Transition-Based Syntactic Linearization

Yijia Liu †‡, Yue Zhang †, Wanxiang Che ‡, Bing Qin ‡
†Singapore University of Technology and Design
‡Research Center for Social Computing and Information Retrieval
Harbin Institute of Technology, China
{yjliu,car,bqin}@ir.hit.edu.cn yue_zhang@sutd.edu.sg

Abstract

Syntactic linearization algorithms take a bag of input words and a set of optional constraints, and construct an output sentence and its syntactic derivation simultaneously. The search problem is NP-hard, and the current best results are achieved by bottom-up best-first search. One drawback of the method is low efficiency; and there is no theoretical guarantee that a full sentence can be found within bounded time. We propose an alternative algorithm that constructs output structures from left to right using beam-search. The algorithm is based on incremental parsing algorithms. We extend the transition system so that word ordering is performed in addition to syntactic parsing, resulting in a linearization system that runs in guaranteed quadratic time. In standard evaluations, our system runs an order of magnitude faster than a state-of-the-art baseline using best-first search, with improved accuracies.

1 Introduction

Linearization is the task of ordering a bag of words into a grammatical and fluent sentence. Syntax-based linearization algorithms generate a sentence along with its syntactic structure. Depending on how much syntactic information is available as inputs, recent work on syntactic linearization can be classified into *free word ordering* (Wan et al., 2009; Zhang et al., 2012; de Gispert et al., 2014), which orders a bag of words without syntactic constraints, *full tree linearization* (He et al., 2009; Bohnet et al., 2010; Song et al., 2014), which orders a bag of words given a full-spanning syntactic tree, and *partial tree linearization* (Zhang, 2013), which orders a bag of words given some syntactic relations between them as partial constraints.

Initial State	$([\,], [1...n], \emptyset)$
Final State	$([\,], [\,], A)$
Induction Rules:	
SHIFT	$\frac{(\sigma, [i\|\beta], A)}{([\sigma\| \ i], \beta, A)}$
LEFTARC	$\frac{([\sigma\|j \ i], \beta, A)}{([\sigma\|i], \beta, A \cup \{j \leftarrow i\})}$
RIGHTARC	$\frac{([\sigma\|j \ i], \beta, A)}{([\sigma\|j], \beta, A \cup \{j \rightarrow i\})}$

Figure 1: The *arc-standard* parsing algorithm.

The search space for syntactic linearization is huge. Even with a full syntax tree being available as constraints, permutation of nodes on each level is an NP-hard problem. As a result, heuristic search has been adopted by most previous work, and the best results have been achieved by a time-constrained best-first search framework (White, 2004a; White and Rajkumar, 2009; Zhang and Clark, 2011b; Song et al., 2014). Though empirically highly accurate, one drawback of this approach is that there is no asymptotic upper bound on the time complexity of finding the first full sentence. As a result, it can take 5–10 seconds to process a sentence, and sometimes fail to yield a full sentence at timeout. This issue is more severe for larger bags of words, and makes the algorithms practically less useful.

We study the effect of an alternative learning and search framework for the linearization prob-

Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL, pages 113–122,
Denver, Colorado, May 31 – June 5, 2015.

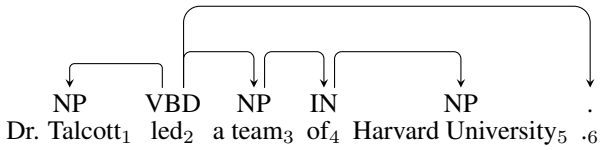

Figure 2: Example dependency tree.

lem, which has a theoretical upper bound on the time complexity, and always yields a full sentence in quadratic time. Our method is inspired by the connection between syntactic linearization and syntactic parsing: both build a syntactic tree over a sentence, with the former performing word ordering in addition to derivation construction. As a result, syntactic linearization can be treated as a generalized form of parsing, for which there is no input word order, and therefore extensions to parsing algorithms can be used to perform linearization.

For syntactic parsing, the algorithm of Zhang and Nivre (2011) gives competitive accuracies under linear complexity. Compared with parsers that use dynamic programming (McDonald and Pereira, 2006; Koo and Collins, 2010), the efficient beam-search system is more suitable for the NP-hard linearization task. We extend the parser of Zhang and Nivre (2011), so that word ordering is performed in addition to syntactic tree construction. Experimental results show that the transition-based linearization system runs an order of magnitude faster than a state-of-the-art best-first baseline, with improved accuracies in standard evaluation. Our linearizer is publicly available under GPL at `http://sourceforge.net/projects/zgen/`.

