Transition-Based Syntactic Linearization with Lookahead Features

Ratish Puduppully †*, Yue Zhang ‡, Manish Shrivastava †
†Kohli Center on Intelligent Systems (KCIS),
International Institute of Information Technology, Hyderabad (IIIT Hyderabad)
‡Singapore University of Technology and Design
ratish.surendran@research.iiit.ac.in yue.zhang@sutd.edu.sg m.shrivastava@iit.ac.in

Abstract

It has been shown that transition-based methods can be used for syntactic word ordering and tree linearization, achieving significantly faster speed compared with traditional best-first methods. State-of-the-art transition-based models give competitive results on abstract word ordering and unlabeled tree linearization, but significantly worse results on labeled tree linearization. We demonstrate that the main cause for the performance bottleneck is the sparsity of $\text{SHIFT}$ transition actions rather than heavy pruning. To address this issue, we propose a modification to the standard transition-based feature structure, which reduces feature sparsity and allows lookahead features at a small cost to decoding efficiency. Our model gives the best reported accuracies on all benchmarks, yet still being over 30 times faster compared with best-first-search.

1 Introduction

Word ordering is the abstract language modeling task of making a grammatical sentence by ordering a bag of words (White, 2004; Zhang and Clark, 2015; De Gispert et al., 2014; Bohnet et al., 2010; Filipova and Strube, 2007; He et al., 2009), which is practically relevant to text-to-text applications such as summarization (Wann et al., 2009) and machine translation (Blackwood et al., 2010). Zhang (2013) built a discriminative word ordering model, which takes a bag of words, together with optional POS and dependency arcs on a subset of input words, and yields a sentence together with its dependency parse tree that conforms to input syntactic constraints. The system is flexible with respect to input constraints, performing abstract word ordering when no constraints are given, but gives increasingly confined outputs when more POS and dependency relations are specified. It has been applied to syntactic linearization (Song et al., 2014) and machine translation (Zhang et al., 2014).

One limitation of Zhang (2013) is relatively low time efficiency, due to the use of time-constrained best-first-search (White and Rajkumar, 2009) for decoding. In practice, the system can take seconds to order a bag of words in order to obtain reasonable output quality. Recently, Liu et al. (2015) proposed a transition-based model to address this issue, which uses a sequence of state transitions to build the output. The system of Liu et al. (2015) achieves significant speed improvements without sacrificing accuracies when working with unlabeled dependency trees. With labeled dependency trees as input constraints, however, the system of Liu et al. (2015) gives much lower accuracies compared with Zhang (2013).

While the low accuracy can be attributed to heavy pruning, we show that it can be mitigated by modifying the feature structure of the standard transition-based framework, which scores the output transition sequence by summing the scores of each transition action. Transition actions are treated as an atomic output component in each feature instance. This works effectively for most structured prediction tasks, including parsing (Zhang and Clark, 2011a). For word ordering, however, transition actions are significantly more complex and sparse compared...
with parsing, which limits the power of the traditional feature model.

We instead break down complex actions into smaller components, merging some components into configuration features which reduces sparsity in the output action and allows flexible lookahead features to be defined according to the next action to be applied. On the other hand, this change in the feature structure prevents legitimate actions to be scored simultaneously for each configuration state, thereby reducing decoding efficiency. Experiments show that our method is slightly slower compared with Liu et al. (2015), but achieves significantly better accuracies. It gives the best results for all standard benchmarks, being over thirty times faster than Zhang (2013). The new feature structures can be applied to other transition-based systems also.

2 Transition-based linearization

Liu et al. (2015) uses a transition-based model for word ordering, building output sentences using a sequence of state transitions. Instead of scoring output syntax trees directly, it scores the transition action sequence for structural disambiguation. Liu et al.’s transition system extends from transition-based parsers (Nivre and Scholz, 2004; Chen and Manning, 2014), where a state consists of a stack to hold partially built outputs. Transition-based parsers use a queue to maintain input word sequences. However, for word ordering, the input is a set without order. Accordingly, Liu et al. uses a set to maintain the input. The transition actions are:

- **SHIFT-Word-POS**, which removes Word from the set, assigns POS to it and pushes it onto the stack as the top word $S_0$;
- **LEFTARC-LABEL**, which removes the second top of stack $S_1$ and builds a dependency arc $S_1 \xrightarrow{\text{LABEL}} S_0$;
- **RIGHTARC-LABEL**, which removes the top of stack $S_0$ and builds a dependency arc $S_1 \xrightarrow{\text{LABEL}} S_0$.

