Efficient Stacked Dependency Parsing by Forest Reranking

Katsuhiko Hayashi and Shuhei Kondo and Yuji Matsumoto
Graduate School of Information Science Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-0192, Japan
{katsuhiko-h,shuhei-k,matsu}@is.naist.jp

Abstract

This paper proposes a discriminative forest reranking algorithm for dependency parsing that can be seen as a form of efficient stacked parsing. A dynamic programming shift-reduce parser produces a packed derivation forest which is then scored by a discriminative reranker, using the 1-best tree output by the shift-reduce parser as guide features in addition to third-order graph-based features. To improve efficiency and accuracy, this paper also proposes a novel shift-reduce parser that eliminates the spurious ambiguity of arc-standard transition systems. Testing on the English Penn Treebank data, forest reranking gave a state-of-the-art unlabeled dependency accuracy of 93.12.

1 Introduction

There are two main approaches of data-driven dependency parsing – one is graph-based and the other is transition-based.

In the graph-based approach, global optimization algorithms find the highest-scoring tree with locally factored models (McDonald et al., 2005). While third-order graph-based models achieve state-of-the-art accuracy, it has $O(n^4)$ time complexity for a sentence of length n. Recently, some pruning techniques have been proposed to improve the efficiency of third-order models (Rush and Petrov, 2012; Zhang and McDonald, 2012).

The transition-based approach usually employs the shift-reduce parsing algorithm with linear-time complexity (Nivre, 2008). It greedily chooses the transition with the highest score and the resulting transition sequence is not always globally optimal. The beam search algorithm improves parsing flexibility in deterministic parsing (Zhang and Clark, 2008; Zhang and Nivre, 2011), and dynamic programming makes beam search more efficient (Huang and Sagae, 2010).

There is also an alternative approach that integrates graph-based and transition-based models (Sagae and Lavie, 2006; Zhang and Clark, 2008; Nivre and McDonald, 2008; Martins et al., 2008). Martins et al. (2008) formulated their approach as stacking of parsers where the output of the first-stage parser is provided to the second as guide features. In particular, they used a transition-based parser for the first stage and a graph-based parser for the second stage. The main drawback of this approach is that the efficiency of the transition-based parser is sacrificed because the second-stage employs full parsing.

This paper proposes an efficient stacked parsing method through discriminative reranking with higher-order graph-based features, which works on the forests output by the first-stage dynamic programming shift-reduce parser and integrates non-local features efficiently with cube-pruning (Huang and Chiang, 2007). The advantages of our method are as follows:

- Unlike the conventional stacking approach, the first-stage shift-reduce parser prunes the search space of the second-stage graph-based parser.
- In addition to guide features, the second-stage graph-based parser can employ the scores of the first-stage parser which cannot be incorpo-

Transactions of the Association for Computational Linguistics, 1 (2013) 139–150. Action Editor: Joakim Nivre.
Submitted 12/2012; Revised 3/2013; Published 5/2013.

$$\text{axiom}(c_0): \quad 0 : (0, 1, \mathbf{w}_0) : \emptyset$$

$$\text{goal}(c_{2n}): \quad 2n : (0, n, \mathbf{s}_0) : \emptyset$$

$$\text{shift}: \quad \frac{\overbrace{\ell : (\_, j, \mathbf{s}_d|\mathbf{s}_{d-1}|\dots|\mathbf{s}_1|\mathbf{s}_0) : \_}^{\text{state } p}}{\ell + 1 : (j, j+1, \mathbf{s}_{d-1}|\mathbf{s}_{d-2}|\dots|\mathbf{s}_0|\mathbf{w}_j) : (p)} \; i < n$$

$$\text{reduce}_{\curvearrowleft}: \quad \frac{\overbrace{\_ : (i, j, \mathbf{s}'_d|\mathbf{s}'_{d-1}|\dots|\mathbf{s}'_1|\mathbf{s}'_0) : \pi'}^{\text{state } p} \quad \overbrace{\ell : (j, k, \mathbf{s}_d|\mathbf{s}_{d-1}|\dots|\mathbf{s}_1|\mathbf{s}_0) : \pi}^{\text{state } q}}{\ell + 1 : (i, k, \mathbf{s}'_d|\mathbf{s}'_{d-1}|\dots|\mathbf{s}'_1|\mathbf{s}'_0 \curvearrowleft \mathbf{s}_0) : \pi'} \; \mathbf{s}'_0.\text{h.w} \neq \mathbf{w}_0 \wedge p \in \pi$$

$$\text{reduce}_{\curvearrowright}: \quad \frac{\overbrace{\_ : (i, j, \mathbf{s}'_d|\mathbf{s}'_{d-1}|\dots|\mathbf{s}'_1|\mathbf{s}'_0) : \pi'}^{\text{state } p} \quad \overbrace{\ell : (j, k, \mathbf{s}_d|\mathbf{s}_{d-1}|\dots|\mathbf{s}_1|\mathbf{s}_0) : \pi}^{\text{state } q}}{\ell + 1 : (i, k, \mathbf{s}'_d|\mathbf{s}'_{d-1}|\dots|\mathbf{s}'_1|\mathbf{s}'_0 \curvearrowright \mathbf{s}_0) : \pi'} \; p \in \pi$$

Figure 1: The arc-standard transition-based dependency parsing system with dynamic programming: _ means "take anything". $\mathbf{a}\curvearrowright\mathbf{b}$ denotes that a tree $\mathbf{b}$ is attached to a tree $\mathbf{a}$.

rated in standard graph-based models.