2 Transition-Based Parsing

The task of *dependency parsing* is to find a *dependency tree* given an input sentence. Figure 2 shows an example dependency tree, which consists of *dependency arcs* that represent syntactic relations between pairs of words. A transition-based dependency parsing algorithm (Nivre, 2008) can be formalized as a *transition system*, $S = (C, T, c_s, C_t)$, where C is the set of states, T is a set of transition actions, c_s is the initial state and C_t is a set of terminal states. The parsing process is modeled as an application of a sequence of actions, transducing the initial state into a final state, while constructing dependency arcs. Each state in the transition system can be formalized as a tuple (σ, β, A), where σ is a stack that maintains a partial derivation, β is a buffer of incoming input words and A is the set of dependency relations that have been built.

	Transition	σ	β	A
0		[]	[1...6]	$\emptyset$
1	SHIFT	[1]	[2...6]	
2	SHIFT	[1 2]	[3...6]	
3	SHIFT	[1 2 3]	[4...6]	
4	SHIFT	[1 2 3 4]	[5,6]	
5	SHIFT	[1 2 3 4 5]	[6]	
6	RIGHTARC	[1 2 3 4]	[6]	$A \cup \{4 \rightarrow 5\}$
7	RIGHTARC	[1 2 3]	[6]	$A \cup \{3 \rightarrow 4\}$
8	RIGHTARC	[1 2]	[6]	$A \cup \{2 \rightarrow 3\}$
9	SHIFT	[1 2 6]	[]	
10	RIGHTARC	[1 2]	[]	$A \cup \{2 \rightarrow 6\}$
11	LEFTARC	[2]	[]	$A \cup \{1 \leftarrow 2\}$

Table 1: *arc-standard* transition action sequence for parsing the sentence in Figure 2.

Our work is based on the *arc-standard* algorithm (Nivre, 2008). The deduction system of the *arc-standard* algorithm is shown in Figure 1. In this system, three transition actions are used: LEFTARC, RIGHTARC and SHIFT. Given a state $s = ([\sigma|\ j\ i], [k|\beta], A)$,

- LEFTARC builds an arc $\{j \leftarrow i\}$ and pops j off the stack.
- RIGHTARC builds an arc $\{j \rightarrow i\}$ and pops i off the stack.
- SHIFT removes the front word k from the buffer β, and shifts it onto the stack.

In the notations above, i, j and k are word indices of an input sentence. The *arc-standard* system assumes that each input word has been assigned a part-of-speech (POS) tag.

The sentence in Figure 2 can be parsed by the transition sequence shown in Table 1. Given an input sentence of n words, the algorithm takes $2n$ transitions to construct an output, because each word needs to be shifted onto the stack once and popped off once before parsing finishes, and all the transition actions are either shifting or popping actions.

Initial State	$([\,], set(1..n), \emptyset)$
Final State	$([\,], \emptyset, A)$

Induction Rules:

$$\textsc{Shift}\text{-}i\text{-}\textsc{pos}\quad \frac{(\sigma, \rho, A)}{([\sigma|i], \rho - \{i\}, A)}$$

$$\textsc{LeftArc}\quad \frac{([\sigma|j\ i], \rho, A)}{([\sigma|i], \rho, A \cup \{j \leftarrow i\})}$$

$$\textsc{RightArc}\quad \frac{([\sigma|j\ i], \rho, A)}{([\sigma|j], \rho, A \cup \{j \rightarrow i\})}$$

Figure 3: Deduction system for transition-based linearization. Indices i, j do not reflect word order.

Algorithm 1: transition-based linearization

Input: C, a set of input syntactic constraints
Output: The highest-scored final state

```
candidates ← ([ ], set(1..n), ∅)
agenda ← ∅
for i ← 1..2n do
    for s in candidates do
        for action in GETPOSSIBLEACTIONS(s, C) do
            agenda ← APPLY(s, action)
    candidates ← TOP-K(agenda)
    agenda ← ∅
best ← BEST(candidates)
return best
```

3 Transition-Based Linearization

The main difference between linearization and dependency parsing is that the input words are unordered for linearization, which results in an unordered buffer ρ. At a certain state $s = (\sigma, \rho, A)$, any word in the buffer ρ can be shifted onto the stack. In addition, unlike a parser, the vanilla linearization task does not assume that input words are assigned POS. To extend the *arc-standard* algorithm for linearization, we incorporate word and POS into the SHIFT operation, transforming the *arc-standard* SHIFT operation to SHIFT-*Word-POS*, which selects the word *Word* from the buffer ρ, tags it with *POS* and shifts it onto the stack. Since the order of words in an output sentence equals to the order in which they are shifted onto the stack, word ordering is performed along with the parsing process.

