A Reranking Approach for Dependency Parsing with Variable-sized Subtree Features

Mo Shen, Daisuke Kawahara, and Sadao Kurohashi
Graduate School of Informatics
Kyoto University
Yoshida-honmachi, Sakyo-ku,
Kyoto, 606-8501, Japan
shen@nlp.ist.i.kyoto-u.ac.jp {dk,kuro}@i.kyoto-u.ac.jp

Abstract

Employing higher-order subtree structures in graph-based dependency parsing has shown substantial improvement over the accuracy, however suffers from the inefficiency increasing with the order of subtrees. We present a new reranking approach for dependency parsing that can utilize complex subtree representation by applying efficient subtree selection heuristics. We demonstrate the effectiveness of the approach in experiments conducted on the Penn Treebank and the Chinese Treebank. Our system improves the baseline accuracy from 91.88% to 93.37% for English, and in the case of Chinese from 87.39% to 89.16%.

1. Introduction

In dependency parsing, graph-based models are prevalent for their state-of-the-art accuracy and efficiency, which are gained from their ability to combine exact inference and discriminative learning methods. The ability to perform efficient exact inference lies on the so-called factorization technique which breaks down a parse tree into smaller substructures to perform an efficient dynamic programming search. This treatment however restricts the representation of features to in a local context which can be, for example, single edges or adjacent edges. Such restriction prohibits the model from exploring large or complex structures for linguistic evidence, which can be considered as the major drawback of the graph-based approach.

Attempts have been made in developing more complex factorization techniques and corresponding decoding methods. Higher-order models that use grand-child, grand-sibling or tri-sibling factorization were proposed in (Koo and Collins, 2010) to explore more expressive features and have proven significant improvement on parsing accuracy. However, the power of higher-order models comes with the cost of expensive computation and sometimes it requires aggressive pruning in the pre-processing.

Another line of research that explores complex feature representations is parse reranking. In its general framework, a K-best list of parse tree candidates is first produced from the base parser; a reranker is then applied to pick up the best parse among these candidates. For constituent parsing, successful results has been reported in (Collins, 2000; Charniak and Johnson, 2005; Huang, 2008). For dependency parsing, the efficient algorithms for produce K-best list for graph-based parsers have been proposed in (Huang and Chiang, 2005) for projective parsing and in (Hall, 2007) for non-projective parsing; Improvements on dependency accuracy has been achieved in (Hall, 2007; Hayashi et al., 2011). However, the feature sets in these studies explored a relatively small context, either by emulating the feature set in the constituent parse reranking, or by factorizing the search space. A desirable approach for the K-best list reranking is to encode features on subtrees extracted from the candidate parse with arbitrary

26 th Pacific Asia Conference on Language, Information and Computation pages 308-317

orders and structures, as long as the extraction process is tractable. It is an open question how to design this subtree extraction process that is able to selects a set of subtrees which provides reliable and concrete linguistic evidence. Another related challenge is to design a proper back-off strategy for any structures extracted, since large subtree instances are always sparse in the training data.

In this paper, we explore a feature set that makes fully use of dependency grammar, can capture global information with less restriction in the structure and the size of the subtrees, and can be encoded efficiently. It exhaustively explores a candidate parse tree for features from the most simple to the most expressive while maintaining the efficiency in the sense that it does not add additional complexities over the K-best parsing.

We choose the K-best list reranking framework rather than the forest reranking in (Huang, 2008) because an explicit representation of parse trees is needed in order to compute the features for reranking. We implemented an edge-factored parser and a second-order sibling-factored parser which emulate models in the MSTParser described in (McDonald et al., 2005; McDonald and Pereira, 2006) as our base parsers.

In the rest part of this paper, we first give a brief description of the dependency parsing, then we describe the feature set for reranking, which is the major contribution of this paper. Finally, we present a set of experiment for the evaluation of our method.

2. Dependency Parsing

The task of dependency parsing is to find a tree structure for a sentence in which edges represent the head-modifier relationship between words: each word is linked to a unique "head" such that the link forms a semantic dependency while the main predicate of the sentence is linked to a dummy "root". An example of dependency parsing is illustrated in Figure 1. A dependency tree is called projective if the links can be drawn on the linearly ordered words without any crossover. We will focus on projective trees throughout this paper.

