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...
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.

### 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 Φ(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:

$$\hat{z} = \arg\max_{z \in Z(x)} \sum_{m \in Nodes(z)} \Lambda \cdot \Phi(m), \quad (1)$$

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

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-

¹Although 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.

²There may be more than one BTG tree which represents the same word reordering (e.g., the word reordering $C_A B_B A_1$ to $A_1 B_2 C_B$ has two possible BTG trees), and there are permutations which cannot be represented with BTG (e.g., $B_2 D_A A_1 C_B$ to $A_1 B_2 C_B D_A$, which is called the 2413 pattern).
Figure 3: Top-down BTG parsing.

```
(0) ⟨[[0, 5]], [], 0⟩
(1) ⟨[[0, 2], [2, 5]], [(2, S)], v1⟩
(2) ⟨[[0, 2]], [(2, S), (3, I)], v2⟩
(3) ⟨[[0, 2]], [(2, S), (3, I), (4, I)], v3⟩
(4) ⟨[[2, S], (3, I), (4, I), (1, S)], v4⟩
```

Table 1: Parser states in top-down parsing.

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

```
Input: Sentence $x$, feature weights $\Lambda$, beam width $k$.
Output: BTG parse tree.
1: $S_0 \leftarrow \{[[], [0, 0]]\}$ // Initial state.
2: for $i \leftarrow 1, \ldots, |x| - 1$ do
3: $S \leftarrow \{\}$ // Set of the next states.
4: foreach $s \in S_{i-1}$ do
5: $S \leftarrow S \cup \tau_{x, \Lambda}(s)$ // Generate next states.
6: $S_i \leftarrow Top_k(S)$ // Select $k$-best states.
7: $s = \text{argmax}_{s \in S_{i-1}} \text{Score}(s)$
8: return $\text{Tree}(s)$
```

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 $(P, C, v)$. $P$ 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 $o \in \{S, I\}$, 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 $([[], []], [0], 0)$, 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 $([[], [c_0, \cdots, c_{|x|-2}], v])$, 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.

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. $\text{Score}(s)$ returns the score of the state $s$, and $\text{Tree}(s)$ 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 $\Lambda$ 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 exam-
ple, 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.\footnote{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.}
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 an-
notated 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) to-
gether 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 pars-
ing with multiple correct actions (Goldberg and
Elhadad, 2010; Sartorio et al., 2013).

Figure 6 describes the training algorithm.\footnote{Although the simple Perceptron algorithm is used for ex-
planation, we actually used the Passive Aggressive algorithm
(Crammer et al., 2006) with the parameter averaging tech-
nique (Freund and Schapire, 1999).} $\Phi(x, s)$ is the feature vector for all the nodes in
the partial parse tree at the state $s$, and $\tau_{x, y}(s)$ 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 incre-
mental parsing if a correct state falls off the beam,
and there is no possibility to obtain a correct out-
put. 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 up-
date, we need to check the validity of each parser
state. We extend the parser state to the four tu-
ple $(P, A, v, w)$, where $w \in \{\text{true}, \text{false}\}$ is the
validity of the state. We remove training exam-
pies which cannot be represented with BTG be-
forehand and set $w$ of the initial state to true. The
function $\text{Valid}(s)$ in Figure 6 returns the validity
of state $s$. One advantage of the top-down pars-
ing algorithm is that it is easy to track the validity
of each state. The validity of a state can be cal-
culated 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:

\[
\forall i \in \{p, \cdots, r - 1\} y_i = -1 \lor \\
\forall i \in \{r, \cdots, q - 1\} y_i = -1 \lor \\
\left( o = S \land \max_{y_i \neq -1} y_i \leq \min_{y_i \neq -1} y_i \right) \lor \\
\left( o = I \land \max_{y_i \neq -1} y_i \leq \min_{y_i \neq -1} y_i \right). \tag{3}
\]