Under such extension, the sentence in Figure 2 can be generated by the transition sequence (SHIFT-*Dr. Talcott*-NP, SHIFT-*led*-VBD, SHIFT-*of*-NP, SHIFT-*a team*-NP, SHIFT-*of*-IN, SHIFT-*Harvard University*-NP, RIGHTARC, RIGHTARC, RIGHTARC, SHIFT-.-., RIGHTARC, LEFTARC), given the unordered bag of words (*Dr. Talcott*, *led*, *a team*, *of*, *Harvard University*, .).

The deduction system for the linearization algorithm is shown in Figure 3. Given an input bag of n words, this algorithm also takes $2n$ transition actions to construct an output, by the same reason as the *arc-standard* parser.

3.1 Search and Learning

We apply the learning and search framework of Zhang and Clark (2011a), which gives state-of-the-art transition-based parsing accuracies and runs in linear time (Zhang and Nivre, 2011). Pseudocode of the search algorithm is shown in Algorithm 1. It performs beam-search by using an agenda to keep the *k*-best states at each incremental step. When decoding starts, the agenda contains only the initial state. At each step, each state in the agenda is advanced by applying all possible transition actions (GETPOSSIBLEACTIONS), leading to a set of new states. The k best are selected for the new states, and used to replace the current states in the agenda, before the next decoding step starts. Given an input bag of n words, the process repeats for $2n$ steps, after which all the states in the agenda are terminal states, and the highest-scored state in the agenda is taken for the final output. The complexity of this algorithm is n^2, because it takes a fixed $2n$ steps to construct an output, and in each step the number of possible SHIFT action is proportional to the size of ρ.

The search algorithm ranks search hypotheses, which are sequences of state transitions, by their scores. A global linear model is used to score search hypotheses. Given a hypothesis h, its score is calculated by:

$$Score(h) = \Phi(h) \cdot \vec{\theta},$$

where $\vec{\theta}$ is the parameter vector of the model and $\Phi(h)$ is the global feature vector of h, extracted by instantiating the feature templates in Table 2 according to each state in the transition sequence.

In the table, S_0 represents the first word on the top of the stack, S_1 represents the second word on the top of the stack, w represents a word and p rep-

Unigrams
S_0w; S_0p; $S_{0,l}w$; $S_{0,l}p$; $S_{0,r}w$; $S_{0,r}p$;
$S_{0,l2}w$; $S_{0,l2}p$; $S_{0,r2}w$; $S_{0,r2}p$;
S_1w; S_1p; $S_{1,l}w$; $S_{1,l}p$; $S_{1,r}w$; $S_{1,r}p$;
$S_{1,l2}w$; $S_{1,l2}p$; $S_{1,r2}w$; $S_{1,r2}p$;
Bigram
$S_0wS_{0,l}w$; $S_0wS_{0,l}p$; $S_0pS_{0,l}w$; $S_0pS_{0,l}p$;
$S_0wS_{0,r}w$; $S_0wS_{0,r}p$; $S_0pS_{0,r}w$; $S_0pS_{0,r}p$;
$S_1wS_{1,l}w$; $S_1wS_{1,l}p$; $S_1pS_{1,l}w$; $S_1pS_{1,l}p$;
$S_1wS_{1,r}w$; $S_1wS_{1,r}p$; $S_1pS_{1,r}w$; $S_1pS_{1,r}p$;
S_0wS_1w; S_0wS_1p; S_0pS_1w; S_0pS_1p
Trigram
$S_0wS_0pS_{0,l}w$; $S_0wS_{0,l}wS_{0,l}p$; $S_0wS_0pS_{0,l}p$;
$S_0pS_{0,l}wS_{0,l}p$; $S_0wS_0pS_{0,r}w$; $S_0wS_{0,l}wS_{0,r}p$;
$S_0wS_0pS_{0,r}p$; $S_0pS_{0,r}wS_{0,r}p$;
$S_1wS_1pS_{1,l}w$; $S_1wS_{1,l}wS_{1,l}p$; $S_1wS_1pS_{1,l}p$;
$S_1pS_{1,l}wS_{1,l}p$; $S_1wS_1pS_{1,r}w$; $S_1wS_{1,l}wS_{1,r}p$;
$S_1wS_1pS_{1,r}p$; $S_1pS_{1,r}wS_{1,r}p$;
Linearizion
w_0; p_0; $w_{-1}w_0$; $p_{-1}p_0$; $w_{-2}w_{-1}w_0$; $p_{-2}p_{-1}p_0$;
$S_{0,l}wS_{0,l2}w$; $S_{0,l}pS_{0,l2}p$; $S_{0,r2}wS_{0,r}w$; $S_{0,r2}pS_{0,r}p$;
$S_{1,l}wS_{1,l2}w$; $S_{1,l}pS_{1,l2}p$; $S_{1,r2}wS_{1,r}w$; $S_{1,r2}pS_{1,r}p$;

Table 2: Feature templates.

resent a POS-tag. The feature templates can be classified into four types: *unigram*, *bigram*, *trigram* and *linearization*. The first three types are taken from the dependency parser of Zhang and Nivre (2011), which capture context information for S_0, S_1 and their modifiers. The original feature templates of Zhang and Nivre (2011) also contain information of the front words on the buffer. However, since the buffer is unordered for linearization, we do not include these features.