- In contrast to joint transition-based/graph-based approaches (Zhang and Clark, 2008; Bohnet and Kuhn, 2012) which require a large beam size and make dynamic programming impractical, our two-stage approach can integrate both models with little loss of efficiency.

In addition, the elimination of spurious ambiguity from the arc-standard shift-reduce parser improves the efficiency and accuracy of our approach.

$\mathbf{s}_0$.h.t $\circ$ $\mathbf{s}_0$.lc.t $\circ$ $\mathbf{s}_0$.lc2.t $\mathbf{s}_0$.h.t $\circ$ $\mathbf{s}_0$.rc.t $\circ$ $\mathbf{s}_0$.rc2.t
$\mathbf{s}_1$.h.t $\circ$ $\mathbf{s}_1$.lc.t $\circ$ $\mathbf{s}_1$.lc2.t $\mathbf{s}_1$.h.t $\circ$ $\mathbf{s}_1$.rc.t $\circ$ $\mathbf{s}_1$.rc2.t
$\mathbf{s}_0$.h.t $\circ$ $\mathbf{s}_0$.lc.t $\circ$ $\mathbf{s}_0$.lc2.t $\circ$ $\mathbf{q}_0$.t
$\mathbf{s}_0$.h.t $\circ$ $\mathbf{s}_0$.rc.t $\circ$ $\mathbf{s}_0$.rc2.t $\circ$ $\mathbf{q}_0$.t
$\mathbf{s}_0$.h.t $\circ$ $\mathbf{s}_1$.h.t $\circ$ $\mathbf{q}_0$.t $\circ$ $\mathbf{q}_1$.t
$\mathbf{s}_0$.h.w $\circ$ $\mathbf{s}_1$.h.t $\circ$ $\mathbf{q}_0$.t $\circ$ $\mathbf{q}_1$.t

Table 1: Additional feature templates for shift-reduce parsers: $\mathbf{q}$ denotes input queue. h, lc and rc are head, leftmost child and rightmost child of a stack element $\mathbf{s}$. lc2 and rc2 denote the second leftmost and rightmost children. t and w are a part-of-speech (POS) tag and a word.

2 Arc-Standard Shift-Reduce Parsing

We use a beam search shift-reduce parser with dynamic programming as our baseline system. Figure 1 shows it as a deductive system (Shieber et al., 1995). A state is defined as the following:

$$\ell : (i, j, \mathbf{s}_d|\mathbf{s}_{d-1}|\dots|\mathbf{s}_1|\mathbf{s}_0) : \pi$$

where ℓ is the step size, $[i, j]$ is the span of the topmost stack element $\mathbf{s}_0$, and $\mathbf{s}_d|\mathbf{s}_{d-1}|\dots|\mathbf{s}_1$ shows a stack with d elements at the top, where d is the window size used for defining features. The axiom is initialized with an input sentence of length n, $x = \mathbf{w}_0 \dots \mathbf{w}_n$ where $\mathbf{w}_0$ is a special root symbol \$0. The system takes $2n$ steps for a complete analysis.

π is a set of pointers to the predictor states, each of which is the state just before shifting the root word of $\mathbf{s}_0$ into stack[1]. Dynamic programming merges equivalent states in the same step if they have the same feature values. We add the feature templates shown in Table 1 to Huang and Sagae (2010)'s feature templates.

Dynamic programming not only makes the shift-reduce parser with beam search more efficient but also produces a packed forest that encodes an exponential number of dependency trees. A packed dependency forest can be represented by a weighted (directed) hypergraph. A weighted hypergraph is a pair $H = \langle V, E \rangle$, where V is the set of vertices and E is the set of hyperedges. Each hyperedge $e \in E$ is a tuple $e = \langle T(e), h(e), f_e \rangle$, where $h(e) \in V$ is

[1]Huang and Sagae (2010)'s dynamic programming is based on a notion of a *push computation* (Kuhlmann et al., 2011). The details are out of scope here and readers may refer to the paper.

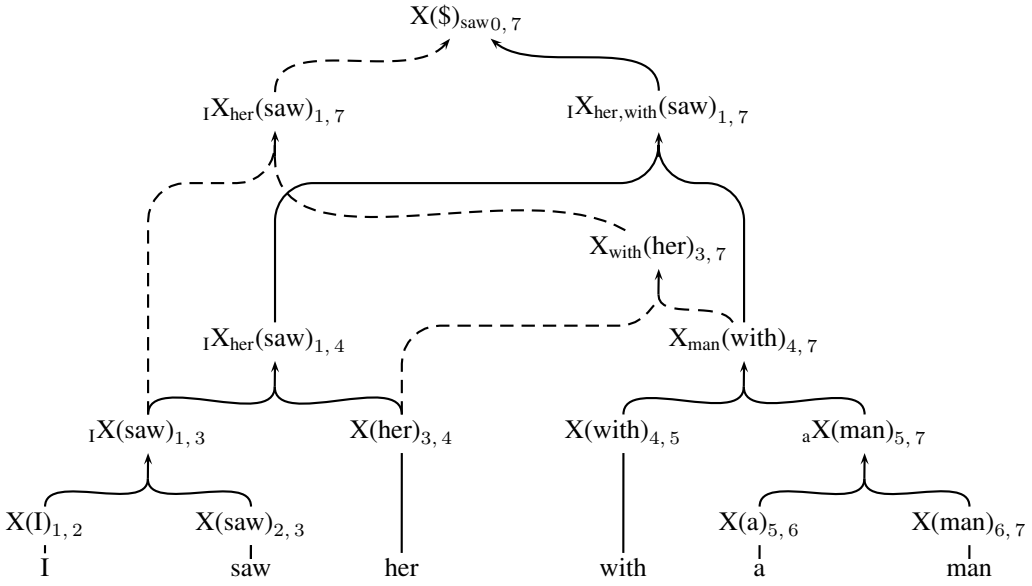

Figure 2: An example of packed dependency (derivation) forest: each vertex has information about the topmost stack element of the corresponding state to it.

its head vertex, $T(e) \in V^+$ is an ordered list of tail vertices, and f_e is a weight for e.