We formally define the dependency parsing task. Give a sentence x, the best parse tree is obtained by searching for the tree with highest score:

$$\tilde{y} = \text{argmax}_{y \in \mathcal{Y}(x)} Score(y, x) , \qquad (1)$$

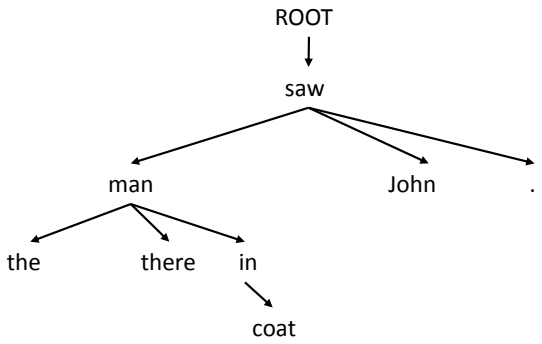

Figure 1. A dependency parse tree of the sentence "the man there in coat saw John."

where $\mathcal{Y}(x)$ is the search space of possible parse trees for x, and y is a parse tree in $\mathcal{Y}(x)$. A problem in solving equation (1) is that the number of candidates in the search space grows exponentially with the length of the sentence which makes the searching infeasible. A common remedy for this problem is to factorize a parse tree into small subtrees, called factors, which are scored independently. The score of parse tree under a factorization is the summation of scores of factors:

$$Score(y, x) = \sum_{t \in y} Score(t, x) , \qquad (2)$$

where t is a factor of y. The search space can be therefore encoded in a compact form which allows dynamic programming algorithms to perform efficient exact inference. The score function for each factor is assigned as an inner product of a feature vector and a weight vector w:

$$Score(t, x) = w \cdot f(t, x) . \qquad (3)$$

The feature vector is defined on the factor t which means it is only able to capture tree-structure information from a small context. This can be seen as the off-set for performing exact inference. The goal of training a parser is to learn a weight vector that assigns scores to effectively discriminate good parses from bad parses.

We use the edge factorization and the sibling factorization models described in (McDonald et al., 2005; McDonald and Pereira, 2006) to construct our base parsers. We learn the weight vector by

applying the averaged perceptron algorithm (Collins, 2002) for its efficiency and stable performance. An illustration for generic perceptron algorithm is shown in Pseudocode 1.

Pseudocode 1: Generic perceptron learning

```
1  for training data (x_i, y_i), i = 1..N
2    for iteration t = 1..T
3      ỹ = argmax_{y∈Y(x)} w · f(y, x_i)
4      if ỹ ≠ y_i
5        w ← w + f(y_i, x_i) − f(ỹ, x_i)
6    end
7  End
```

3. Parse Reranking

In this section, we describe our reranking approach and introduce the feature set consists of three different types.

3.1 Overview of Parse Reranking

The task of reranking is similar with that of parsing instead of that the searching of parse tree is performed on a K-best list with selected parse candidates rather than the entire search space:

$$\tilde{y} = \text{argmax}_{y \in Kbest(x)} Score'(y, x) \quad (4)$$

The scoring function is defined as:

$$Score'(y, x) = L(y, x) + w \cdot f(y, x) \quad (5)$$

Where $L(y, x)$ is the score of y output by the base parser. We define the oracle parse y^+ to be the parse in the K-best list with highest accuracy compared with the gold-standard parse. The goal of reranking is to learn the weight vector so that the reranker can pick up the oracle parse as many times as possible. Note that in the reranking framework, the feature is defined on the entire parse tree which enables the encoding of global information. We learn the weight vector of the reranker also by the averaged perceptron algorithm shown in Pseudocode 1 with slight modification that only substitute the search space $\mathcal{Y}(x)$ with the K-best output Kbest(x), and gold parse y_i with oracle parse y_i^+.

3.2 Feature Sets for Reranking

Benefit from the K-best list obtained in the parsing stage, we are able to perform discriminative learning in order to select a good parse among candidates in a shrunk search space, which allows utilization of global features. We define three types of features below.

Trimmed subtree: For each node in a given parse tree, we check its dominated subtrees to see whether they are likely to appear in a good parse tree or not. To efficiently obtain these subtrees, we set a local window that bound a node from its left side, right side and bottom. We then extract the maximum subtree inside this window, means that we cut off those nodes that are too distant in sequential order or too deep in a tree.

The above subtree extraction often results in very large instances which are extremely sparse in the training data, therefore it is necessary to keep smaller subtrees as back-offs. In most cases, however, it is prohibitively expensive to enumerate all the smaller subtrees. Instead of enumeration, we design a back-off strategy that select subtrees by attempting to leave out nodes that are far away from the subtree's root and keeps those that are nearby. Precisely, after extracted the first subtree of a node, we vary the three boundaries (the left, the right and the bottom boundary respectively) from their original positions to positions that are closer to the root of the subtree, such that it tightens up the local window. For each possible combination of the variable boundaries, we extract the largest subtree from the new local window and add it to the set of the so called "trimmed subtrees" set of the node. This back-off strategy comes from our observation that nodes that are close to the root may provide more reliable information than those that are distant. As it is infeasible to enumerate all small subtrees as back-offs, throwing away the redundant nodes from the outer part of a large subtree is a reasonable choice.