The *linearization* feature templates are specific for linearization, and captures surface ngram information. Each search state represents a partially linearized sentence. We represents the last word in the partially linearized sentence as w_0 and the second last as w_{-1}.

Given a set of labeled training examples, the averaged perceptron (Collins, 2002) with early update (Collins and Roark, 2004; Zhang and Nivre, 2011) is used to train the parameters $\vec{\theta}$ of the model.

3.2 Input Syntactic Constraints

The use of syntactic constraints to achieve better linearization performance has been studied in previous work. Wan et al. (2009) employ POS constraints in learning a dependency language model. Zhang and Clark (2011b) take supertags as constraints to a CCG linearizer. Zhang (2013) demonstrates the possibility of *partial-tree linearization*, which allows a whole spectrum of input syntactic constraints. In practice, input syntactic constraints, including POS and dependency relations, can be obtained from earlier stage of a generation pipeline, such as lexical transfer results in machine translation.

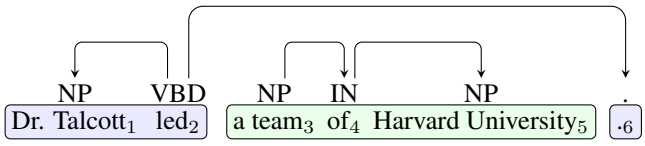

Figure 4: Example partial tree. Words in the same sub dependency trees are grouped by rounded boxes. Word indices do not specify their orders. Base phrases (e.g. *Dr. Talcott*) are treated as single words.

It is relatively straightforward to apply input constraints to a best-first system (Zhang, 2013), but less so for beam-search. In this section, we utilize the input syntactic constraints by letting the information decide the possible actions for each state, namely the return value of GETPOSSIBLEACTIONS in Algorithm 1, thus, when input POS-tags and dependencies are given, the generation system can achieve more specified outputs.

3.2.1 POS Constraints

POS is the simplest form of constraints to the transition-based linearization system. When the POS of an input word is given, the POS-tag component in SHIFT-*Word-POS* operation is fixed, and the number of SHIFT actions for the word is reduced from the number of all POS to 1.

3.2.2 Partial Tree Constraints

In partial tree linearization, a set of dependency arcs that form a partial dependency tree is given to the linearization system as input constraints. Figure 4 illustrate an example. The search space can be reduced by ignoring the transition sequences that do not result in a dependency tree that is consistent with the input constraints. Take the partial tree in Figure 4 for example. At the state s = ([*Harvard University*$_5$], set(1..n)-{5}, $\emptyset$), it is illegal to shift the base phrase *a team*$_3$ onto the stack, be-

Algorithm 2: GETPOSSIBLEACTIONS for partial tree linearization, where C is a partial tree

Input: A state $s = ([\sigma|j\ i], \rho, A)$ and partial tree C
Output: A set of possible transition actions T

```
1  if s.σ is empty then
2      for k ∈ s.ρ do
3          T ← T ∪ (SHIFT, POS, k)
4  else
5      if REDUCABLE(s, i, j, C) then
6          T ← T ∪ (LEFTARC)
7      if REDUCABLE(s, j, i, C) then
8          T ← T ∪ (RIGHTARC)
9      for k ∈ s.β do
10         if SHIFTLEGAL(s, k, C) then
11             T ← T ∪ (SHIFT, POS, k)
12 return T
```

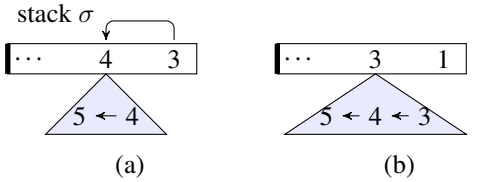

Figure 5: Two conditions for a valid LEFTARC action in partial-tree linearization. The indices correspond to those in Figure 4. A shaded triangle represents the readily built arcs under a root word.

cause this action will result in a sub-sequence (*Harvard University*$_5$, *a team*$_3$, *of*$_4$), which cannot have the dependency arcs $\{3 \rightarrow 4\}, \{4 \rightarrow 5\}$ by using *arc-standard* actions.