Using the state transition system, the bag of words \{John, loves, Mary\} can be ordered by (SHIFT-John-NNP, SHIFT-loves-VBZ, LEFTARC-SBJ, SHIFT-Mary-NNP, RIGHTARC-OBJ).

Liu et al. (2015) use a discriminative perceptron model with beam search (Zhang and Clark, 2011a), designing decoding algorithms that accommodate flexible constraints. The features include word($w$), pos($p$) and dependency label($l$) information of words on the stack ($S_0$, $S_1$, ... from the top). For example, the word on top of stack is $S_0 w$ and the POS of the stack top is $S_0 p$. The full set of feature templates can be found in Table 2 of Liu et al. (2015), reproduced here in Table 1. These templates are called configuration features. When instantiated, they are combined with each legal output action to score the action. Therefore, actions are atomic in feature instances.

Formally, given a configuration $C$, the score of a possible action $a$ is calculated as:

$$\text{Score}(a) = \bar{\theta} \cdot \Phi(\tilde{C}, a),$$

where $\bar{\theta}$ is the model parameter vector of the model and $\Phi(\tilde{C}, a)$ denotes a sparse feature vector that consists of features with configuration and action components i.e $\Phi(\tilde{C}, a)$ is sparse. $\bar{\theta}$ has to be loaded for each $a$.

For efficiency considerations and following transition-based models, Liu et al. (2015) scores all possible actions given a configuration simultaneously. This is effectively the same as formulating the

<table>
<thead>
<tr>
<th>Unigrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0 w; S_0 p; S_0, w; S_0, p; S_0, r; w; S_0, r; p;$</td>
</tr>
<tr>
<td>$S_0, t; w; S_0, t; w; S_0, r; w; S_0, r; p;$</td>
</tr>
<tr>
<td>$S_1 w; S_1 p; S_1, w; S_1, p; S_1, r; w; S_1, r; p;$</td>
</tr>
<tr>
<td>$S_1, t; w; S_1, t; w; S_1, r; w; S_1, r; p;$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0 w S_0, w; S_0 w S_0, p; S_0 w S_0, r; p;$</td>
</tr>
<tr>
<td>$S_0 w S_0, t; w; S_0 w S_0, t; p; S_0 w S_0, t; r; p;$</td>
</tr>
<tr>
<td>$S_0 w S_0, r; p; S_0 w S_0, r; w; S_0 w S_0, r; p;$</td>
</tr>
<tr>
<td>$S_1 w S_1, w; S_1 w S_1, p; S_1 w S_1, r; p;$</td>
</tr>
<tr>
<td>$S_1 w S_1, r; w; S_1 w S_1, r; p; S_1 w S_1, r; p;$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0 w S_0, p S_0, w; S_0 w S_0, p S_0, p; S_0 w S_0, p S_0, r; p;$</td>
</tr>
<tr>
<td>$S_0 w S_0, t; w; S_0 w S_0, t; p; S_0 w S_0, t; r; p;$</td>
</tr>
<tr>
<td>$S_0 w S_0, r; p; S_0 w S_0, r; w; S_0 w S_0, r; p;$</td>
</tr>
<tr>
<td>$S_1 w S_1, w S_1, p; S_1 w S_1, w S_1, r; p;$</td>
</tr>
<tr>
<td>$S_1 w S_1, r; w; S_1 w S_1, r; p; S_1 w S_1, r; p;$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linearization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_0; w_{-1}; w_{-2}; w_{-3}; \ldots ; p_{-2}; p_{-1} p_0; }$</td>
</tr>
<tr>
<td>$S_0, t; w; S_0, t; w; S_0, r; w; S_0, r; p; S_0, r; p S_0, r; p;$</td>
</tr>
<tr>
<td>$S_1, t; w S_1, t; w; S_1, p S_1, t; w; S_1, r S_1, r; w; S_1, r S_1, r; p;$</td>
</tr>
</tbody>
</table>
score into
\[ \text{Score}(a) = \bar{\theta}_a \cdot \Phi(C), \quad a \in A. \]
Here \( A \) is the full set of actions and \( \Phi(C) \) is fixed, and \( \bar{\theta}_a \) for all \( a \) can be loaded simultaneously. In a hash-based parameter model, it significantly improves the time efficiency.