Figure 2 shows an example of a packed forest. Each binary hyperedge corresponds to a reduce action, and each leaf vertex corresponds to a shift action. Each vertex also corresponds to a state, and parse histories on the states can be encoded into the vertices. In the example, information about the topmost stack element is attached to the corresponding vertex marked with a non-terminal symbol X.

Weights are omitted in the example. In practice, we attach each reduction weight to the corresponding hyperedge, and add the shift weight to the reduction weight when a shifted word is reduced.

3 Arc-Standard Shift-Reduce Parsing without Spurious Ambiguity

One solution to remove spurious ambiguity in the arc-standard transition system is to give priority to the construction of left arcs over that of right arcs (or vice versa) like Eisner (1997). For example, an Earley dependency parser (Hayashi et al., 2012) attaches all left dependents to a word before right dependents. The parser uses a scan action to stop the construction of left arcs.

We apply this idea to the arc-standard transition system and show the resulting transition system in Figure 3. We introduce the $*$ symbol to indicate that the root node of the topmost element on the stack has not been scanned yet. The shift and reduce$_\curvearrowright$ actions can be used only when the root of the topmost element on the stack has already been scanned, and all left arcs are always attached to the head before the head is scanned.

The arc-standard shift-reduce parser without spurious ambiguity takes $3n$ steps to finish parsing, and the additional n scan actions add surplus vertices and (unary) hyperedges to a packed forest. However, it is easy to remove them from the packed forest because the consequent state of a scan action has a unique antecedent state and all the hyperedges going out from a vertex corresponding to the consequent state can be attached to the vertex corresponding to the antecedent state. The scan weight of the removed unary hyperedge is added to each weight of the hyperedges attached to the antecedent.

4 Experiments (Spurious Ambiguity vs. Non-Spurious Ambiguity)

We conducted experiments on the English Penn Treebank (PTB) data to compare spurious and non-spurious shift-reduce parsers. We split the WSJ part of PTB into sections 02-21 for training, section 22 for development, and section 23 for test. We used the head rules (Yamada and Matsumoto, 2003) to convert phrase structure to dependency structure.

$$\text{axiom}(c_0): \quad 0 : (0, 1, \mathbf{w}_0) : \emptyset$$

$$\text{goal}(c_{3n}): \quad 3n : (0, n, \mathbf{s}_0) : \emptyset$$

$$\text{shift}: \quad \frac{\overbrace{\ell : (\_, j, \mathbf{s}_d|\mathbf{s}_{d-1}|\dots|\mathbf{s}_1|\mathbf{s}_0) : \_}^{\text{state } p}}{\ell + 1 : (j, j+1, \mathbf{s}_{d-1}|\mathbf{s}_{d-2}|\dots|\mathbf{s}_0|\mathbf{w}_j^*) : (p)} \; j < n$$

$$\text{scan}: \quad \frac{\ell : (i, j, \mathbf{s}_d|\mathbf{s}_{d-1}|\dots|\mathbf{s}_1|\mathbf{s}_0^*) : \pi}{\ell + 1 : (i, j, \mathbf{s}_d|\mathbf{s}_{d-1}|\dots|\mathbf{s}_1|\mathbf{s}_0) : \pi}$$

$$\text{reduce}_{\curvearrowleft}: \quad \frac{\overbrace{\_ : (i, j, \mathbf{s}'_d|\mathbf{s}'_{d-1}|\dots|\mathbf{s}'_0|\mathbf{s}'_0) : \pi'}^{\text{state } p} \quad \overbrace{\ell : (j, k, \mathbf{s}_d|\mathbf{s}_{d-1}|\dots|\mathbf{s}_1|\mathbf{s}_0^*) : \pi}^{\text{state } q}}{\ell + 1 : (i, k, \mathbf{s}'_d|\mathbf{s}'_{d-1}|\dots|\mathbf{s}'_1|\mathbf{s}'_0 \curvearrowleft \mathbf{s}_0^*) : \pi'} \; \mathbf{s}'_0.\text{h}.\mathbf{w} \neq \mathbf{w}_0 \wedge p \in \pi$$

$$\text{reduce}_{\curvearrowright}: \quad \frac{\overbrace{\_ : (i, j, \mathbf{s}'_d|\mathbf{s}'_{d-1}|\dots|\mathbf{s}'_1|\mathbf{s}'_0) : \pi'}^{\text{state } p} \quad \overbrace{\ell : (j, k, \mathbf{s}_d|\mathbf{s}_{d-1}|\dots|\mathbf{s}_1|\mathbf{s}_0) : \pi}^{\text{state } q}}{\ell + 1 : (i, k, \mathbf{s}'_d|\mathbf{s}'_{d-1}|\dots|\mathbf{s}'_1|\mathbf{s}'_0 \curvearrowright \mathbf{s}_0) : \pi'} \; p \in \pi$$

Figure 3: The dynamic programming arc-standard transition-based deductive system without spurious ambiguity: the symbol represents that the root node of the topmost element on the stack has not been scanned yet.

