Transition-Based Parsing for Deep Dependency Structures

Xun Zhang[*]
Peking University

Yantao Du[*]
Peking University

Weiwei Sun[*]
Peking University

Xiaojun Wan[*]
Peking University

Derivations under different grammar formalisms allow extraction of various dependency structures. Particularly, bilexical deep dependency structures beyond surface tree representation can be derived from linguistic analysis grounded by CCG, LFG, *and* HPSG. *Traditionally, these dependency structures are obtained as a by-product of grammar-guided parsers. In this article, we study the alternative data-driven, transition-based approach, which has achieved great success for tree parsing, to build general dependency graphs. We integrate existing tree parsing techniques and present two new transition systems that can generate arbitrary directed graphs in an incremental manner. Statistical parsers that are competitive in both accuracy and efficiency can be built upon these transition systems. Furthermore, the heterogeneous design of transition systems yields diversity of the corresponding parsing models and thus greatly benefits parser ensemble. Concerning the disambiguation problem, we introduce two new techniques, namely, transition combination and tree approximation, to improve parsing quality. Transition combination makes every action performed by a parser significantly change configurations. Therefore, more distinct features can be extracted for statistical disambiguation. With the same goal of extracting informative features, tree approximation induces tree backbones from dependency graphs and re-uses tree parsing techniques to produce tree-related features. We conduct experiments on* CCG*-grounded functor–argument analysis,* LFG*-grounded grammatical relation analysis, and* HPSG*-grounded semantic dependency analysis for English and Chinese. Experiments demonstrate that data-driven models with appropriate transition systems can produce high-quality deep dependency analysis, comparable to more complex grammar-driven*

[*] The authors are with the Institute of Computer Science and Technology, the MOE Key Laboratory of Computational Linguistics, Peking University, Beijing 100871, China.
E-mail: {zhangxunah,duyantao,ws,wanxiaojun}@pku.edu.cn. Weiwei Sun is the corresponding author.

Submission received: 6 May 2015; revised submission received: 2 November 2015; accepted for publication: 18 January 2016.

doi:10.1162/COLI_a_00252

models. Experiments also indicate the effectiveness of the heterogeneous design of transition systems for parser ensemble, transition combination, as well as tree approximation for statistical disambiguation.

1. Introduction

The derivations licensed by a grammar under deep grammar formalisms, for example, combinatory categorial grammar (CCG; Steedman 2000), lexical-functional grammar (LFG; Bresnan and Kaplan 1982) and head-driven phrase structure grammar (HPSG; Pollard and Sag 1994), are able to produce rich linguistic information encoded as bilexical dependencies. Under CCG, this is done by relating the lexical heads of functor categories and their arguments (Clark, Hockenmaier, and Steedman 2002). Under LFG, bilexical grammatical relations can be easily derived as the backbone of F-structures (Sun et al. 2014). Under HPSG, predicate–argument structures (Miyao, Ninomiya, and ichi Tsujii 2004) or reduction of minimal recursion semantics (Ivanova et al. 2012) can be extracted from typed feature structures corresponding to whole sentences. Dependency analysis grounded in deep grammar formalisms is usually beyond tree representations and well-suited for producing meaning representations. Figure 1 is an example from CCGBank. The deep dependency graph conveniently represents more semantically motivated information than the surface tree. For instance, it directly captures the *Agent–Predicate* relations between word "people" and conjuncts "fight," "eat," as well as "drink."

Automatically building deep dependency structures is desirable for many practical NLP applications, for example, information extraction (Miyao et al. 2008) and question answering (Reddy, Lapata, and Steedman 2014). Traditionally, deep dependency graphs are generated as a by-product of grammar-guided parsers. The challenge is that a deep-grammar-guided parsing model usually cannot produce full coverage and the time complexity of the corresponding parsing algorithms is very high. Previous work on data-driven dependency parsing mainly focused on tree-shaped representations. Nevertheless, recent work has shown that a data-driven approach is also applicable to generate more general linguistic graphs. Sagae and Tsujii (2008) present an initial study on applying transition-based methods to generate HPSG-style predicate–argument structures, and have obtained competitive results. Furthermore, Titov et al. (2009) and Henderson et al. (2013) have shown that more general graphs rather than planars can be produced by augmenting existing transition systems.