Figure 2 illustrates the construction of the "trimmed subtrees" set of the node "saw", for the sentence in Figure 1. The initial boundary parameters are set large enough so the local window contains the entire parse tree[1]. #LEFT, #RIGHT and #BOTTOM represents the three boundary variables, which range from -6 to -1, from 3 to 1 and from 3 to 0 respectively. Context

[1] In practice we use smaller local window with fixed size.

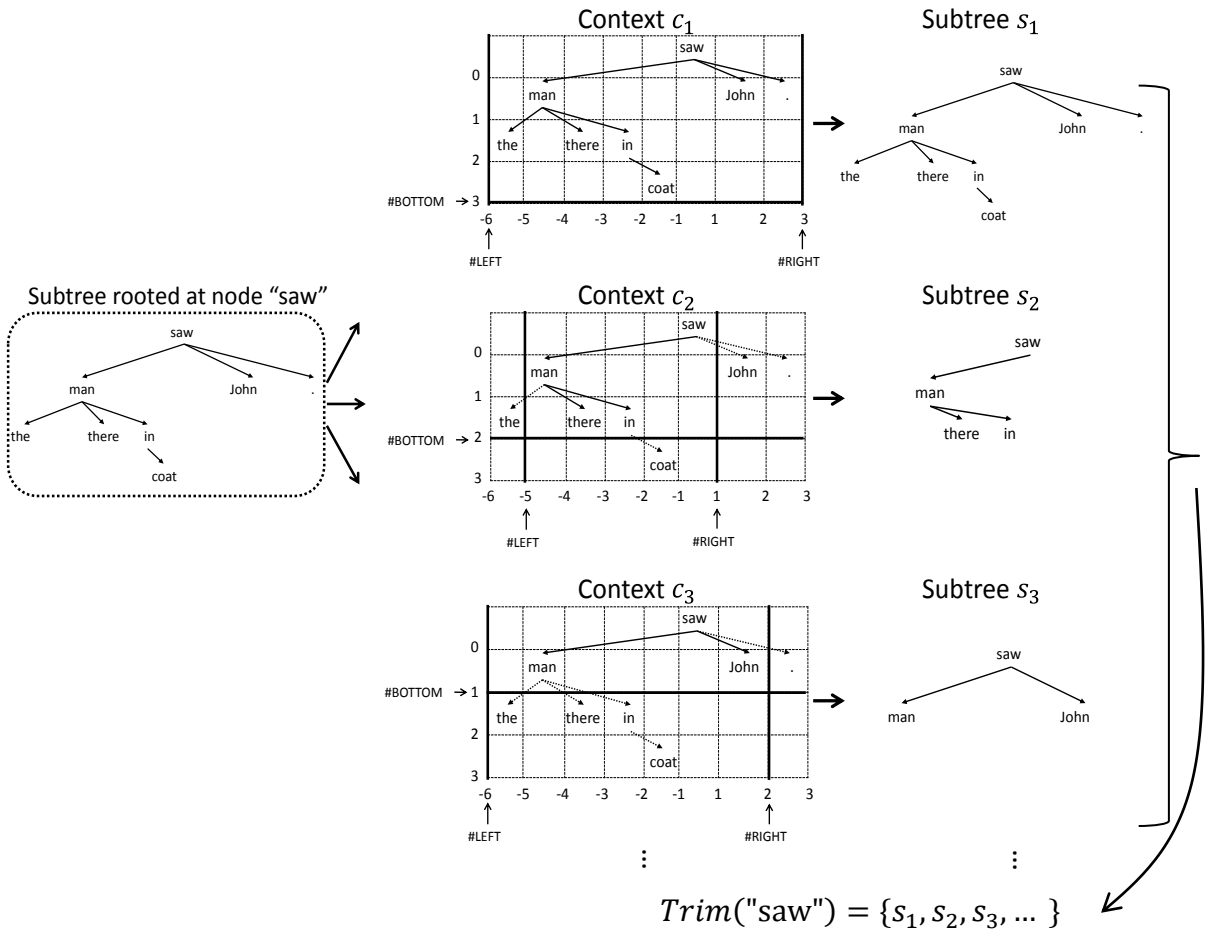

Figure 2. Extraction of trimmed subtrees from the node "saw". "#LEFT", "#RIGHT" and "#BOTTOM" represents the three boundaries that can vary along possible positions on the corresponding axis. Contexts c_1, c_2 and c_3 represnt three instances of possible combinations of boundary positions. s_1, s_2 and s_3 are resulted subtrees that are elements in the trimmed subtrees set of the node "saw".

c_1, c_2 and c_3 represent three different combinations of boundary positions. Subtree s_1, s_2 and s_3 are the extracted subtrees in the correspond context. They and other similarly extracted subtrees together consist in the set $Trim$("saw"), the trimmed subtrees set of the node "saw". We use this set in two ways. First, for each element in this set, we encode a series of features. Second, this set is kept for reuse in another type of feature, which we describe latter. We repeat this extraction process for all nodes in a parse tree and keep their trimmed subtrees set.