			8	16	32	64	128
dev.	spurious	UAS (w/o punc.)	92.5 (93.5)	**92.7** (93.6)	92.6 (93.6)	92.6 (93.6)	92.6 (93.6)
		sec. (per sent.)	0.01	**0.017**	0.03	**0.06**	0.13
	non-sp.	UAS (w/o punc.)	92.5 (**93.6**)	92.6 (93.6)	92.6 (93.6)	92.6 (93.6)	92.6 (93.6)
		sec. (per sent.)	0.01	0.018	0.03	0.07	0.13
test	spurious	UAS (w/o punc.)	92.7 (93.3)	92.7 (93.3)	92.7 (93.3)	92.8 (93.3)	92.8 (93.3)
		sec. (per sent.)	0.01	**0.017**	0.03	0.06	0.13
	non-sp.	UAS (w/o punc.)	**92.8** (**93.4**)	**92.9** (**93.5**)	**92.9** (**93.5**)	**92.9** (**93.5**)	**92.9** (**93.5**)
		sec. (per sent.)	0.01	0.018	0.03	0.06	0.13

Table 2: Unlabeled accuracy scores (UAS) and parsing times (+forest dumping times, second per sentence) for parsing development (WSJ22) and test (WSJ23) data with spurious shift-reduce and proposed shift-reduce parser (non-sp.) using several beam sizes.

We used an early update version of the averaged perceptron algorithm (Collins and Roark, 2004; Huang et al., 2012) to train two shift-reduce dependency parsers with beam size of 12.

Table 2 shows experimental results of parsing the development and test datasets with each of the spurious and non-spurious shift-reduce parsers using several beam sizes. Parsing accuracies were evaluated by unlabeled accuracy scores (UAS) with and without punctuations. The parsing times were measured on an Intel Core i7 2.8GHz. The average cpu time (per sentence) includes that of dumping packed forests. This result indicates that the non-spurious parser achieves better accuracies than the spurious parser without loss of efficiency.

beam size	8	32	128
% of distinct trees (10)	93.5	94.8	95.0
% of distinct trees (100)	81.8	84.9	87.2
% of distinct trees (1000)	70.6	73.1	77.6
% of distinct trees (10000)	62.1	64.3	65.6

Table 3: The percentages of distinct dependency trees in 10, 100, 1000 and 10000 best trees extracted from spurious forests with several beam sizes.

Figure 4 shows oracle unlabeled accuracies of spurious k-best lists, non-spurious k-best lists, spurious forests, and non-spurious forests. We extract an oracle tree from each packed forest using the for-

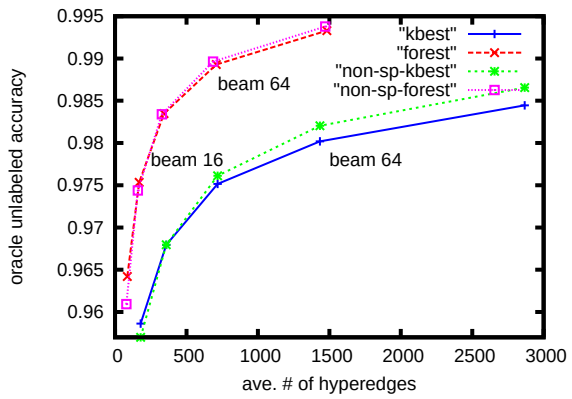

Figure 4: Each plot shows oracle unlabeled accuracies of spurious k-best lists, spurious forests, and non-spurious forests. The oracle accuracies are evaluated using UAS with punctuations.

est oracle algorithm (Huang, 2008). Both forests produce much better results than the k-best lists, and non-spurious forests have almost the same oracle accuracies as spurious forests.

However, as shown in Table 3, spurious forests encode a number of non-unique dependency trees while all dependency trees in non-spurious forests are distinct from each other.

5 Forest Reranking

5.1 Discriminative Reranking Model

We define a reranking model based on the graph-based features as the following:

$$\hat{y} = \operatorname*{argmax}_{y \in H} \alpha \cdot \mathbf{f}_{\mathrm{g}}(x, y) \quad (1)$$

where α is a weight vector, $\mathbf{f}_{\mathrm{g}}$ is a feature vector (g indicates "graph-based"), x is the input sentence, y is a dependency tree and H is a dependency forest. This model assumes a hyperedge factorization which induces a decomposition of the feature vector as the following:

$$\alpha \cdot \mathbf{f}_{\mathrm{g}}(x, y) = \sum_{e \in y} \alpha \cdot \mathbf{f}_{\mathrm{g},e}(e). \quad (2)$$

The search problem can be solved by simply using the (generalized) Viterbi algorithm (Klein and Manning, 2001). When using non-local features, the hyperedge factorization is redefined to the following:

$$\alpha \cdot \mathbf{f}_{\mathrm{g}}(x, y) = \sum_{e \in y} \alpha \cdot \mathbf{f}_{\mathrm{g},e}(e) + \alpha \cdot \mathbf{f}_{\mathrm{g},e,N}(e) \quad (3)$$

where $\mathbf{f}_{\mathrm{g},e,N}$ is a non-local feature vector. Though the cube-pruning algorithm (Huang and Chiang, 2007) is an approximate decoding technique based on a k-best Viterbi algorithm, it can calculate the non-local scores efficiently.

The baseline score can be taken into the reranker as a linear interpolation:

$$\hat{y} = \operatorname*{argmax}_{y \in H} \beta \cdot \mathbf{sc}_{\mathrm{tr}}(x, y) + \alpha \cdot \mathbf{f}_{\mathrm{g}}(x, y) \quad (4)$$

where $\mathbf{sc}_{\mathrm{tr}}$ is the score from the baseline parser (tr indicates "transition-based"), and β is a scaling factor.