This work follows early encouraging research and studies transition-based approaches to construct deep dependency graphs. The computational challenge to incremental graph spanning is the existence of a large number of crossing arcs in deep

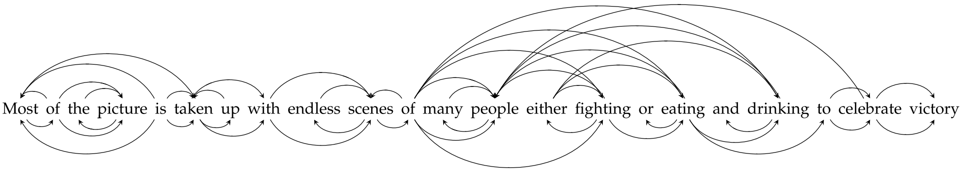

Most of the picture is taken up with endless scenes of many people either fighting or eating and drinking to celebrate victory

Figure 1
An example from CCGBank. The upper curves represent a deep dependency graph and the bottom curves represent a traditional dependency tree.

dependency analysis. To tackle this problem, we integrate insightful ideas, especially the ones illustrated in Nivre (2009) and Gómez-Rodríguez and Nivre (2010), developed in the tree spanning scenario, and design two new transition systems, both of which are able to produce arbitrary directed graphs. In particular, we explore two techniques to localize transition actions to maximize the effect of a greedy search procedure. In this way, the corresponding parsers for generating linguistically motivated bilexical graphs can process sentences in close to linear time with respect to the number of input words. This efficiency advantage allows deep linguistic processing for very-large-scale text data.

For syntactic parsing, ensembled methods have been shown to be very helpful in boosting accuracy (Sagae and Lavie 2006; Zhang et al. 2009; McDonald and Nivre 2011). In particular, Surdeanu and Manning (2010) presented a nice comparative study on various ensemble models for dependency tree parsing. They found that the diversity of base parsers is more important than complex ensemble models for learning. Motivated by this observation, the authors proposed a hybrid transition-based parser that achieved state-of-the-art performance by combining complementary prediction powers of different transition systems. One advantage of their architecture is the linear-time decoding complexity, given that all base models run in linear-time. Another concern of our work is about the model diversity obtained by the heterogeneous design of transition systems for general graph spanning. Empirical evaluation indicates that statistical parsers built upon our new transition systems as well as the existing best transition system—namely, Titov et al. (2009)'s system (THMM, hereafter)—exhibit complementary parsing strengths, which benefit system combination. In order to take advantage of this model diversity, we propose a simple yet effective ensemble model to build a better hybrid system.

We implement statistical parsers using the structured perceptron algorithm (Collins 2002) for transition classification and use a beam decoder for global inference. Concerning the disambiguation problem, we introduce two new techniques, namely, transition combination and tree approximation, to improve parsing quality. To increase system coverage, the ARC transitions designed by the THMM as well as our systems do not change the nodes in the stack nor buffer in a configuration: Only the nodes linked to the top of the stack or buffer are modified. Therefore, features derived from the configurations before and after an ARC transition are not distinct enough to train a good classifier. To deal with this problem, we propose the transition combination technique and three algorithms to derive oracles for modified transition systems. When we apply our models to semantics-oriented deep dependency structures, for example, CCG-grounded functor–argument analysis and HPSG-grounded reduced minimal recursion semantics (MRS; Copestake et al. 2005) analysis, we find that syntactic trees can provide very helpful features. In case the syntactic information is not available, we introduce a tree approximation technique to induce tree backbones from deep dependency graphs. Such tree backbones can be utilized to train a tree parser which provides pseudo tree features.

To evaluate transition-based models for deep dependency parsing, we conduct experiments on CCG-grounded functor–argument analysis (Hockenmaier and Steedman 2007; Tse and Curran 2010), LFG-grounded grammatical relation analysis (Sun et al. 2014), and HPSG-grounded semantic dependency analysis (Miyao, Ninomiya, and ichi Tsujii 2004; Ivanova et al. 2012) for English and Chinese. Empirical evaluation indicates some non-obvious facts:

1. Data-driven models with appropriate transition systems and disambiguation techniques can produce high-quality deep dependency analysis, comparable to more complex grammar-driven models.

2. Parsers built upon heterogeneous transition systems and decoding orders have complementary prediction strengths, and the parsing quality can be significantly improved by system combination; compared to the best individual system, system combination gets an absolute labeled F-score improvement of 1.21 on average.

3. Transition combination significantly improves parsing accuracy on a wide range of conditions, resulting in an absolute labeled F-score improvement of 0.74 on average.

4. Pseudo trees contribute to semantic dependency parsing (SDP) equally well to syntactic trees, and result in an absolute labeled F-score improvement of 1.27 on average.

We compare our parser with representative state-of-the-art parsers (Miyao and Tsujii 2008; Auli and Lopez 2011b; Martins and Almeida 2014; Xu, Clark, and Zhang 2014; Du, Sun, and Wan 2015) with respect to different architectures. To evaluate the impact of grammatical knowledge, we compare our parser with parsers guided by treebank-induced HPSG and CCG grammars. Both of our individual and ensembled parsers achieve equivalent accuracy to HPSG and CCG chart parsers (Miyao and Tsujii 2008; Auli and Lopez 2011b), and outperform a shift-reduce CCG parser (Xu, Clark, and Zhang 2014). It is worth noting that our parsers exclude all syntactic and grammatical information. In other words, strictly less information is used. This result demonstrates the effectiveness of data-driven approaches to the deep linguistic processing problem. Compared to other types of data-driven parsers, our individual parser achieves equivalent performance to and our hybrid parser obtains slightly better results than factorization parsers based on dual decomposition (Martins and Almeida 2014; Du, Sun, and Wan 2015). This result highlights the effectiveness of the lightweight, transition-based approach.