In Figure 3 we show some of the extracted subtrees in the set $Trim$("saw"), among which the subtree (c) can be regard as a grand sibling factor and the subtree (d) is similar with a tri-sibling factor in (Koo and Collins, 2010), but the siblings are located in both sides of the head node. The subtree (a) and subtree (b) are subtrees we extracted that cannot be represented in common factorization methods, which confirmed the ability of this feature set to capture a large variety of structures.

It should be noted that, while in a direct calculation there are 72 (6-by-3-by-4) possible combinations for boundary positions in the example in Figure 2, this number can almost always be reduced in practice. In this example, when #LEFT reached the position at index -4, the entire left branch of the root node is in fact cut so no further movement for #LEFT is allowed. Moreover, after #BOTTOM moved to the position

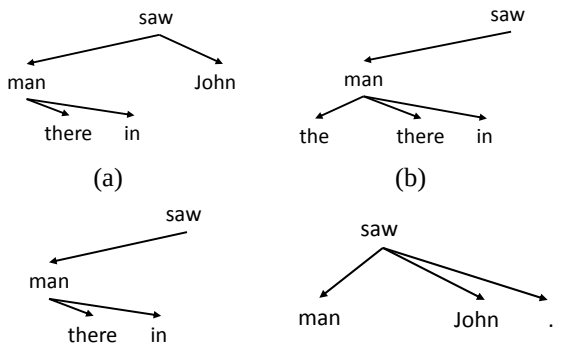

Figure 3. Some of the extracted trimmed subtrees by the process described in Figure 2. (c) is identical with a grand-sibling factor in a third-order parsing model and (d) is similar to a tri-sibling factor but siblings are on both sides of the head.

at index 1, the sequential order distance between "man" and "saw" is updated and reduced to 1, which restricts #LEFT to only two possible positions, either to the left or to the right of the word "man". Therefore one can verify that the true number of combinations of boundary positions is actually 25. Briefly, for a node we are focusing on, we decompose the extracted subtree from the initial local window into three parts: the node itself, the sequence of its left descendants and the sequence of its right descendants. The two sequences of descendants are in a preordering of depth-first search, during which we mark "anchor" nodes as the next-possible cut-in positions for the left/right boundary variables. Furthermore, the list of anchor nodes will keep updating whenever the bottom boundary variable moved to a new position. As a result, we are able to minimize the number of boundary combinations to speed up the subtrees extraction.

For each extracted subtree, we encode features as follow. A trimmed subtree feature is represented as an n-tuple: $\langle \mathrm{a}_1, \ldots, \mathrm{a}_n \rangle$ where a_1 is the root of the subtree, and $\mathrm{a}_i, \mathrm{i} > 1$ are nodes in the subtree in preordering through a depth-first search from a_1. For a_1 we encode its word form, Part-of-Speech tag, and the combination of them. For any non-root node, we encode its Part-of-Speech tag, a binary value indicating the branch direction from its head, and its depth from a_1. We also encode features that omit the Part-of-Speech tags of the sequence $\mathrm{a}_2, \ldots, \mathrm{a}_n$, so that only the structural preference of the subtree's root is retained. An example is shown below which illustrates a feature for the subtree in Figure 3(a):

$$\langle (\mathrm{saw}, \mathrm{V}), (\mathrm{N}, \mathrm{LEFT}, \mathrm{depth} = 1), (\mathrm{N}, \mathrm{RIGHT}, \mathrm{depth} = 2), (\mathrm{P}, \mathrm{RIGHT}, \mathrm{depth} = 2), (\mathrm{N}, \mathrm{RIGHT}, \mathrm{depth} = 1) \rangle ,$$

where V, N and P are Part-of-Speech tags of corresponding nodes; we use simplified tags for illustration purpose. The preordering of nodes together with their branch direction and depth information guarantees that the mapping from a given subtree structure to its corresponding feature string is injective. Another example below shows a feature that omits all the Part-of-Speech tags except on the root of the subtree:

$$\langle (\mathrm{saw}, \mathrm{V}), (-, \mathrm{LEFT}, \mathrm{depth} = 1), (-, \mathrm{RIGHT}, \mathrm{depth} = 2), (-, \mathrm{RIGHT}, \mathrm{depth} = 2), (-, \mathrm{RIGHT}, \mathrm{depth} = 1) \rangle$$

Finally, we associate the list of features encoded for a subtree rooted on a node a with the corresponding element in the set $Trim(a)$. We make use of this set in the next type of features to avoid repeated computation.