Algorithm 3 shows pseudocode of GETPOSSIBLEACTIONS when C is a partial tree. Given a state $s = ([\sigma|j\ i], \rho, A)$ the LEFTARC action builds an arc $\{j \leftarrow i\}$ and pops the word j off the stack. Since the popped word j cannot be linked to any words in future transitions, all the descendants of j should have been processed and removed from the stack. In addition, constrained by the given partial tree, the arc $\{j \leftarrow i\}$ should be an arc in C (Figure 5a), or j should be the root of a sub dependency tree in C (Figure 5b). We denote the conditions as REDUCABLE(s, i, j, C) (lines 5-6). The case for RIGHTARC is similar to LEFTARC (lines 7-8).

For the SHIFT action, the conditions are more complex. Due to space limitation, we briefly sketch

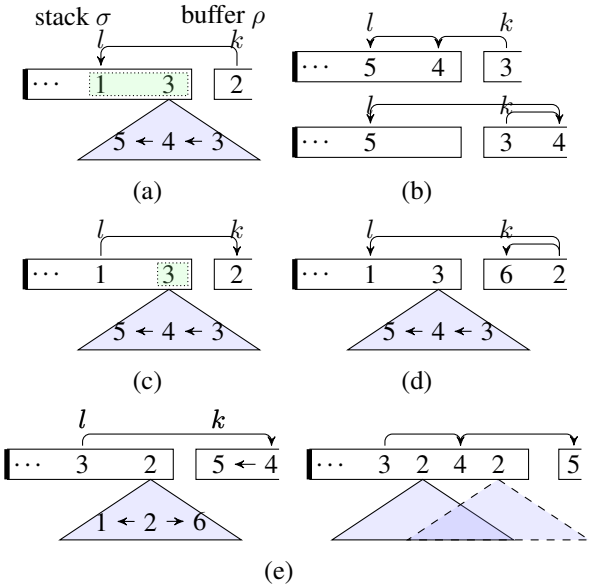

Figure 6: 5 relations between k and l. The indices correspond to those in Figure 4. The words in green boxes must have arcs with k in future transitions.

the SHIFTLEGAL function below. Detailed algorithm pseudocode for SHIFTLEGAL is given in the supplementing material. For a word k in ρ to be shifted onto the stack, all the words on the stack must satisfy certain constraints. There are 5 possible relations between k and a word l on the stack. (1) If l is a child of k in C (Figure 6a), all the words on the stack from l to the top of the stack should be *reducable to* k, because only LEFTARC can be applied between k and these words in future actions. (2) If l is a grand child of k (Figure 6b), no legal sentence can be constructed if k is shifted onto the stack. (3) If l is the parent of k (Figure 6c), legal SHIFTs require all the words on the stack from l to the top to be *reducable* to k. (4) If l is a grand parent of k, all the words on the stack from l to the top will become descendants of l in the output (Figure 6e). Thus these words must be descendants of l in C, or the root of different subdependency trees. (5) If l is a siblings of k, we denote a as the least common ancestor of k and l. a will become in the buffer and l should be a direct child of a. All the words from l to the top of the stack should be the descendants of a in the output (Figure 6d), and thus a should have the same conditions as in (4). Finally, if no word on the stack is in the same subdependency tree as k in C, then k can be safely shifted.

Algorithm 3: GETPOSSIBLEACTIONS for full tree linearization, where C is a full tree

Input: A state $s = ([\sigma|j\ i], \rho, A)$ and gold tree C
Output: A set of possible transition actions T

```
1  T ← ∅
2  if s.σ is empty then
3      for k ∈ s.ρ do
4          T ← T ∪ (SHIFT, POS, k)
5  else
6      if ∃j, j ∈ (DESCENDANTS(i) ∩ s.ρ) then
7          for j ∈ (DESCENDANTS(i) ∩ s.ρ) do
8              T ← T ∪ (SHIFT, POS, j)
9      else
10         if {j → i} ∈ C then
11             T ← T ∪ (RIGHTARC)
12         else if {j ← i} ∈ C then
13             T ← T ∪ (LEFTARC)
14         else
15             for
               k ∈ (SIBLINGS(i) ∪ HEAD(i)) ∩ s.ρ do
16             T ← T ∪ (SHIFT, POS, k)
17 return T
```

3.2.3 Full Tree Constraints

Algorithm 2 can also be used with full-tree constraints, which are a special case of partial-tree constraints. However, there is a conceptually simpler algorithm that leverages full-tree constraints. Because tree linearization is frequently studied in the literature, we describe this algorithm in Algorithm 3. When the stack is empty, we can freely move any word in the buffer ρ onto the stack (line 2-4). If not all the descendants of the stack top i have been processed, the next transition actions should move them onto the stack, so that arcs can be constructed between i and these words (line 6-8). If all the descendants of i have been processed, the next action should eagerly build arcs between top two words i and j on the stack (line 10-13). If no arc exists between i and j, the next action should shift the parent word of i or a word in i's sibling tree (line 14-16).