3 Feature structure modification

3.1 Two limitations of the baseline model

There are two major limitations in the feature structure of Liu et al. (2015). First, the \textsc{Shift} actions, which consist of the word to shift and its POS, are highly sparse. Since the action is combined with all configuration features, there will be no active feature for disambiguating the shift actions for OOV words. This issue does not exist in transition-based parsers because words are not a part of their transition actions. Second, input constraints are not leveraged by the feature model. Although the dependency relations of the word to shift can be given as inputs, they are used only as constraints to the decoder, but not as features to guide the shift action. Such lookahead information on the to-be-shifted word can be highly useful for disambiguation.

For example, consider the bag of words \{John, loves, Mary\}. Without constraints, both ‘John loves Mary’ and ‘Mary loves John’ are valid word ordering results. However, given the constraint (John, SBJ, loves), the correct answer is reduced to the former. The first action to build the two examples are (\textsc{Shift-}John-NNP) and (\textsc{Shift-}Mary-NNP), respectively. According to Liu et al.’s feature model, there is no feature to disambiguate the first \textsc{Shift} action if both ‘John’ and ‘Mary’ are OOV words. The system has to maintain both hypotheses and rely on the search algorithm to disambiguate them after the dependency arcs (John, SBJ, loves) and (Mary, OBI, loves) are built. However, given the syntactic constraint that ‘John’ is the subject, the disambiguation can be done right when performing the first \textsc{Shift} action. This requires the dependency arc label to be extracted for the word to shift e.g.‘John, Mary), which is a lookahead feature. In addition, the OOV word ‘John’ must be excluded from the feature instance, which implies that the \textsc{Shift-}John-NNP action must be simplified.

| set of label and POS of child nodes of L |
| \( L_{cls}; L_{clns}; L_{cps}; L_{cpns}; \) |
| \( S_0 w_{L_{cls}}; S_0 p_{L_{cls}}; S_1 w_{L_{cls}}; S_1 p_{L_{cls}}; \) |
| \( S_0 w_{L_{clns}}; S_0 p_{L_{clns}}; S_1 w_{L_{clns}}; S_1 p_{L_{clns}}; \) |
| \( S_0 w_{L_{cps}}; S_0 p_{L_{cps}}; S_1 w_{L_{cps}}; S_1 p_{L_{cps}}; \) |

| set of label and POS of siblings of L |
| \( L_{sls}; L_{slns}; L_{spgs}; \) |
| \( S_0 w_{L_{sls}}; S_0 p_{L_{sls}}; S_1 w_{L_{sls}}; S_1 p_{L_{sls}}; \) |
| \( S_0 w_{L_{slns}}; S_0 p_{L_{slns}}; S_1 w_{L_{slns}}; S_1 p_{L_{slns}}; \) |
| \( S_0 w_{L_{spgs}}; S_0 p_{L_{spgs}}; S_1 w_{L_{spgs}}; S_1 p_{L_{spgs}}; \) |

| parent label, POS and word of L |
| \( L_{psL_{lp}}; L_{psL_{lp}}; L_{psL_{lp}}; \) |
| \( S_0 w_{L_{psL_{lp}}}; S_0 p_{L_{psL_{lp}}}; S_1 w_{L_{psL_{lp}}}; S_1 p_{L_{psL_{lp}}}; \) |
| \( S_0 w_{L_{psL_{lp}}}; S_0 p_{L_{psL_{lp}}}; S_1 w_{L_{psL_{lp}}}; S_1 p_{L_{psL_{lp}}}; \) |
| \( S_0 w_{L_{psL_{lp}}}; S_0 p_{L_{psL_{lp}}}; S_1 w_{L_{psL_{lp}}}; S_1 p_{L_{psL_{lp}}}; \) |

| set of label and POS of child nodes of S0 |
| \( S_{cls}; S_{clns}; S_{cps}; S_{cpns}; \) |
| \( S_0 w_{S_{cls}}; S_0 p_{S_{cls}}; S_1 w_{S_{cls}}; S_1 p_{S_{cls}}; \) |
| \( S_0 w_{S_{clns}}; S_0 p_{S_{clns}}; S_1 w_{S_{clns}}; S_1 p_{S_{clns}}; \) |
| \( S_0 w_{S_{cps}}; S_0 p_{S_{cps}}; S_1 w_{S_{cps}}; S_1 p_{S_{cps}}; \) |
| \( S_0 w_{S_{cpns}}; S_0 p_{S_{cpns}}; S_1 w_{S_{cpns}}; S_1 p_{S_{cpns}}; \) |