5.2 Features for Discriminative Model

5.2.1 Local Features

While the inference algorithm is a simple Viterbi algorithm, the discriminative model can use all tri-sibling features and some grand-sibling features[2] (Koo and Collins, 2010) as a local scoring factor in addition to the first- and sibling second-order graph-based features. This is because the first stage shift-reduce parser uses features described in Section 2 and this information can be encoded into vertices of a hypergraph.

The reranking model also uses guide features extracted from the 1-best tree predicted by the first stage shift-reduce parser. We define the guide features as first-order relations like those used in Nivre and McDonald (2008) though our parser handles only unlabeled and projective dependency structures. We summarize the features for discriminative reranking model as the following:

- First- and second-order features: these features are the same as those used in MST parser[3].
- Grand-child features: we define tri-gram POS features with POS tags of grand parent, parent, and rightmost or leftmost child.
- Tri-sibling features: we define tri-gram features with three POS-tags of child, sibling, and tri-sibling. We also define tri-gram features with one word and two POS tags of the above.

[2]The grand-child and grand-sibling features can be used only when interacting with the leftmost or rightmost child and sibling. In case of local reranking, we did not use grand-sibling features because in our experiments, they were not effective.

[3]http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html

- Guide feaures: we define a feature indicating whether an arc from a child to its parent is present in the 1-best tree predicted by the first-stage shift-reduce parser, conjoined with the POS tags of the parent and child.

- PP-Attachment features: when a parent word is a preposition, we define tri-gram features with the parent word and POS tags of grand parent and the rightmost child.

5.2.2 Non-local Features

To define richer features as a non-local factor, we extend a local reranking algorithm by augmenting each k-best item with all child vertices of its head vertex[4]. Information about all children enables the reranker to calculate the following features when reducing the head vertex:

- Grand-child features: we define tri-gram features with one word and two POS tags of grand parent, parent, and child.

- Grand-sibling features: we define 4-gram POS features with POS tags of grand parent, parent, child and sibling. We also define coordination features with POS tags of grand parent, parent and child when the sibling word is a coordinate conjunction.

- Valency features: we define a feature indicating the number of children of a head, conjoined with each of its word and POS tag.

When using non-local features, we removed the local grand-child features from the model.

5.3 Oracle for Discriminative Training

A discriminative reranking model is trained on packed forests by using their oracle trees as the correct parse. More accurate oracles are essential to train a discriminative reranking model well.

While large size forests have much more accurate oracles than small size forests, large forests have too many hyperedges to train a discriminative model on them, as shown in Figure 4. The usual forest reranking algorithms (Huang, 2008; Hayashi et al., 2011) remove low quality hyperedges from large forests by using inside-outside forest pruning.

However, producing large forests and pruning them is computationally very expensive. Instead, we propose a simpler method to produce small forests which have more accurate oracles by forcing the beam search shift-reduce parser to keep the correct state in the beam buffer. As a result, the correct tree will always be encoded in a packed forest.

6 Experiments (Discriminative Reranking)

6.1 Experimental Setting

Following (Huang, 2008), the training set (WSJ02-21) is split into 20 folds, and each fold is parsed by each of the spurious and non-spurious shift-reduce parsers using beam size 12 with the model trained on sentences from the remaining 19 folds, dumping the outputs as packed forests.

The reranker is modeled by either equation (1) or (4). By our preliminary experiments using development data (WSJ22), we modeled the reranker with equation (1) when training, and with equation (4) when testing[5] (i.e., the scores of the first-stage parser are not considered during training of the reranking model). This prevents the discriminative reranking features from *under-training* (Sutton et al., 2006; Hollingshead and Roark, 2008).

A discriminative reranking model is trained on the packed forests by using the averaged perceptron algorithm with 5 iterations. When training non-local reranking models, we set k-best size of cube-pruning to 5.

For dumping packed forests for test data, spurious and non-spurious shift-reduce parsers are trained by the averaged perceptron algorithm. In all experiments on English data, we fixed beam size to 12 for training both parsers.

6.2 Test with Gold POS tags

We show the comparison of dumped spurious and non-spurious packed forests for training data in Table 4. Both oracle accuracies are 100.0 due to the

[4]If each item is augmented with richer information, even features based on the entire subtree can be defined.

[5]The scaling factor β was tuned by minimum error rate training (MERT) algorithm (Och, 2003) using development data. The MERT algorithm is suited to tune low-dimensional parameters. The β was set to about 1.2 in case of local reranking, and to about 1.5 in case of non-local reranking.

system	w/ rerank.	sec. (per sent.)	UAS (w/o punc.)
sr (12)	–	0.011	92.8 (93.3)
(8)	w/ local	0.009 + 0.0056	93.03 (93.69)
(12)	w/ local	0.011 + 0.0079	93.03 (93.68)
(32)	w/ local	0.03 + 0.019	93.07 (93.67)
(64)	w/ local	0.06 + 0.039	93.0 (93.61)
(12, k=3)	w/ non-local	0.011 + 0.0085	93.17 (93.78)
(64, k=3)	w/ non-local	0.06 + 0.046	93.19 (93.78)
non-sp sr (12)	–	0.012	92.9 (93.5)
(8)	w/ local	0.01 + 0.005	93.05 (93.73)
(12)	w/ local	0.012 + 0.0074	93.21 (93.87)
(32)	w/ local	0.031 + 0.0184	93.22 (93.84)
(64)	w/ local	0.061 + 0.0375	93.23 (93.83)
(12, k=3)	w/ non-local	0.012 + 0.0083	93.28 (93.9)
(64, k=3)	w/ non-local	0.061 + 0.045	93.39 (93.96)

