarrow \quad \forall a \in A_i \setminus A_j, \forall b \in A_j \; a \leq b \wedge \forall a \in A_i, \forall b \in A_j \setminus A_i \; a \leq b. \quad (4)$$

Baseline Feature Template
$o(q-p), oBalance(p,q,r),$ $ox^{\mathrm{w}}_{p-1}, ox^{\mathrm{w}}_{p}, ox^{\mathrm{w}}_{r-1}, ox^{\mathrm{w}}_{r}, ox^{\mathrm{w}}_{q-1}, ox^{\mathrm{w}}_{q}, ox^{\mathrm{w}}_{p}x^{\mathrm{w}}_{q-1}, ox^{\mathrm{w}}_{r-1}x^{\mathrm{w}}_{r},$ $ox^{\mathrm{p}}_{p-1}, ox^{\mathrm{p}}_{p}, ox^{\mathrm{p}}_{r-1}, ox^{\mathrm{p}}_{r}, ox^{\mathrm{p}}_{q-1}, ox^{\mathrm{p}}_{q}, ox^{\mathrm{p}}_{p}x^{\mathrm{p}}_{q-1}, ox^{\mathrm{p}}_{r-1}x^{\mathrm{p}}_{r},$ $ox^{\mathrm{c}}_{p-1}, ox^{\mathrm{c}}_{p}, ox^{\mathrm{c}}_{r-1}, ox^{\mathrm{c}}_{r}, ox^{\mathrm{c}}_{q-1}, ox^{\mathrm{c}}_{q}, ox^{\mathrm{c}}_{p}x^{\mathrm{c}}_{q-1}, ox^{\mathrm{c}}_{r-1}x^{\mathrm{c}}_{r}.$
Additional Feature Template
$o\min(r-p,5)\min(q-r,5), oo', oo'd,$ $ox^{\mathrm{w}}_{p-1}x^{\mathrm{w}}_{p}, ox^{\mathrm{w}}_{p}x^{\mathrm{w}}_{r-1}, ox^{\mathrm{w}}_{p}x^{\mathrm{w}}_{r}, ox^{\mathrm{w}}_{r-1}x^{\mathrm{w}}_{q-1}, ox^{\mathrm{w}}_{r}x^{\mathrm{w}}_{q-1}, ox^{\mathrm{w}}_{q-1}x^{\mathrm{w}}_{q},$ $ox^{\mathrm{w}}_{r-2}x^{\mathrm{w}}_{r-1}x^{\mathrm{w}}_{r}, ox^{\mathrm{w}}_{p}x^{\mathrm{w}}_{r-1}x^{\mathrm{w}}_{r}, ox^{\mathrm{w}}_{r-1}x^{\mathrm{w}}_{r}x^{\mathrm{w}}_{q-1}, ox^{\mathrm{w}}_{r-1}x^{\mathrm{w}}_{r}x^{\mathrm{w}}_{r+1},$ $ox^{\mathrm{w}}_{p}x^{\mathrm{w}}_{r-1}x^{\mathrm{w}}_{r}x^{\mathrm{w}}_{q-1},$ $oo'dx^{\mathrm{w}}_{p}, oo'dx^{\mathrm{w}}_{r-1}, oo'dx^{\mathrm{w}}_{r}, oo'dx^{\mathrm{w}}_{q-1}, oo'dx^{\mathrm{w}}_{p}x^{\mathrm{w}}_{q-1},$ $ox^{\mathrm{p}}_{p-1}x^{\mathrm{p}}_{p}, ox^{\mathrm{p}}_{p}x^{\mathrm{p}}_{r-1}, ox^{\mathrm{p}}_{p}x^{\mathrm{p}}_{r}, ox^{\mathrm{p}}_{r-1}x^{\mathrm{p}}_{q-1}, ox^{\mathrm{p}}_{r}x^{\mathrm{p}}_{q-1}, ox^{\mathrm{p}}_{q-1}x^{\mathrm{p}}_{q},$ $ox^{\mathrm{p}}_{r-2}x^{\mathrm{p}}_{r-1}x^{\mathrm{p}}_{r}, ox^{\mathrm{p}}_{p}x^{\mathrm{p}}_{r-1}x^{\mathrm{p}}_{r}, ox^{\mathrm{p}}_{r-1}x^{\mathrm{p}}_{r}x^{\mathrm{p}}_{q-1}, ox^{\mathrm{p}}_{r-1}x^{\mathrm{p}}_{r}x^{\mathrm{p}}_{r+1},$ $ox^{\mathrm{p}}_{p}x^{\mathrm{p}}_{r-1}x^{\mathrm{p}}_{r}x^{\mathrm{p}}_{q-1},$ $oo'dx^{\mathrm{p}}_{p}, oo'dx^{\mathrm{p}}_{r-1}, oo'dx^{\mathrm{p}}_{r}, oo'dx^{\mathrm{p}}_{q-1}, oo'dx^{\mathrm{p}}_{p}x^{\mathrm{p}}_{q-1},$ $ox^{\mathrm{c}}_{p-1}x^{\mathrm{c}}_{p}, ox^{\mathrm{c}}_{p}x^{\mathrm{c}}_{r-1}, ox^{\mathrm{c}}_{p}x^{\mathrm{c}}_{r}, ox^{\mathrm{c}}_{r-1}x^{\mathrm{c}}_{q-1}, ox^{\mathrm{c}}_{r}x^{\mathrm{c}}_{q-1}, ox^{\mathrm{c}}_{q-1}x^{\mathrm{c}}_{q},$ $ox^{\mathrm{c}}_{r-2}x^{\mathrm{c}}_{r-1}x^{\mathrm{c}}_{r}, ox^{\mathrm{c}}_{p}x^{\mathrm{c}}_{r-1}x^{\mathrm{c}}_{r}, ox^{\mathrm{c}}_{r-1}x^{\mathrm{c}}_{r}x^{\mathrm{c}}_{q-1}, ox^{\mathrm{c}}_{r-1}x^{\mathrm{c}}_{r}x^{\mathrm{c}}_{r+1},$ $ox^{\mathrm{c}}_{p}x^{\mathrm{c}}_{r-1}x^{\mathrm{c}}_{r}x^{\mathrm{c}}_{q-1},$ $oo'dx^{\mathrm{c}}_{p}, oo'dx^{\mathrm{c}}_{r-1}, oo'dx^{\mathrm{c}}_{r}, oo'dx^{\mathrm{c}}_{q-1}, oo'dx^{\mathrm{c}}_{p}x^{\mathrm{c}}_{q-1}.$