| set of label and POS of siblings of S0 |
| \( S_{sls}; S_{slns}; S_{spgs}; \) |
| \( S_0 w_{S_{sls}}; S_0 p_{S_{sls}}; S_1 w_{S_{sls}}; S_1 p_{S_{sls}}; \) |
| \( S_0 w_{S_{slns}}; S_0 p_{S_{slns}}; S_1 w_{S_{slns}}; S_1 p_{S_{slns}}; \) |
| \( S_0 w_{S_{spgs}}; S_0 p_{S_{spgs}}; S_1 w_{S_{spgs}}; S_1 p_{S_{spgs}}; \) |
| \( S_0 w_{S_{spgs}}; S_0 p_{S_{spgs}}; S_1 w_{S_{spgs}}; S_1 p_{S_{spgs}}; \) |

| parent label and POS of S0 |
| \( S_{psL_{lp}}; S_{psL_{lp}}; \) |
| \( S_0 w_{S_{psL_{lp}}}; S_0 p_{S_{psL_{lp}}}; S_1 w_{S_{psL_{lp}}}; S_1 p_{S_{psL_{lp}}}; \) |
| \( S_0 w_{S_{psL_{lp}}}; S_0 p_{S_{psL_{lp}}}; S_1 w_{S_{psL_{lp}}}; S_1 p_{S_{psL_{lp}}}; \) |
| \( S_0 w_{S_{psL_{lp}}}; S_0 p_{S_{psL_{lp}}}; S_1 w_{S_{psL_{lp}}}; S_1 p_{S_{psL_{lp}}}; \) |

Table 2: Lookahead feature templates

As a second example, information about dependents can also be useful for disambiguating \textsc{Shift} actions. In the above case, the fact that the subject has not been shifted onto the stack can be a useful indicator for not shifting the verb ‘loves’ onto the stack in the beginning. Inspired by the above, we exploit a range of lookahead features from syntactic constraints.

3.2 New feature structure for \textsc{Shift} actions

We modify the feature structure of Liu et al. (2015) by breaking down the \textsc{Shift-Word-POS} action into three components, namely \textsc{Shift}, Word and POS, using only the action type \textsc{Shift} as the output action component in feature instances, while combin-
ing Word and POS with other configuration features to form a set of lookahead features.

For example, consider the configuration feature $S_0w$, which captures the word on the top of the stack. Under the feature structure of Liu et al., it is combined with each possible action to form features for scoring the action. As a result, for scoring the action $\text{SHIFT-Lw-Lp}$, $S_0w$ is instantiated into $S_0w$-$\text{SHIFT-Lw-Lp}$, where $Lw$ is the word to shift and $Lp$ is its POS. Under our new feature structure, the action component is reduced to $\text{SHIFT}$ only, while $Lw$ and $Lp$ should be used in lookahead features. Now a effectively equivalent configuration feature to Liu et al.’s $S_0w$ is $S_0w$-$Lw-Lp$, with the lookahead $Lw$ and $Lp$. It gives $S_0w$-$Lw-Lp$-$\text{SHIFT}$ when combined with the action $\text{SHIFT}$.

This new feature structure reformulates the $\text{SHIFT}$ action features only. The $\text{LEFTARC}$/ $\text{RIGHTARC}$ actions remain $\text{LEFTARC}$/ $\text{RIGHTARC}$-$\text{LABEL}$ since they are not sparse. Note that the change is in the $\text{action features}$ rather than the $\text{actions}$ themselves. Given the bag of words $\{\text{John, loves, Mary}\}$, the action $\text{SHIFT-John}$-$\text{NNP}$ is still different from the action $\text{SHIFT-Mary}$-$\text{NNP}$. However, the $\text{action component}$ of the features becomes $\text{SHIFT}$ only, and the words $\text{John}$/ $\text{Mary}$ must be used as lookahead $\text{configuration features}$ for their disambiguation.

The new feature structure can reduce feature sparsity by allowing lookahead features without word information. For example, a configuration feature $S_0w$-$Lp$, which contains only the stack top word and the POS of the lookahead word, can still fire even if the word to shift is OOV, thereby disambiguating OOV words of different POS. In addition, the lookahead $Lw$ and $Lp$ do not have to be combined with every other configuration feature, as with Liu et al. (2015), thereby allowing more flexible feature combination and a leaner model.