Table 7: Unlabeled accuracy scores and cpu times per sentence (parsing+reranking) when parsing and reranking test data (WSJ23) with gold POS tags: shift-reduce parser is denoted as sr (beam size, k: k-best size of cube pruning).

	sp.	non-sp.
ave. # of hyperedges	141.9	133.3
ave. # of vertices	199.1	187.6
ave. % of distinct trees	82.5	100.0
1-best UAS w/ punc.	92.5	92.6
oracle UAS w/ punc.	100.0	100.0

Table 4: Comparison of spurious (sp.) and non-spurious (non-sp.) forests: each forest is produced by baseline and proposed shift-reduce parsers using beam size 12 for 39832 training sentences with gold POS tags.

reranker	pre-comp.	training
spurious	16.4 min.	34.9 min.
non-spurious	15.5 min.	32.9 min.
spurious non-local	17.3 min.	64.3 min.
non-spurious non-local	16.2 min.	60.3 min.

Table 5: Training times on both spurious and non-spurious packed forests (beam 12): pre-comp. denotes cpu time for feature extraction and attaching features to all hyperedges. The non-local models were trained setting k-best size of cube-pruning to 5, and non-local features were calculated on-the-fly while training.

method described in Section 5.3. The 1-best accuracy of the non-spurious forests is higher than that of the spurious forests. As we expected, the results show that there are many non-unique dependency trees in the spurious forests. The spurious forests also get larger than the non-spurious forests.

Table 5 shows how long the training on spurious and non-spurious forests took on an Opteron 8356 2.3GHz. It is clear from the results that training on non-spurious forests is more efficient than that on spurious forests.

Table 6 shows the statistics of spurious and non-spurious packed forests dumped by shift-reduce parsers using beam size 12 for test data. The trends are similar to those for training data shown in Table 4. We show the results of the forest reranking algorithms for test data in Table 7. Each spurious and non-spurious shift-reduce parser produces packed forests using four beam sizes 8, 12, 32, and 64. The reranking on non-spurious forests achieves better accuracies and is slightly faster than that on spurious forests consistently.

6.3 Test with Automatic POS tags

To compare the proposed reranking system with other systems, we evaluate its parsing accuracy on test data with automatic POS tags. We used the Stanford POS tagger[6] with a model trained on sections 02-21 to tag development and test data, and used 10-way jackknifing to tag training data. The tagging accuracies on training, development, and test data were 97.1, 97.2, and 97.5.

Table 8 lists the accuracy and parsing speed of

[6] http://nlp.stanford.edu/software/tagger.shtml

	sp.	non-sp.
ave. # of hyperedges	127.0	119.1
ave. # of vertices	178.6	168.5
ave. % of distinct trees	82.4	100.0
1-best UAS w/ punc.	92.8	92.9
oracle UAS w/ punc.	97.0	97.0

Table 6: Comparison of spurious (sp.) and non-spurious (non-sp.) forests: each forest is produced by baseline and proposed shift-reduce parsers using beam size 12 for test data (WSJ23) with gold POS tags.

system	tok./sec.	UAS w/o punc.
sr (12)	2130	92.5
w/ local (12)	1290	92.8
non-sp sr (12)	1950	92.6
w/ local (12)	1300	92.98
w/ non-local (12, k=1)	1280	93.1
w/ non-local (12, k=3)	1180	**93.12**
w/ non-local (12, k=12)	1060	**93.12**
Huang10 sr (8)	782	92.1
Rush12 sr (16)	4780	92.5
Rush12 sr (64)	1280	92.7
Koo10	–	93.04
Rush12 third	20	93.3
Rush12 vine	4400	93.1
H-Zhang12 third	50	92.81
H-Zhang12 (label)	220	93.06
Y-Zhang11 (64, label)	680	92.9
Bohnet12 (80, label)	120	93.39

Table 8: Comparison with other systems: the results were evaluated on testing data (WSJ23) with automatic POS tags: label means labeled dependency parsing and the cpu times of our systems were taken on Intel Core i7 2.8GHz.

our proposed systems together with results from related work. The parsing times are reported in tokens/second for comparison. Note that, however, the difference of the parsing time does not represent the efficiency of the algorithm directly because each system was implemented in different programming language and the times were measured on different environments.

The accuracy of local reranking on non-spurious forests is the best among unlabeled shift-reduce parsers, but slightly behind the third-order graph-based systems (Koo and Collins, 2010; Zhang and McDonald, 2012; Rush and Petrov, 2012). It is likely that the difference comes from the fact that our local reranking model can define only some of the grand-child related features.

	w/ guide.	w/o guide.
UAS / feature	**92.98**	92.86
Linear (first)	89,330	89,215
CorePos (first)	1,047,948	1,053,796
TwoObs (first)	1,303,911	1,325,990
Sibling (second)	290,291	292,849
Trip (second)	19,333	19,267
Grand-child	16,975	16,951
Guide	4,934	–
Tri-sibling	277,770	279,720
PP-Attachment	32,695	32,993
total	3,083,187	3,110,781

Table 9: Accuracy and the number of non-zero weighted features of the local reranking models with and without guide features: the first- and second-order features are named for MSTParser.

To define all grand-child features and other non-local features, we also experimented with the non-local reranking algorithm on non-spurious packed forests. It achieved almost the same accuracy as the previous third-order graph-based algorithms. Moreover, the computational overhead is very small when setting k-best size of cube-pruning small.