4 Experiments

We follow previous work and conduct experiments on the Penn Treebank (PTB), using Wall Street Journal sections 2–21 for training, 22 for development testing and 23 for final testing. Gold-standard dependency trees are derived from bracketed sentences in the treebank using Penn2Malt[1], and base noun phrases are treated as a single word (Wan et al., 2009; Zhang, 2013). The BLEU score (Papineni et al., 2002) is used to evaluate the performance of linearization, which has been adopted in former literals (Wan et al., 2009; White and Rajkumar, 2009; Zhang and Clark, 2011b) and recent shared-tasks (Belz et al., 2011). We use our implementation of the best-first system of Zhang (2013), which gives the state-of-the-art results, as the baseline.

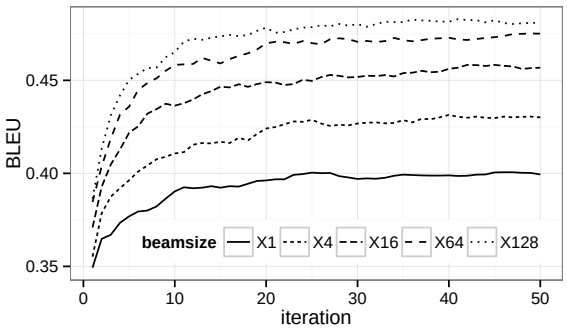

Figure 7: Dev. results with different beam sizes.

4.1 Influence of Beam size

We first study the influence of beam size by performing free word ordering on the development test data. BLEU score curves with different beam sizes are shown in Figure 7. From this figure, we can see that the systems with beam 64 and 128 achieve the best results. However, the 128-beam system does not improve the performance significantly (48.2 vs 47.5), but runs twice slower. As a result, we set the beam size to 64 in the remaining experiments.

4.2 Input Syntactic Constraints

To test the effectiveness of GETPOSSIBLEACTIONS under different input constraints, we follow Zhang (2013) and feed different amounts of POS-tags and dependencies to our transition-based linearization system. Input syntactic constraints are obtained by randomly sampling POS and dependencies from the gold dependency tree. Nine development experiments under different inputs are performed, and the

[1] http://stp.lingfil.uu.se/~nivre/research/Penn2Malt.html

	no pos no dep		50% pos no dep		all pos no dep		no pos 50% dep		50% pos 50% dep		all pos 50% dep		no pos all dep		50% pos all dep		all pos all dep	
	BL	SP	BL	SP	BL	SP	BL	SP	BL	SP	BL	SP	BL	SP	BL	SP	BL	SP
Z13	42.9	4872	43.4	4856	44.7	4826	50.5	4790	51.4	4737	52.2	4720	73.3	4600	74.7	4431	76.3	4218
Ours	47.5	155	47.9	119	48.8	74	54.8	132	55.2	91	56.2	41	77.8	40	79.1	28	81.1	22

Table 3: Partial-tree linearizion results on the development test set. *BL* – the BLEU score, *SP* – number of milliseconds to order one sentence. Z13 refers to the best-first system of Zhang (2013).

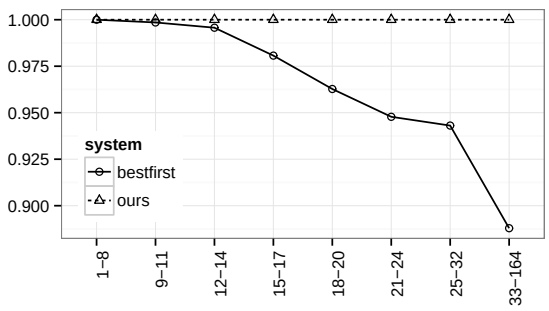

Figure 8: Comparison between transition-based and best-first systems on surface string brevity.

	Precision		Recall		F	
len	Z13	ours	Z13	ours	Z13	ours
< 5	24.63	20.45	14.56	21.82	18.3	21.11
< 10	15.20	16.33	10.59	15.88	12.48	16.1
< 15	10.82	14.73	9.38	14.08	10.05	14.4
< 30	8.18	12.54	8.26	12.43	8.22	12.49

Table 4: Precision, recall and F-score comparison on different spans lengths.