## 3.3 The new features

The new feature structure includes two types of features. The first is the same feature set as Liu et al. (2015), but with the $\text{SHIFT}$ action component not having Word and POS information. We call this type of features as $\text{base features}$. The second is a set of $\text{lookahead features}$, which are shown in Table 2. Here $L_{\text{cls}}$ represents set of arc labels on child nodes (of the word $L$ to shift) that have been shifted on to the stack, $L_{\text{ctns}}$ represents set of labels on child nodes that have not been shifted, $L_{\text{spns}}$ the set of shifted sibling nodes, $L_{\text{lspa}}$ the label set of unshifted sibling nodes, $L_{\text{cpps}}$ the POS set of shifted child nodes, $L_{\text{lcpms}}$ the POS set of unshifted child nodes, $L_{\text{spms}}$ the POS set of shifted sibling nodes and $L_{\text{spms}}$ the POS set of unshifted sibling nodes. $L_{\text{ps}}$ is a binary feature indicating if the parent has been shifted. $L_{\text{lp}}$ represents label on the parent, $L_{\text{pp}}$ POS of parent and $L_{\text{wp}}$ the parent word form. We define similar lookahead features for $S_0$. These features are instantiated only for $\text{SHIFT}$ actions.

The new feature structure prevents all possible actions from being scored simultaneously, because the lookahead Word and POS are now in configuration features, rather than output actions, making it necessary to score the shifting of different words or POS separately. This leads to reduced search speed. Nevertheless, our experiments show that they give a desirable tradeoff between efficiency and accuracy.

Note that the new features are much less than a full Cartesian product of lookahead features and the original features. This is a result of manual feature engineering, which allows similar accuracies to be achieved using a much smaller model, thereby increasing the time efficiency.
4 Experiments

Following previous work we conduct experiments on the Penn TreeBank (PTB), using Wall Street
Journal sections 2-21 for training, 22 for development and 23 for testing. Gold-standard dependency
trees are derived from bracketed sentences using
Penn2Malt, and base noun phrases are treated as a
single word. The BLEU score is used to evaluate
the performance of linearization.

Table 4 shows a difference in scores between
transition-based linearization system of Liu et al.
(2015) (L15) and best-first system of Zhang (2013)
(Z13). L15 performs better for word ordering with
unlabeled dependency arcs, but poorly for the task
of labeled syntactic linearization.

Table 3 shows a series of development experi-
ments comparing our system with Z13 and L15.
We vary the amount of input syntactic constraints
by randomly sampling from POS and dependency
labels of the development set. Our system gives
consistently higher accuracies when compared with
both Z13 and L15. Compared to L15, the increase
in scores for unconstrained word ordering is due to
the introduction of reduced feature sparsity. The
improvements on tree linearization tasks involving
partial to full dependency constraints are also due to
lookahead features that leverage tree information to
reduce ambiguity early. Though slower than L15,
our system is over 30 times faster compared to Z13.

We compare final test scores with previous meth-
ods in the literature in Table 4. Our system im-
proves upon the previous best scores by 8.7 BLEU
points for the task of unlabeled syntactic lineariza-
tion. For the task of labeled syntactic linearization,
we achieve the score of 91.8 BLEU points, the high-
est results reported so far.

Table 5 contains examples of fully constrained
output. In the first example ‘will’ is the ROOT node
with two child nodes ‘also’ and ‘compete’. Looka-
head feature for child dependency labels $L_{cls}, L_{clns}$
on the node ‘will’ can help order the segment ‘also
will compete’ correctly in our system. Without such
features, the system of L15 yields an output that
starts with ‘The spinoff with Fujitsu’ which is locally
fluent, but leaving the words ‘also’ and ‘will’ diffi-
cult to handle. In the second example, ‘Dr. Talcott’
is OOV. Hence system of L15 is not able to score it
and thus order it correctly. Our system makes use of
both POS and dependency label of ‘Dr. Talcott’ to
order it correctly.

5 Conclusion

We identified a feature sparsity issue in state-of-the-
art transition-based word ordering, proposing a so-
lution by redefining the feature structure and intro-
ducing lookahead features. The new method gives
the best accuracies on a set of benchmarks, which
show that transition-based methods are a fast and
accurate choice for syntactic linearization. Future
work include the testing of this model in a lineariza-
tion shared task (Belz et al., 2011) and investigating
the integration of large scale training data (Zhang et
al., 2012; Liu and Zhang, 2015).

We release our source code under GPL at
https://github.com/SUTDNLP/ZGen/
releases/tag/v0.2.

Acknowledgments

We thank Yijia Liu for helpful discussions and for
sharing the Latex templates and the anonymous re-
viewers for their constructive comments. This work
was supported by the Singapore Ministry of Educa-
tion (MOE) AcRF Tier 2 grant T2MOE201301.
References