Sibling subtree: The trimmed subtree features consider the preference of a node toward its dominated subtree—whether the subtree is likely to appear in a good parse. In the reranking framework, however, as we do not factorize a parse tree, we may suffer from a problem that the information we got among candidates are unbalanced. Typically, when computing the trimmed subtree features, a candidate parse with most nodes being leaves will provide little information except on the root node, while on another parse that has fewer leaves and more depth we can have a bunch of features that give more information. This defect makes the comparison between candidates be "unfair" and thus less reliable. Therefore, it is natural to raise the question the other way round—whether a node is a good head for a subtree. To answer this question, we consider a dynamic programming structure called *complete span* introduced in (Eisner, 1996).

A complete span consists of a head node and all its descendants on one side, which can also be

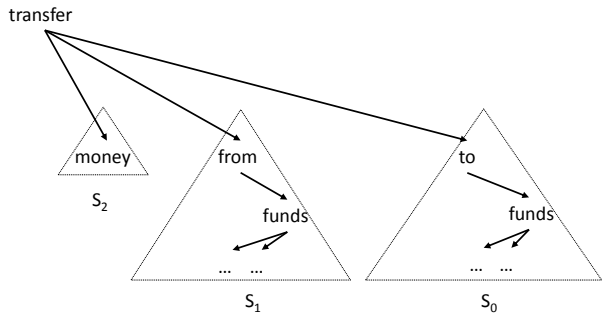

Figure 4. A complete span for the clause "transfer money from the new funds to other investment funds" where we omitted some of the details. This structure functions as a relatively independent and complete component in the entire parse tree. Features are encoded over the tuples: <transfer, -,s_2>, <transfer, s_2,s_1>, <transfer, s_1,s_0>, <transfer, s_0,->.

considered as a head node and sibling subtrees shown in Figure 4. In our observation, a complete span functions as a relatively independent and complete semantic structure in the parse tree, we thus believe that it can provide sufficient information to decide the head of a subtree without looking at any larger context.

Specifically, for each node m in a candidate parse, its sibling subtree features is the collection of all 3-tuples:

$$\langle h, \mathrm{f}(s, p_1, i_1), \mathrm{f}(m, p_2, i_2)\rangle$$

where h represents the word form, the Part-of-Speech tag, or the combination of the word form and the Part-of-Speech tag of the head node of *m*; s is the nearest sibling node of *m* in-between *h* and *m*; and the expression $\mathrm{f}(a, p, i)$ represents the i_{th} feature encoded on a trimmed subtree in the set $Trim(a)$, such that the trimmed subtree is the one extracted within the local window p. Here an important point is that we make use of trimmed subtrees extracted in the previous phase. As mentioned before, since we keep the history of trimmed subtree extraction, it eliminates the need to re-compute any subtree structures on the sibling nodes and hence is efficient to encode.

The way we define our sibling subtree features for reranking can also be seen as the natural extension of the sibling factorization in (McDonald and Pereira, 2006) from the word-based case to the subtree-based case, while the original sibling factor can be represented as a 3-tuple $\langle h, s, m\rangle$ using the same notation.

Chain: A chain type feature encodes information for a subtree that each node has exactly one incoming edge and one outgoing edge, except on the two ends (hence a "chain"). We extract all these kind of subtrees from a parse tree in the candidates list with a parameter set to limit the number of edges in the subtree. This type of features emulates the common grandparent-grandchildren structure in dependency parsing, while we loosen the restriction on the order of the subtree. It functions as a complementary for other types of features.

From the parse tree of the sentence in Figure 1, we extract all chains whose order is larger than 2, since otherwise features defined on edges have already been utilized in our base parsers which are edge-factored and sibling factored. We show these chain type subtrees in Figure 5. For a consideration of efficiency, a proper value of the order limit should be set no larger than 5 according to our experience.

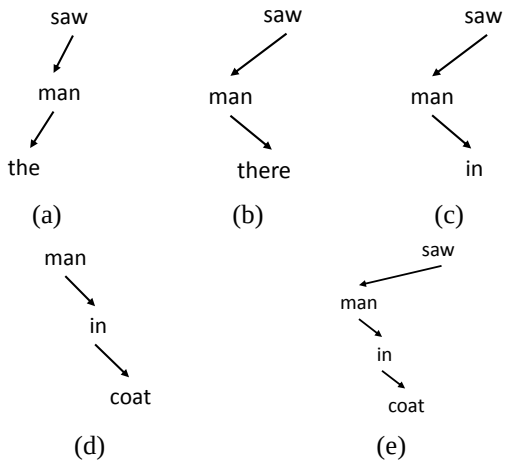

Figure 5. All chain type subtrees extracted from the gold-standard parse tree of the sentence "the man there in coat saw John."