Table 2: Feature templates.

Then, we sort x using the order relation and assign the position of x_i in the sorted result to y_i. If there are two words x_i and x_j in x which satisfy neither $x_i \leq x_j$ nor $x_j \leq x_i$ (that is, x does not make a totally ordered set with the order relation), then x cannot be sorted, and the example is removed from the training data. -1 is assigned to the words which do not have aligned target words. Two words x_i and x_j are regarded to have the same position if $x_i \leq x_j$ and $x_j \leq x_i$.

The quality of training data is important to make accurate preordering systems, but automatically word-aligned data by EM algorithms tend to have many wrong alignments. We use forced-decoding in order to make training data for preordering. Given a parallel sentence pair and a phrase table, forced-decoding tries to translate the source sentence to the target sentence, and produces phrase alignments. We train the parameters for forced-decoding using the same parallel data used for training the final translation system. Infrequent phrase translations are pruned when the phrase table is created, and forced-decoding does not always succeed for the parallel sentences in the training data. Forced-decoding tends to succeed for shorter sentences, and the phrase-alignment data obtained by forced-decoding is biased to contain more shorter sentences. Therefore, we apply the following processing for the output of forced-decoding to make training data for preordering:

1. Remove sentences which contain less than 3 or more than 50 words.
2. Remove sentences which contain less than 3 phrase alignments.
3. Remove sentences if they contain word 5-grams which appear in other sentences in order to drop boilerplates.
4. Lastly, randomly resample sentences from the pool of filtered sentences to make the distribution of the sentence lengths follow a normal distribution with the mean of 20 and the standard deviation of 8. The parameters were determined from randomly sampled sentences from the Web.