BLEU scores along with the average time to order one sentence are shown in Table 3.

With more syntactic information in the input, our linearization system achieves better performance, showing that GETPOSSIBLEACTIONS can take advantage of the input constraints and yield more specified output. In addition, because input constraints reduce the search space, the systems with more syntactic information achieve faster decoding speeds. In comparison with Zhang (2013), the transition-based system achieves improved accuracies under the settings, and the decoding speed can be over two orders of magnitude faster (22ms vs. 4218ms). We give more detailed analysis next.

4.3 Comparison with Best-First

The beam-search linearizer takes a very different search strategy compared with best-first search, which affects the error distribution. As mentioned earlier, one problem of best-first is the lack of theoretical guarantee on time complexity. As a result, a time constraint is used and default output can be constructed when no full output is found (White, 2004b; Zhang and Clark, 2011b). This may result in incomplete output sentences and intuitively, this problem is more severe for larger bag of words. In contrast, the transition-based linearization algorithm takes $|2n|$ steps to generate a sentence and thus guarantees to order all the input words. Figure 8 shows the results by comparing the brevity scores (i.e. the number of words in the output divided by the number of words in reference sentence) on different sizes of inputs. Best-search can fail to order all the input words even on bags of 9 – 11 words, and the case is more severe for larger bag of words. On the other hand, the transition-based method uses all the input words to generate output and the brevity score is constant 1. Since the BLEU score consists two parts: the n-gram precision and brevity, this comparison partly explains why the transition-based linearization algorithm achieves higher BLEU scores.

To further compare the difference between the two systems, we evaluate the qualities of projective spans, which are dependency treelets. Both systems build outputs bottom-up by constructing projective spans, and a break-down of span accuracies against span sizes shows the effects of the different search algorithms. The results are shown in Table 4. According to this table, the best-first system tends to construct smaller spans more precisely, but the recall is relatively lower. Overall, higher F-scores are achieved by the transition-based system.

During the decoding process, the best-first system compares spans of different sizes and expands

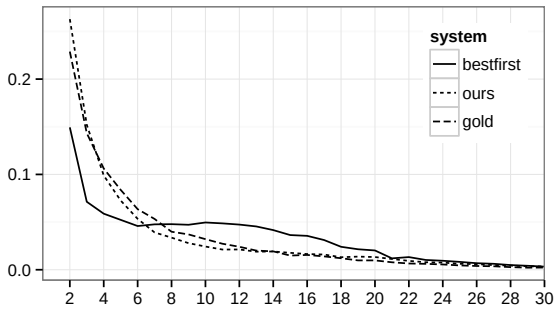

Figure 9: Distributions of spans outputted by the best-first, transition-based systems and the gold trees.

	no pos no dep	all pos no dep	all pos all dep
Wan et al. (2009)	-	33.7	-
Zhang and Clark (2011b)	-	40.1	-
Zhang et al. (2012)	-	43.8	-
Zhang (2013)	44.7	46.8	76.2
This paper	**49.4**	**50.8**	**82.3**

Table 5: Final results.

those that have higher scores. As a result, the number of expanded spans do not have a fixed correlation with the size, and there can be fewer but better small spans expanded. In contrast, the transition-based system models transition sequences rather than individual spans, and therefore the distribution of spans of different sizes in each hypothesis resembles that of the training data. Figure 9 verifies the analysis by counting the distributions of spans with respect to the length, in the search algorithms of the two systems and the gold dependency trees. The distribution of the transition-based system is closer to that of gold dependency trees, while the best-first system outputs less smaller spans and more longer ones. This explains the higher precision for the best-first system on smaller spans.

4.4 Final Results

The final results on the test set of Penn Treebank are shown in Table 5. Compared with previous studies, our transition-based linearization system achieves the best results on all the tests. Table 6 shows some example output sentences, when there are no input constraints. For longer sentences, the transition-based method gives noticeably better results.

	output	BL
ref.	There is no asbestos in our products now .	
Z13	There is no asbestos now in our products .	43.5
ours	There is now our products in no asbestos .	17.8
ref.	Previously , watch imports were denied such duty-free treatment .	
Z13	such duty-free treatment Previously , watch imports were denied .	67.6
ours	Previously , watch imports were denied such duty-free treatment .	100
ref.	Despite recent declines in yields , investors continue to pour cash into money funds .	
Z13	continue yields investors pour to recent declines in cash , into money funds	20.1
ours	Despite recent declines in yields into money funds , investors continue to pour cash .	67.0

Table 6: Example outputs.