The information encoded from extracted subtrees includes word form, Part-of-Speech tag and relative position in the subtree for each node. When dealing with long subtrees, however, encoding lexical information suffers from data sparsity. We therefore encode lexical information only on one of the two ends of the subtree in each time, while for all nodes we encode their

grammatical and positional information. Thus for the subtree (e) in Figure 5, a feature can appear as:

$$\langle (\mathrm{V}, \mathrm{saw}, -), (\mathrm{N}, -, \mathrm{left}), (\mathrm{P}, -, \mathrm{right}), (\mathrm{N}, -, \mathrm{right}) \rangle$$

A binary value, here we denote as "left" and "right", is used to indicate the direction of branch of a node from its head.

4. Evaluation

We present our experimental results on two languages, English and Chinese. For English experiment, we use the Penn Treebank WSJ part. We convert the constituent structure in the Treebank into dependency structure with the tool Penn2Malt and the head-extraction rule identical with that in (Yamada and Matsumoto, 2003). To align with previous work, we use the standard data division: section 02-21 for training, section 24 for development, and section 23 for testing. As our system assumes Part-of-Speech tags as input, we use MXPOST, a MaxEnt tagger (Ratnaparkhi, 1996) to automatically tag the test data. The tagger is trained on the same training data.

For Chinese, we use the Chinese Treebank 5.0 with the following data division: files 1-270 and files 400-931 for training, files 271-300 for testing, and files 301-325 for development. We use Penn2Malt to convert the Treebank into dependency structure and the set of head-extraction rules for Chinese is identical with the one in (Zhang and Clark, 2008). Moreover, for Chinese we use the gold standard Part-of-Speech tags in evaluation.

We apply unlabeled attachment score (UAS) to measure the effectiveness of our method, which is the percentage of words that correctly identified their heads. For all experiments conducted, we use the parameters tuned in the development set.

We train two base parsers which are the re-implementation of the first-order and second-order parsers in the MSTParser (McDonald et al., 2005; McDonald and Pereira, 2006) with 10 iterations on English and Chinese training dataset. We use 30-way cross-validation on the identical training dataset to provide training data for the rerankers. We use the following parameter setting for the feature sets throughout the experiments: for chain-type features, the maximum order of chains is set to 5; the left, right and bottom boundary for the trimmed subtree features are 10, 10 and 5 respectively. For the main experiments we use K=50, the capacity of the list of parse tree candidates, in the training of the rerankers. Moreover, as it is not necessary to use identical value of K in the training and the test, we also conduct an experiment using miss-matching K values on Chinese dataset.

System	English UAS
McDonald05	90.9
McDonald06	91.5
Zhang11	92.9
Koo10	93.04
Martins10	93.26
Order 1	90.91
Order 2	91.88
Order 1 reranked	92.50
Order 2 reranked	**93.37**
Koo08^{+}	93.16
Chen09^{+}	93.16
Suzuki09^{+}	93.79

Table 1. English UAS of previous work, our base parsers, and reranked results. "$^{+}$": semi-supervised parsers.

4.1 Experimental Results

We show the experimental results for English in Table 1. Each row in this table shows the UAS of the corresponding system. "McDonald05" and "McDonald06" stand for the first-order and second-order models in the MSTParser (McDonald et al., 2005; McDonald and Pereira, 2006). "Zhang11" stands for the transition-based parser proposed in (Zhang and Nivre, 2011). "Koo10" stands for the Model 1 in (Koo and Collins, 2010) which is a third-order model. "Martins10" stands for the turbo parser proposed in (Martins et al., 2010). "Order 1" and "Order 2" are our re-implementation of MSTParser and are used as the base parsers for our reranking experiments. "Order 1 reranked" and "Order 2 reranked" are rerankers pipelined on the two base parsers. "Koo08", "Chen09" and "Suzuki09" are parsers using semi-supervised methods (Koo et al., 2008; Chen et al., 2009; Suzuki et al., 2009). In Table 2 we show the results for Chinese. "Duan07" and "Yu08" stands for the two probabilistic parsers in (Duan et al., 2007; Yu et al., 2008). "Chen09" stands for the same system in Table 1.

System	Chinese UAS
Duan07	84.36
Yu08	87.26
Order 1	85.44
Order 2	87.39
Order 1 reranked	87.63
Order 2 reranked	**89.16**
Chen09^{+}	89.91

Table 2. Chinese UAS of previous work, our baseline parsers, and reranked results. "+": semi-supervised parsers.