4 Experiments

4.1 Experimental Settings

We conduct experiments for 12 language pairs: Dutch (nl)-English (en), en-nl, en-French (fr), en-Japanese (ja), en-Spanish (es), fr-en, Hindi (hi)-en, ja-en, Korean (ko)-en, Turkish (tr)-en, Urdu (ur)-en and Welsh (cy)-en.

We use a phrase-based statistical machine translation system which is similar to (Och and Ney, 2004). The decoder adopts the regular distance distortion model, and also incorporates a maximum entropy based lexicalized phrase reordering model (Zens and Ney, 2006). The distortion limit is set to 5 words. Word alignments are learned using 3 iterations of IBM Model-1 (Brown et al., 1993) and 3 iterations of the HMM alignment model (Vogel et al., 1996). Lattice-based minimum error rate training (MERT) (Macherey et al., 2008) is applied to optimize feature weights. 5-gram language models trained on sentences collected from various sources are used.

The translation system is trained with parallel sentences automatically collected from the Web. The parallel data for each language pair consists of around 400 million source and target words. In order to make the development data for MERT and test data (3,000 and 5,000 sentences respectively for each language), we created parallel sentences by randomly collecting English sentences from the Web, and translating them by humans into each language.

As an evaluation metric for translation quality, BLEU (Papineni et al., 2002) is used. As intrinsic evaluation metrics for preordering, Fuzzy Reordering Score (FRS) (Talbot et al., 2011) and Kendall's τ (Kendall, 1938; Birch et al., 2010; Isozaki et al., 2010) are used. Let ρ_i denote the position in the input sentence of the $(i+1)$-th token in a preordered word sequence excluding unaligned words in the gold-standard evaluation data. For

		en-ja				ja-en			
		Training (min.)	Preordering (sent./sec.)	FRS	τ	Training (min.)	Preordering (sent./sec.)	FRS	τ
Top-Down	(EM-100k)	63	87.8	77.83	87.78	81	178.4	74.60	83.78
Top-Down (Basic Feat.)	(EM-100k)	9	475.1	75.25	87.26	9	939.0	73.56	83.66
Lader	(EM-100k)	1562	4.3	75.41	86.85	2087	12.3	74.89	82.15

Table 3: Speed and accuracy of preordering.

		en-ja			ja-en		
		FRS	τ	BLEU	FRS	τ	BLEU
Top-Down	(Manual-8k)	81.57	90.44	18.13	79.26	86.47	14.26
	(EM-10k)	74.79	85.87	17.07	72.51	82.65	14.55
	(EM-100k)	77.83	87.78	17.66	74.60	83.78	14.84
	(Forced-10k)	76.10	87.45	16.98	75.36	83.96	14.78
	(Forced-100k)	78.76	89.22	17.88	76.58	85.25	**15.54**
Lader	(EM-100k)	75.41	86.85	17.40	74.89	82.15	14.59
No-Preordering		46.17	65.07	13.80	59.35	65.30	10.31
Manual-Rules		80.59	90.30	**18.68**	73.65	81.72	14.02
Auto-Rules		64.13	84.17	16.80	60.60	75.49	12.59
Classifier		80.89	90.61	**18.53**	74.24	82.83	13.90

Table 4: Performance of preordering for various training data. Bold BLEU scores indicate no statistically significant difference at $p < 0.05$ from the best system (Koehn, 2004).