**Proof.** Let $\pi_i$ denote the position of $x_i$ after re-
ordering by BTG parsing. If Condition (3) does
not hold, there are $i$ and $j$ which satisfy $\pi_i <
\pi_j \land y_i > y_j \land y_i \neq -1 \land y_j \neq -1$, and $\pi_i$ and $\pi_j$
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 am-
biguities. 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 separ-
able permutation if and only if it contains nei-
ther the 2413 nor the 3142 patterns. Since $s$ is
valid, $y$ is a separable permutation. $y$ does not con-
tain the 2413 nor the 3142 patterns, and any sub-
sequence 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 ambigu-
ities (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.,
$y' = 2, 1, 0, 0, 0, 1, 1$ 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
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 BTTG 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 \(\min_{i=p, \ldots, r} x_i, \max_{i=p, \ldots, r} x_i\) and \(\min_{i=r, \ldots, q} x_i, \max_{i=r, \ldots, q} x_i\) 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 \(x_i^t\) 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, \(d \in \{\text{left}, \text{right}\}\) 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 \(\{<, =, >\}\) 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.

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 has \(y\) and \(y'\) which represents correct word positions after reordering. Furthermore, 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 source-side words which are aligned to the target-side word \(x_i\). We define an order relation between two words:

\[ x_i \leq x_j \iff \forall a \in A_i \setminus A_j, \forall b \in A_j a \leq b \land \forall a \in A_i, \forall b \in A_j \setminus A_i a \leq b. \]
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. In-frequent 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.

| Table 2: Feature templates. |
|-----------------------------|
| Baseline Feature Template |
| \( o(q - p), oBalance(p, q, r), \) |
| \( ox_{p-1}, oxw_{p-1}, ox_{q-1}, oxw_{q-1}, ox_{r-1}, oxw_{r-1}, ox_{p'}_{r}, oxw_{p'}_{r}, ox_{q'}, \) |
| \( ox_{p}, oxw_{p}, ox_{q}, oxw_{q}, ox_{r}, oxw_{r}, ox_{p'}, oxw_{p'}, ox_{q'}, \) |
| \( ox_{p+1}, oxw_{p+1}, ox_{q+1}, oxw_{q+1}, ox_{r+1}, oxw_{r+1}, ox_{p''}, oxw_{p''}, \) |
| \( ox_{p''}, oxw_{p''}, ox_{q''}, oxw_{q''}, ox_{r''}, oxw_{r''}, ox_{p'''}, oxw_{p'''}, \) |
| \( ox_{p'''}, oxw_{p'''}, ox_{q'''}, \) |
| Additional Feature Template |
| \( o min(r - p, 3) min(q - r, 3), oo, oo' d, \) |
| \( ox_{p-1}', oxw_{p-1}', ox_{p}, oxw_{p}, ox_{p+1}, oxw_{p+1}, ox_{p''}, oxw_{p''}, ox_{q}, oxw_{q}, ox_{q+1}, oxw_{q+1}, ox_{q''}, oxw_{q''}, \) |
| \( ox_{p''}, oxw_{p''}, ox_{q''}, oxw_{q''}, ox_{r'}, oxw_{r'}, ox_{r''}, oxw_{r''}, ox_{r'''}, oxw_{r'''}, \) |
| \( ox_{p'''}, oxw_{p'''}, ox_{q'''}, \) |
| \( oo' d_{x_{i+1}}, oo' d_{x_i}, oo' d_{x_{i+1}}', oo' d_{x_{i+1}}'' \) |
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 \( \tau \) are calculated as follows:

\[
FRS = \frac{B}{|\rho| + 1},
\]

(5)

\[
B = \sum_{i=0}^{\left|\rho\right|-2} \delta(y_{\rho_i} = y_{\rho_{i+1}}) + \delta(y_{\rho_0} = 0) + \delta(y_{\rho_{|\rho|-1}} = \max_i y_i),
\]

(6)

\[
\tau = \frac{\sum_{i=0}^{\left|\rho\right|-2} \sum_{j=i+1}^{\left|\rho\right|-1} \delta(y_{\rho_i} \leq y_{\rho_j})}{\frac{1}{2}|\rho|(|\rho| - 1)},
\]

(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 \( \tau \) 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 et al., 2004).
- **Auto-Rules** This system uses the preordering method based on automatically created rules (Xu et al., 2004).
- **Classifier** This system uses a classifier to predict the best preordering.

\(^5\)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.
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. 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 pre-ordering. Top-down had higher preordering accuracy in FRS and \( \tau \) 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-
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
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 Okano, 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.