As we can see from the results, for English, the accuracy increased from 90.91% ("Order 1") to 92.50% ("Order 1 reranked") for the first-order parse reranker and from 91.88%("Order 2") to 93.37%("Order 2 reranked") for the second-order parse reranker. For Chinese, the accuracy increased from 85.44% to 87.63% for the first-order parse reranker, and for the second order case it increased from 87.39% to 89.16%. It shows that our reranking systems obtain the highest accuracy among supervised systems. For English, the reranker "Order 2 reranked" even slightly outperforms "Martins10", the turbo parser which to the best of our knowledge achieved the highest accuracy in Penn Treebank. Although our rerankers are beaten by the semi-supervised systems "Suzuki09" and "Chen09", but as our method is orthogonal with semi-supervising methods, it is possible to further improve the accuracy by combing these techniques.

We investigate the effects of the three feature types we proposed in this paper. We in turn activate each feature type and their combinations in the evaluation, while during the training we keep all types of feature due to the limitation of time. We conduct this experiment based on the system "Order 2 reranked" for English. The result is shown in Table 3. The first row represents the system with all feature types activated; others are systems with corresponding feature sets activated in the evaluation phase. Here "Ch" stands for the chain-type feature set, "Trim" stands for the trimmed subtree feature set, and "Sib" stands for the sibling subtree feature set.

System	UAS
$\text{Reranker}_{\text{Ch+Trim+Sib}}$	93.37
$\text{Reranker}_{\text{Ch}}$	92.41
$\text{Reranker}_{\text{Trim}}$	92.77
$\text{Reranker}_{\text{Ch+Trim}}$	93.03
$\text{Reranker}_{\text{Trim+Sib}}$	93.10

Table 3. Influence of activated feature types on English test data. "Ch": chain-type features activated; "Trim": trimmed subtree features activated; "Sib": sibling subtree features activated.

In Table 4 we investigate the influence of miss-matched K values for the training and the evaluation. We traine a separate system for the Chinese dataset using "Order 1" with K=10 in the reranker's training and variant K values in the evaluation. The row "Rerank" shows that even for a small K used in the training, a better accuracy can be achieved with relatively larger K: the highest accuracy for this system is achieved when K=20 in the evaluation. We also show the oracle accuracies among the top-K candidates in the last row.

K	1	10	20	30	50
Rerank	85.44	86.81	87.49	87.45	87.33
Oracle	85.44	89.66	90.70	91.17	91.65

Table 4. Reranking experiment for Chinese with miss-matched K values.

In Table 5 we show the oracle accuracies among top-K candidates using the "Order 2" parser. The oracle accuracies can increase as much as absolutely 5.14% for English and absolutely 5.15% for Chinese compared with the 1-best accuracies.

K	1	10	20	30	50
English	91.88	95.61	96.30	96.65	97.02
Chinese	87.39	90.43	91.28	92.02	92.54

Table 5. Oracle accuracies of top-K candidates.

4.2 Efficiency

We show the training time and the parsing time of the base parser "Order 2" and the pipelined reranking system "Order 2 reranked" in Table 6.

	Training	Parsing
Order 2	1642 min	0.24 sec/sent
Order 2 reranked	3552 min	11.54 sec/sent

Table 6. Training time and parsing speed comparison for English.

Both systems run on a Xeon 2.4GHz CPU. We calculated the parsing time by running the systems on the first 100 sentences on the development data of the two languages. The reranking system takes twice the time than the base parser in the training. It is much slower than the base parser in parsing new sentences, which is mainly due to the time required for outputting the 50-best candidates list; this can be seen as an unavoidable trade-off to obtain high accuracy in the reranking framework.

5. Related Work

McDonald (2005, 2006) proposed an edge-factored parser and a second-order parser that both trained by discriminative online learning methods. Huang (2005) proposed the efficient algorithm for produce K-best list for graph-based parsers, which add a factor of $K \log K$ to the parsing complexity of the base parser. Sangati (2009) has shown that a discriminative parser is very effective at filtering out bad parses from a factorized search space which agreed with the conclusion in (Hall, 2007) that an edge-factored model can reach good oracle performance when generating relatively small K-best list. Successful results have been reported for constituent parse reranking in (Collins, 2000; Charniak and Johnson, 2005; Huang, 2008), in which feature sets defined on constituent parses have been proposed that are able to capture rich non-local information. These feature sets, however, cannot be directly applied to parse tree under dependency grammar. Attempts have been made to use similar feature sets in dependency parse reranking, which include the work in (Hall, 2007) that defined a feature set similar with the one in (Charniak and Johnson, 2005). Hayashi in (Hayashi et al., 2011) presented a forest reranking model which applied third-order factorizations emulating Model 1 and Model 2 in (Koo and Collins, 2010) on the search space of the reranker.