example, the preordering result "New York I to went" for the gold-standard data in Figure 5 has $\rho = 3, 4, 2, 1$. Then FRS and τ are calculated as follows:

$$FRS = \frac{B}{|\rho| + 1}, \quad (5)$$

$$B = \sum_{i=0}^{|\rho|-2} \delta(y_{\rho_i}{=}y_{\rho_{i+1}} \vee y_{\rho_i}{+}1{=}y_{\rho_{i+1}}) + \delta(y_{\rho_0}{=}0) + \delta(y_{\rho_{|\rho|-1}}{=}\max_i y_i), \quad (6)$$

$$\tau = \frac{\sum_{i=0}^{|\rho|-2} \sum_{j=i+1}^{|\rho|-1} \delta(y_{\rho_i} \leq y_{\rho_j})}{\frac{1}{2}|\rho|(|\rho| - 1)}, \quad (7)$$

where $\delta(X)$ is the Kronecker's delta function which returns 1 if X is true or 0 otherwise. These scores are calculated for each sentence, and are averaged over all sentences in test data. As above, FRS can be calculated as the precision of word bigrams (B is the number of the word bigrams which exist both in the system output and the gold standard data). This formulation is equivalent to the original formulation based on chunk fragmentation by Talbot et al. (2011). Equation (6) takes into account the positions of the beginning and the ending words (Neubig et al., 2012). Kendall's τ is equivalent to the (normalized) crossing alignment link score used by Genzel (2010).

We prepared three types of training data for learning model parameters of BTG-based preordering:

Manual-8k Manually word-aligned 8,000 sentence pairs.

EM-10k, EM-100k These are the data obtained with the EM-based word alignment learning. From the word alignment result for phrase translation extraction described above, 10,000 and 100,000 sentence pairs were randomly sampled. Before the sampling, the data filtering procedure 1 and 3 in Section 3.4 were applied, and also sentences were removed if more than half of source words do not have aligned target words. Word alignment was obtained by symmetrizing source-to-target and target-to-source word alignment with the INTERSECTION heuristic.[5]

Forced-10k, Forced-100k These are 10,000 and 100,000 word-aligned sentence pairs obtained with forced-decoding as described in Section 3.4.

As test data for intrinsic evaluation of preordering, we manually word-aligned 2,000 sentence pairs for en-ja and ja-en.

Several preordering systems were prepared in order to compare the following six systems:

No-Preordering This is a system without preordering.

Manual-Rules This system uses the preordering method based on manually created rules (Xu

[5] In our preliminary experiments, the UNION and GROW-DIAG-FINAL heuristics were also applied to generate the training data for preordering, but INTERSECTION performed the best.

	No-Preordering	Manual-Rules	Auto-Rules	Classifier	Lader (EM-100k)	Top-Down (EM-100k)	Top-Down (Forced-100k)
nl-en	34.01	-	34.24	**35.42**	33.83	**35.49**	**35.51**
en-nl	25.33	-	25.59	**25.99**	25.30	**25.82**	25.66
en-fr	25.86	-	26.39	26.35	26.50	**26.75**	**26.81**
en-ja	13.80	**18.68**	16.80	**18.53**	17.40	17.66	17.88
en-es	29.50	-	29.63	**30.09**	29.70	**30.26**	**30.24**
fr-en	32.33	-	32.09	32.28	32.43	**33.00**	**32.99**
hi-en	19.86	-	-	-	24.24	**24.98**	**24.97**
ja-en	10.31	14.02	12.59	13.90	14.59	14.84	**15.54**
ko-en	14.13	-	15.86	19.46	18.65	**19.67**	**19.88**
tr-en	18.26	-	-	-	22.80	**23.91**	**24.18**
ur-en	14.48	-	-	-	16.62	17.65	**18.32**
cy-en	**41.68**	-	-	-	**41.79**	**41.95**	**41.86**

Table 5: BLEU score comparison.

	Distortion Limit	No-Preordering	Manual-Rules	Auto-Rules	Classifier	Lader (EM-100k)	Top-Down (EM-100k)	Top-Down (Forced-100k)
en-ja	5	13.80	**18.68**	16.80	**18.53**	17.40	17.66	17.88
en-ja	0	11.99	**18.34**	16.87	**18.31**	16.95	17.36	17.88
ja-en	5	10.31	14.02	12.59	13.90	14.59	14.84	**15.54**
ja-en	0	10.03	12.43	11.33	13.09	14.38	14.72	**15.34**

Table 6: BLEU scores for different distortion limits.

et al., 2009). We made 43 precedence rules for en-ja, and 24 for ja-en.