6.4 Analysis

One advantage of our reranking approach is that guide features can be defined as in stacked parsing. To analyze the effect of the guide features on parsing accuracy, we remove the guide features from baseline reranking models with and without non-local features used in Section 6.3. The results are shown in Table 9 and 10. The parsing accuracies of the baseline reranking models are better than those of the models without guide features though the number of guide features is not large. Additionally, each model with guide features is smaller than that without guide features. This indicates that stacking has a good effect on training the models.

To further investigate the effects of guide features, we tried to define unlabeled versions of the second-order guide features used in (Martins et al., 2008; McClosky et al., 2012). However, these features did not produce good results, and investigation to find the cause is an important future work.

We also examined parsing errors in more detail. Table 11 shows root and sentence complete rates of three systems, the non-spurious shift-reduce

	w/ guide.	w/o guide.
UAS / feature	**93.12**	93.04
Linear (first)	88,634	88,934
CorePos (first)	1,035,897	1,045,242
TwoObs (first)	1,274,834	1,301,103
Sibling (second)	284,341	288,796
Trip (second)	19,201	19,219
Guide	4,916	–
Tri-sibling	272,418	276,025
PP-Attachment	32,085	32,577
Grand-child	718,064	730,663
Grand-sibling	72,865	73,103
Valency	49,262	49,677
total	3,852,517	3,905,339

Table 10: Accuracy and the number of non-zero weighted features of the non-local reranking models with and without guide features: the first- and second-order features are named for MSTParser.

system	UAS	root	comp.
non-sp sr	92.6	95.8	45.6
local	92.98	96.1	48.1
non-local	**93.12**	**96.3**	**48.2**

Table 11: Unlabeled accuracy, root correct rate, and sentence complete rate: these scores are measured on test data (WSJ23) without punctuations.

parser, local reranking, and non-local reranking. The two reranking systems outperform the shift-reduce parser significantly, and the non-local reranking system is the best among them.

Part of the difference between the shift-reduce parser and reranking systems comes from the correction of coordination errors. Table 12 shows the head correct rate, recall, precision, F-measure and complete rate of coordination structures, by which we mean the head and siblings of a token whose POS tag is CC. The head correct rate denotes how correct a head of the CC token is. The recall, precision, F-measure are measured by counting arcs between the head and siblings. When the head of the CC token is incorrect, all arcs of the coordination structure are counted as incorrect. Therefore, the recall, precision, F-measure are greatly affected by the head correct rate, and though the complete rate of non-local reranking is higher than that of local reranking, the results of the first three measures are lower.

	non-sp sr	local	non-local
head correct	87.73	**88.97**	88.83
recall	82.38	**84.35**	84.11
precision	83.07	**84.57**	83.98
F-measure	82.72	**84.46**	84.05
comp.	62.92	64.52	**65.18**

Table 12: Head correct rate, recall, precision, F-measure, and complete rate of coordination strutures: these are measured on test data (WSJ23).

system	recall	precision	F-measure
non-sp sr	91.58	92.5	92.04
local	91.96	92.95	92.45
non-local	**92.44**	**93.07**	**92.75**

Table 13: Recall, precision, and F-measure of grand-child structures whose grand parent is an artificial root symbol: these are measured on test data (WSJ23).

We assume that the improvements of non-local reranking over the others can be mainly attributed to the better prediction of the structures around the sentence root because most of the non-local features are useful for predicting these structures. Table 13 shows the recall, precision and F-measure of grand-child structures whose grand parent is a sentence root symbol \$. The results support the above assumption. The root correct rate directly influences on prediction of the overall structures of a sentence, and it is likely that the reduction of root prediction errors brings better results.

6.5 Experiments on Chinese

We also experiment on the Penn Chinese Treebank (CTB5). Following Huang and Sagae (2010), we split it into training (secs 001-815 and 1001-1136), development (secs 886-931 and 1148-1151), and test (secs 816-885 and 1137-1147) sets, and use the head rules of Zhang and Clark (2008). The training set is split into 10 folds to dump packed forests for training of reranking models.

We set the beam size of both spurious and non-spurious parsers to 12, and the number of perceptron training iterations to 25 for the parsers and to 8 for both rerankers. Table 14 shows the results for the test sets. As we expected, reranking on non-spurious forests outperforms that on spurious forests.

system	UAS	root	comp.
sr (12)	85.3	78.6	33.4
w/ non-local (12, k=3)	85.8	79.4	34.2
non-sp sr (12)	85.3	78.4	33.7
w/ non-local (12, k=3)	**85.9**	**79.6**	**34.3**

Table 14: Results on Chinese Treebank data (CTB5): evaluations are performed without punctuations.

7 Related Works

7.1 How to Handle Spurious Ambiguity

The graph-based approach employs Eisner and Satta (1999)'s algorithm where spurious ambiguities are eliminated by the notion of split head automaton grammars (Alshawi, 1996).

However, the arc-standard transition-based parser has the spurious ambiguity problem. Cohen et al. (2012) proposed a method to eliminate the spurious ambiguity of shift-reduce transition systems. Their method covers existing systems such as the arc-standard and non-projective transition-based parsers (Attardi, 2006). Our system copes only with the projective case, but is simpler than theirs and we show its efficacy empirically through some experiments.

The arc-eager shift-reduce parser also has a spurious ambiguity problem. Goldberg and Nivre (2012) addressed this problem by not only training with a canonical transition sequence but also with alternate optimal transitions that are calculated dynamically for a current state.