5 Related Work

The input to practical natural language generation (NLG) system (Reiter and Dale, 1997) can range from a bag of words and phrases to a bag of lemmas without punctuation (Belz et al., 2011). The linearization module of this paper can serve as the final stage in a pipeline when the bag of words and their optional syntactic information are given. There has also been work to jointly perform linearization and morphological generation (Song et al., 2014).

There has been work on linearization with unlabeled and labeled dependency trees (He et al., 2009; Zhang, 2013). These methods mostly use greedy or best-first algorithms to order each tree node. Our work is different by performing word ordering using a transition process.

Besides dependency grammar, linearization with other syntactic grammars, such as CFG and CCG (White and Rajkumar, 2009; Zhang and Clark, 2011b), has also been studied. In this paper, we adopt the dependency grammar for transition-based linearization. However, since transition-based parsing algorithms has been successfully applied to different grammars, including CFG (Sagae et al., 2005) and CCG (Xu et al., 2014), our linearization method can be applied to these grammars.

6 Conclusion

We studied transition-based syntactic linearization as an extension to transition-based parsing. Compared with best-first systems, the advantage of our transition-based algorithm includes bounded time complexity, and the guarantee to yield full sentences when given a bag of words. Experimental results show that our algorithm achieves improved accuracies, with significantly faster decoding speed compared with a state-of-the-art best-first baseline. We publicly release our code at `http://sourceforge.net/projects/zgen/`.

For future work, we will study the incorporation of large-scale language models, and the integration of morphology generation and linearization.

Acknowledgments

We thank the anonymous reviewers for their constructive comments. This work was supported by the National Key Basic Research Program of China via grant 2014CB340503 and the Singapore Ministry of Education (MOE) AcRF Tier 2 grant T2MOE201301 and SRG ISTD 2012 038 from Singapore University of Technology and Design.

References

Anja Belz, Mike White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. 2011. The first surface realisation shared task: Overview and evaluation results. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*, pages 217–226, Nancy, France, September. Association for Computational Linguistics.

Bernd Bohnet, Leo Wanner, Simon Mill, and Alicia Burga. 2010. Broad coverage multilingual deep sentence generation with a stochastic multi-level realizer. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 98–106, Beijing, China, August. Coling 2010 Organizing Committee.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics, July.

Adrià de Gispert, Marcus Tomalin, and Bill Byrne. 2014. Word ordering with phrase-based grammars. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 259–268, Gothenburg, Sweden, April. Association for Computational Linguistics.

Wei He, Haifeng Wang, Yuqing Guo, and Ting Liu. 2009. Dependency based chinese sentence realization. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 809–816, Suntec, Singapore, August. Association for Computational Linguistics.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden, July. Association for Computational Linguistics.

Ryan T McDonald and Fernando CN Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL*.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July. Association for Computational Linguistics.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. *Nat. Lang. Eng.*, 3(1):57–87, March.

Kenji Sagae, Alon Lavie, and Brian MacWhinney. 2005. Automatic measurement of syntactic development in child language. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 197–204, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Linfeng Song, Yue Zhang, Kai Song, and Qun Liu. 2014. Joint morphological generation and syntactic linearization. In *AAAI*, pages 1522–1528.

Stephen Wan, Mark Dras, Robert Dale, and Cécile Paris. 2009. Improving grammaticality in statistical sentence generation: Introducing a dependency spanning tree algorithm with an argument satisfaction model. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 852–860, Athens, Greece, March. Association for Computational Linguistics.

Michael White and Rajakrishnan Rajkumar. 2009. Perceptron reranking for CCG realization. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 410–419, Singapore, August. Association for Computational Linguistics.

Michael White. 2004a. Reining in CCG chart realization. In *In Proc. INLG-04*, pages 182–191.

Michael White. 2004b. Reining in ccg chart realization. In *Natural Language Generation*, pages 182–191. Springer.

Wenduan Xu, Stephen Clark, and Yue Zhang. 2014. Shift-reduce ccg parsing with a dependency model. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 218–227, Baltimore, Maryland, June. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2011a. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

Yue Zhang and Stephen Clark. 2011b. Syntax-based grammaticality improvement using CCG and guided search. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1147–1157, Edinburgh, Scotland, UK., July. Association for Computational Linguistics.

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: Human Language Technologies*, pages 188–193, Portland, Oregon, USA, June. Association for Computational Linguistics.

Yue Zhang, Graeme Blackwood, and Stephen Clark. 2012. Syntax-based word ordering incorporating a large-scale language model. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 736–746, Avignon, France, April. Association for Computational Linguistics.

Yue Zhang. 2013. Partial-tree linearization: Generalized word ordering for text synthesis. In *IJCAI*.