Auto-Rules This system uses the rule-based preordering method which automatically learns the rules from word-aligned data using the Variant 1 learning algorithm described in (Genzel, 2010). 27 to 36 rules were automatically learned for each language pair.

Classifier This system uses the preordering method based on statistical classifiers (Lerner and Petrov, 2013), and the 2-step algorithm was implemented.

Lader This system uses Latent Derivation Reorderer (Neubig et al., 2012), which is a BTG-based preordering system using the CYK algorithm.[6] The basic feature templates in Table 2 are used as features.

Top-Down This system uses the preordering system described in Section 3.

Among the six systems, Manual-Rules, Auto-Rules and Classifier need dependency parsers for source languages. A dependency parser based on the shift-reduce algorithm with beam search (Zhang and Nivre, 2011) is used. The dependency parser and all the preordering systems need POS taggers. A supervised POS tagger based on conditional random fields (Lafferty et al., 2001) trained with manually POS annotated data is used for nl, en, fr, ja and ko. For other languages, we use a POS tagger based on POS projection (Täckström et al., 2013) which does not need POS annotated data. Word classes in Table 2 are obtained by using Brown clusters (Koo et al., 2008) (the number of classes is set to 256). For both Lader and Top-Down, the beam width is set to 20, and the number of training iterations of online learning is set to 20.

The CPU time shown in this paper is measured using Intel Xeon 3.20GHz with 32GB RAM.

4.2 Results

4.2.1 Training and Preordering Speed

Table 3 shows the training time and preordering speed together with the intrinsic evaluation metrics. In this experiment, both Top-Down and Lader were trained using the EM-100k data. Compared to Lader, Top-Down was faster: more than 20 times in training, and more than 10 times in preordering. Top-down had higher preordering accuracy in FRS and τ for en-ja. Although Lader uses sophisticated loss functions, Top-Down uses a larger number of features.

Top-Down (Basic feats.) is the top-down method using only the basic feature templates in Table 2. It was much faster but less accurate than Top-Down using the additional features. Top-Down (Basic feats.) and Lader use exactly the same features. However, there are differences in the two systems, and they had different accuracies. Top-Down uses the beam search-based top-down method for parsing and the Passive-Aggressive algorithm for parameter estimation, and Lader uses the CYK algorithm with cube pruning and an on-

[6] lader 0.1.4. http://www.phontron.com/lader/

line SVM algorithm. Especially, Lader optimizes FRS in the default setting, and it may be the reason that Lader had higher FRS.

4.2.2 Performance of Preordering for Various Training Data

Table 4 shows the preordering accuracy and BLEU scores when Top-Down was trained with various data. The best BLEU score for Top-Down was obtained by using manually annotated data for en-ja and 100k forced-decoding data for ja-en. The performance was improved by increasing the data size.

4.2.3 End-to-End Evaluation for Various Language Pairs

Table 5 shows the BLEU score of each system for 12 language pairs. Some blank fields mean that the results are unavailable due to the lack of rules or dependency parsers. For all the language pairs, Top-Down had higher BLEU scores than Lader. For ja-en and ur-en, using Forced-100k instead of EM-100k for Top-Down improved the BLEU scores by more than 0.6, but it did not always improved.

Manual-Rules performed the best for en-ja, but it needs manually created rules and is difficult to be applied to many language pairs. Auto-Rules and Classifier had higher scores than No-Preordering except for fr-en, but cannot be applied to the languages with no available dependency parsers. Top-Down (Forced-100k) can be applied to any language, and had statistically significantly better BLEU scores than No-Preordering, Manual-Rules, Auto-Rules, Classifier and Lader for 7 language pairs (en-fr, fr-en, hi-en, ja-en, ko-en, tr-en and ur-en), and similar performance for other language pairs except for en-ja, without dependency parsers trained with manually annotated data.

In all the experiments so far, the decoder was allowed to reorder even after preordering was carried out. In order to see the performance without reordering after preordering, we conducted experiments by setting the distortion limit to 0. Table 6 shows the results. The effect of the distortion limits varies for language pairs and preordering methods. The BLEU scores of Top-Down were not affected largely even when relying only on preordering.