6. Conclusion

We have proposed a novel feature set for dependency parse reranking that successfully extracts complex structures for collecting linguistic evidence, and efficient feature back-off strategy is proposed to relieve data sparsity. Through experiment we confirmed the effectiveness and efficiency of our method, and observed significant improvement over the base system as well as other known systems.

To further improve the proposed method, we mention several possibilities for our future work. An advantage of the reranking framework we used is that it has no overlap with many of the semi-supervised parsing methods, such as word clustering (Koo et al., 2008) and subtree features integration using auto-parsed data (Chen et al., 2009). We are interested in the performance of our system when combining with these methods. Another interesting approach is to incorporate information from large-scale structured data, such as case frame (Kawahara and Kurohashi, 2006), which provides lexical predicate-argument selection preference and is an effective way to help to overcome data sparse problem in discriminative learning. While the relatively complex data structure in the case frame prohibits its incorporation in any existing factorization methods, it can be well utilized in the reranking framework with the proposed feature set.

References

E. Charniak and M. Johnson. 2005. Coarse-to-fine N-best Parsing and MaxEnt Discriminative Reranking. In Proceedings of the 43rd ACL.

M. Collins. 2000. Discriminative Reranking for Natural Language Parsing. In Proceedings of the ICML.

M. Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In Proceedings of the 7th EMNLP, pages 1–8.

W. Chen, J. Kazama, K. Uchimoto and K. Torisawa. 2009. Improving Dependency Parsing with Subtrees from Auto-Parsed Data, In Proceedings of EMNLP2009, pages 570-579.

X. Duan, J. Zhao, and B. Xu. 2007. Probabilistic Models for Action-based Chinese Dependency Parsing. In Proceedings of ECML/ECPPKDD.

J. Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In Proceedings of the 16th COLING, pages 340–345.

K. Hall. 2007. K-best Spanning Tree Parsing. In Proceedings of ACL 2007.

K. Hayashi, T. Watanabe, M. Asahara and Y. Matsumoto. 2011. Third-order Variational Reranking

on Packed-Shared Dependency Forests. In Proceedings of EMNLP 2011, pages 1479-1488.

L. Huang and D. Chiang. 2005. Better K-best Parsing. In Proceedings of the IWPT, pages 53–64.

L. Huang. 2008. Forest reranking: Discriminative Parsing with Non-local Features. In Proceedings of the 46th ACL, pages 586–594.

D. Kawahara and S. Kurohashi. 2006. Case Frame Compilation from the Web Using High performance Computing. In Proceedings of the 5th International Conference on Language Resources and Evaluation.

T. Koo, X. Carreras, and M. Collins. 2008. Simple Semi-supervised Dependency Parsing. In Proceedings of the 46th ACL, pages 595–603.

T. Koo and M. Collins. 2010. Efficient Third-order Dependency Parsers. In Proceedings of the 48th ACL, pages 1–11.

A. F. T. Martins, N. A. Smith, and E. P. Xing. 2010. Turbo Parsers: Dependency Parsing by Approximate Variational Inference. In Proceedings of EMNLP 2010, pages 34–44.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online Large-Margin Training of Dependency Parsers. In Proceedings of the 43rd ACL, pages 91–98.

R. McDonald and F. Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In Proceedings of the 11th EACL, pages 81–88.

A. Ratnaparkhi. 1996. A Maximum Entropy Model for Part-Of-Speech Tagging. In Proceedings of the 1st EMNLP, pages 133–142.

F. Sangati, W. Zuidema, and R. Bod. 2009. A Generative Re-ranking Model for Dependency Parsing. In Proceedings of the 11th IWPT, pages 238–241.

J. Suzuki, H. Isozaki, X. Carreras, and M. Collins. 2009. An Empirical Study of Semi-supervised Structured Conditional Models for Dependency Parsing. In Proceedings of EMNLP 2009, pages 551–560.

H. Yamada and Y. Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In Proceedings of the IWPT 2003, pages 195–206.

K. Yu, D. Kawahara, and S. Kurohashi. 2008. Chinese Dependency Parsing with Large Scale Automatically Constructed Case Structures. In Proceedings of Coling 2008, pages 1049–1056.

Y. Zhang and S. Clark. 2008. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing. In Proceedings of EMNLP 2008, pages 562–571.

Y. Zhang and J. Nivre. 2011. Transition-based Dependency Parsing with Rich Non-local Features. In Proceedings of ACL 2011, page 188-193.