7.2 Methods to Improve Dependency Parsing

Higher-order features like third-order dependency relations are essential to improve dependency parsing accuracy (Koo and Collins, 2010; Rush and Petrov, 2012; Zhang and McDonald, 2012). A reranking approach is one effective solution to introduce rich features to a parser model in the context of constituency parsing (Charniak and Johnson, 2005; Huang, 2008).

Hall (2007) applied a k-best maximum spanning tree algorithm to non-projective dependency analysis, and showed that k-best discriminative reranking improves parsing accuracy in several languages. Sangati et al. (2009) proposed a k-best dependency reranking algorithm using a third-order generative model, and Hayashi et al. (2011) extended it to a forest algorithm. Though forest reranking requires some approximations such as cube-pruning to integrate non-local features, it can explore larger search space than k-best reranking.

The stacking approach (Nivre and McDonald, 2008; Martins et al., 2008) uses the output of one dependency parser to provide guide features for another. Stacking improves the parsing accuracy of second stage parsers on various language datasets. The joint graph-based and transition-based approach (Zhang and Clark, 2008; Bohnet and Kuhn, 2012) uses an arc-eager shift-reduce parser with a joint graph-based and transition-based model. Though it improves parsing accuracy significantly, the large beam size of the shift-reduce parser harms its efficiency. Sagae and Lavie (2006) showed that combining the outputs of graph-based and transition-based parsers can improve parsing accuracies.

8 Conclusion

We have presented a discriminative forest reranking algorithm for dependency parsing. This can be seen as a kind of joint transition-based and graph-based approach because the first-stage parser is a shift-reduce parser and the second-stage reranker uses a graph-based model.

Additionally, we have proposed a dynamic programming arc-standard transition-based dependency parser without spurious ambiguity, along with a heuristic that encodes the correct tree in the output packed forest for reranker training, and shown that forest reranking works well on packed forests produced by the proposed parser.

To improve the accuracy of reranking, we will engage in feature engineering. We need to further investigate effective higher-order guide and non-local features. It also seems promising to extend the unlabeled reranker to a labeled one because labeled information often improves unlabeled accuracy.

In this paper, we adopt a reranking approach, but a rescoring approach is more promising to improve efficiency because it does not have the overhead of dumping packed forests.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. This work was partly

supported by Grant-in-Aid for Japan Society for the Promotion of Science (JSPS) Research Fellowship for Young Scientists.

References

H. Alshawi. 1996. Head automata for speech translation. In *Proc. the ICSLP*.

G. Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proc. of the 10th Conference on Natural Language Learning*, pages 166–170.

B. Bohnet and J. Kuhn. 2012. The best of bothworlds – a graph-based completion model for transition-based parsers. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 77–87.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 173–180.

S. B. Cohen, C. Gómez-Rodríguez, and G. Satta. 2012. Elimination of spurious ambiguity in transition-based dependency parsing. Technical report.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL'04)*.

J. M. Eisner and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 457–464.

J. Eisner. 1997. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 5th International Workshop on Parsing Technologies (IWPT)*, pages 54–65.

Y. Goldberg and J. Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24rd International Conference on Computational Linguistics (Coling 2012)*.

K. Hall. 2007. K-best spanning tree parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 392–399.

K. Hayashi, T. Watanabe, M. Asahara, and Y. Matsumoto. 2011. The third-order variational reranking on packed-shared dependency forests. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1479–1488.

K. Hayashi, T. Watanabe, M. Asahara, and Y. Matsumoto. 2012. Head-driven transition-based parsing with top-down prediction. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 657–665.

K. Hollingshead and B. Roark. 2008. Reranking with baseline system scores and ranks as features. In *CSLU-08-001, Center for Spoken Language Understanding, Oregon Health and Science University*.

L. Huang and D. Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151.

L. Huang and K. Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL'10)*, pages 1077–1086.

L. Huang, S. Fayong, and Y. Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151.

L. Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, pages 586–594.

D. Klein and C. D. Manning. 2001. Parsing and hypergraphs. In *Proceedings of the 7th International Workshop on Parsing Technologies*.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL'10)*, pages 1–11.

M. Kuhlmann, C. Gómez-Rodríguez, and G. Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 673–682.

André F. T. Martins, D. Das, N. A. Smith, and E. P. Xing. 2008. Stacking dependency parsers. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 157–166.

D. McClosky, W. Che, M. Recasens, M. Wang, R. Socher, and C. D. Manning. 2012. Stanfords system for parsing the english web. In *Proceedings of First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL) at NAACL 2012*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98.

J. Nivre and R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-08: HLT*, pages 950–958.

J. Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.

F. J. Och. 2003. Minimum error rate training in statistical machine translation. In *Proc. the 41st ACL*, pages 160–167.

A. Rush and S. Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 498–507.

K. Sagae and A. Lavie. 2006. Parser combination by reparsing. In *Proc. HLT*, pages 129–132.

F. Sangati, W. Zuidema, and R. Bod. 2009. A generative re-ranking model for dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 238–241.

S. M. Shieber, Y. Schabes, and F. C. N. Pereira. 1995. Principles and implementation of deductive parsing. *J. Log. Program.*, 24(1&2):3–36.

C. Sutton, M. Sindelar, and A. McCallum. 2006. Reducing weight undertraining in structured discriminative learning. In *Conference on Human Language Technology and North American Association for Computational Linguistics (HLT-NAACL)*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT'03)*, pages 195–206.

Y. Zhang and S. Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571.

H. Zhang and R. McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331.

Y. Zhang and J. 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.