5 Conclusion

In this paper, we proposed a top-down BTG parsing method for preordering. The method incrementally builds parse trees by splitting larger spans into smaller ones. The method provides an easy way to check the validity of each parser state, which allows us to use early update for latent variable Perceptron with beam search. In the experiments, it was shown that the top-down parsing method is more than 10 times faster than a CYK-based method. The top-down method had better BLEU scores for 7 language pairs without relying on supervised syntactic parsers compared to other preordering methods. Future work includes developing a bottom-up BTG parser with latent variables, and comparing the results to the top-down parser.

References

Alexandra Antonova and Alexey Misyurev. 2011. Building a Web-Based Parallel Corpus and Filtering Out Machine-Translated Text. In *Proceedings of the 4th Workshop on Building and Using Comparable Corpora: Comparable Corpora and the Web*, pages 136–144.

Alexandra Birch, Miles Osborne, and Phil Blunsom. 2010. Metrics for MT Evaluation: Evaluating Reordering. *Machine Translation*, 24(1):15–26.

Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw. 1998. Pattern matching for permutations. *Information Processing Letters*, 65(5):277–283.

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.

David Chiang. 2007. Hierarchical Phrase-Based Translation. *Computational Linguistics*, 33(2):201–228.

Michael Collins and Brian Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 111–118.

Michael Collins, Philipp Koehn, and Ivona Kucerova. 2005. Clause Restructuring for Statistical Machine Translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 531–540.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585.

John DeNero and Jakob Uszkoreit. 2011. Inducing Sentence Structure from Parallel Corpora for Reordering. In *Proceedings of the 2011 Conference on*

Empirical Methods in Natural Language Processing, pages 193–203.

Yoav Freund and Robert E. Schapire. 1999. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296.

Kuzman Ganchev and Mark Dredze. 2008. Small Statistical Models by Random Feature Mixing. In *Proceedings of the ACL-08: HLT Workshop on Mobile Language Processing*, pages 19–20.

Dmitriy Genzel. 2010. Automatically Learning Source-side Reordering Rules for Large Scale Machine Translation. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 376–384.

Yoav Goldberg and Michael Elhadad. 2010. An Efficient Algorithm for Easy-first Non-directional Dependency Parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured Perceptron with Inexact Search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151.

Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. 2010. Automatic Evaluation of Translation Quality for Distant Language Pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 944–952.

Laura Jehl, Adrià de Gispert, Mark Hopkins, and Bill Byrne. 2014. Source-side Preordering for Translation using Logistic Regression and Depth-first Branch-and-Bound Search. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 239–248.

Maurice G. Kendall. 1938. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93.

Maxim Khalilov and Khalil Sima'an. 2010. Source reordering using MaxEnt classifiers and supertags. In *Proceedings of the 14th Annual Conference of the European Association for Machine Translation*.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical Phrase-Based Translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 48–54.

Philipp Koehn. 2004. Statistical Significance Tests for Machine Translation Evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple Semi-supervised Dependency Parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 595–603.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289.

Uri Lerner and Slav Petrov. 2013. Source-Side Classifier Preordering for Machine Translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 513–523.

Chi-Ho Li, Minghui Li, Dongdong Zhang, Mu Li, Ming Zhou, and Yi Guan. 2007. A Probabilistic Approach to Syntax-based Reordering for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 720–727.

Ji Ma, Jingbo Zhu, Tong Xiao, and Nan Yang. 2013. Easy-First POS Tagging and Dependency Parsing with Beam Search. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–114.

Wolfgang Macherey, Franz Och, Ignacio Thayer, and Jakob Uszkoreit. 2008. Lattice-based Minimum Error Rate Training for Statistical Machine Translation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 725–734.

Valerio Antonio Miceli Barone and Giuseppe Attardi. 2013. Pre-Reordering for Machine Translation Using Transition-Based Walks on Dependency Parse Trees. In *Proceedings of the 8th Workshop on Statistical Machine Translation*, pages 164–169.

Hwidong Na and Jong-Hyeok Lee. 2013. A Discriminative Reordering Parser for IWSLT 2013. In *Proceedings of the 10th International Workshop for Spoken Language Translation*, pages 83–86.

Graham Neubig, Taro Watanabe, and Shinsuke Mori. 2012. Inducing a Discriminative Parser to Optimize Machine Translation Reordering. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 843–853.

Joakim Nivre. 2004. Incrementality in Deterministic Dependency Parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57.

Franz Josef Och and Hermann Ney. 2004. The Alignment Template Approach to Statistical Machine Translation. *Computational Linguistics*, 30(4):417–449.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318.

Kenji Sagae and Alon Lavie. 2005. A Classifier-Based Parser with Linear Run-Time Complexity. In *Proceedings of the 9th International Workshop on Parsing Technology*, pages 125–132.

Francesco Sartorio, Giorgio Satta, and Joakim Nivre. 2013. A Transition-Based Dependency Parser Using a Dynamic Parsing Strategy. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 135–144.

Xu Sun, Takuya Matsuzaki, Daisuke Okanohara, and Jun'ichi Tsujii. 2009. Latent Variable Perceptron Algorithm for Structured Classification. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1236–1242.

Oscar Täckström, Dipanjan Das, Slav Petrov, Ryan McDonald, and Joakim Nivre. 2013. Token and Type Constraints for Cross-Lingual Part-of-Speech Tagging. *Transactions of the Association of Computational Linguistics*, 1:1–12.

David Talbot, Hideto Kazawa, Hiroshi Ichikawa, Jason Katz-Brown, Masakazu Seno, and Franz J. Och. 2011. A Lightweight Evaluation Framework for Machine Translation Reordering. In *Proceedings of the 6th Workshop on Statistical Machine Translation*, pages 12–21.

Christoph Tillman. 2004. A Unigram Orientation Model for Statistical Machine Translation. In *Proceedings of the 2004 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (Short Papers)*, pages 101–104.

Roy Tromble and Jason Eisner. 2009. Learning Linear Ordering Problems for Better Translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1007–1016.

Jakob Uszkoreit, Jay M. Ponte, Ashok C. Popat, and Moshe Dubiner. 2010. Large Scale Parallel Document Mining for Machine Translation. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1101–1109.

Karthik Visweswariah, Jiri Navratil, Jeffrey Sorensen, Vijil Chenthamarakshan, and Nandakishore Kambhatla. 2010. Syntax Based Reordering with Automatically Derived Rules for Improved Statistical Machine Translation. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1119–1127.

Karthik Visweswariah, Rajakrishnan Rajkumar, Ankur Gandhe, Ananthakrishnan Ramanathan, and Jiri Navratil. 2011. A Word Reordering Model for Improved Machine Translation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 486–496.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. HMM-based Word Alignment in Statistical Translation. In *Proceedings of the 16th Conference on Computational Linguistics*, pages 836–841.

Dekai Wu. 1997. Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora. *Computational Linguistics*, 23(3):377–403.

Fei Xia and Michael McCord. 2004. Improving a Statistical MT System with Automatically Learned Rewrite Patterns. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 508–514.

Peng Xu, Jaeho Kang, Michael Ringgaard, and Franz Och. 2009. Using a Dependency Parser to Improve SMT for Subject-Object-Verb Languages. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 245–253.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 195–206.

Nan Yang, Mu Li, Dongdong Zhang, and Nenghai Yu. 2012. A Ranking-based Approach to Word Reordering for Statistical Machine Translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 912–920.

Heng Yu, Liang Huang, Haitao Mi, and Kai Zhao. 2013. Max-Violation Perceptron and Forced Decoding for Scalable MT Training. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1112–1123.

Richard Zens and Hermann Ney. 2006. Discriminative Reordering Models for Statistical Machine Translation. In *Proceedings on the Workshop on Statistical Machine Translation*, pages 55–63.

Yue Zhang and Joakim Nivre. 2011. Transition-based Dependency Parsing with Rich Non-local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Short Papers*, pages 188–193.