Parsers based on the two new transition systems have been utilized as base components for parser ensemble (Du et al. 2014) for SemEval 2014 Task 8 (Oepen et al. 2014). Our hybrid system obtained the best overall performance of the closed track of this shared task. In this article, we re-implement all models, calibrate features more carefully, and thus obtain improved accuracy. The idea to extract tree-shaped backbone from a deep dependency graph has also been used to design other types of parsing models in our early work (Du et al. 2014, 2015; Du, Sun, and Wan 2015). Nevertheless, the idea to train a pseudo tree parser to serve a transition-based graph parser is new.

The implementation of our parser is available at `http://www.icst.pku.edu.cn/lcwm/grass`.

2. Transition Systems for Graph Spanning

2.1 Background Notations

A dependency graph $G = (V, A)$ is a labeled directed graph, such that for sentence $x = w_1, \ldots, w_n$ the following holds:

1. $V = \{0, 1, 2, \ldots, n\}$,

2. $A \subseteq V \times R \times V$.

The vertex set V consists of $n + 1$ nodes, each of which is represented by a single integer. In particular, 0 represents a virtual root node w_0, and all others correspond to words in x. The arc set A represents the labeled dependency relations of the particular analysis G. Specifically, an arc $(i, r, j) \in A$ represents a dependency relation r from head w_i to dependent w_j. A dependency graph G is thus a set of labeled dependency relations between the root and the words of x. To simplify the description in this section, we mainly consider unlabeled parsing and assume the relation set R is a singleton. Or, taking it another way, we assume $A \subseteq V \times V$. It is straightforward to adapt the discussions in this article for labeled parsing. To do so, we can parameterize transitions with possible dependency relations. For empirical evaluation as discussed in Section 5, we will test both labeled and unlabeled parsing models.

Following Nivre (2008), we define a transition system for dependency parsing as a quadruple $S = (\mathcal{C}, T, c_s, \mathcal{C}_t)$, where

1. $\mathcal{C}$ is a set of configurations, each of which contains a buffer β of (remaining) words and a set A of dependency arcs,
2. T is a set of transitions, each of which is a (partial) function $t : \mathcal{C} \mapsto \mathcal{C}$,
3. c_s is an initialization function, mapping a sentence x to a configuration with $\beta = [1, \ldots, n]$,
4. $\mathcal{C}_t \subseteq \mathcal{C}$ is a set of terminal configurations.

Given a sentence $x = w_1, \ldots, w_n$ and a graph $G = (V, A)$ on it, if there is a sequence of transitions $t_1, \ldots, t_m$ and a sequence of configurations $c_0, \ldots, c_m$ such that $c_0 = c_s(x)$, $t_i(c_{i-1}) = c_i (i = 1, \ldots, m)$, $c_m \in \mathcal{C}_t$, and $A_{c_m} = A$, we say the sequence of transitions is an **oracle** sequence. And we define $\bar{A}_{c_i} = A - A_{c_i}$ for the arcs to be built of c_i. We could denote a transition sequence as either $\mathrm{t}_{1,m}$ or $\mathrm{c}_{0,m}$.

In a typical transition-based parsing process, the input words are put into a queue and partially built structures are organized by a stack. A set of SHIFT/REDUCE actions are performed sequentially to consume words from the queue and update the partial parsing results organized by the stack. Our new systems designed for deep parsing differ with respect to their information structures to define a configuration and the behaviors of transition actions.

2.2 Naive Spanning and Locality

For every two nodes, a simple graph-spanning strategy is to check if they can be directly connected by an arc. Accordingly, a "naive" spanning algorithm can be implemented by exploring a left-to-right checking order, as introduced by Covington (2001) and modified by Nivre (2008).

```
PARSE($x = (w_1, \ldots, w_n)$)
    for $j = 1..n$
        for $k = j - 1..1$
            Link($j, k$)
```

The operation Link chooses between 1) adding the arc (i, j) or (j, i) and 2) adding no arc at all. In this way, the algorithm builds a graph by incrementally trying to link every pair of words.

Left-Arc	$(\sigma\|i,j\|\beta) \Rightarrow (\sigma\|i,j\|\beta)$
Right-Arc	$(\sigma\|i,j\|\beta) \Rightarrow (\sigma\|i,j\|\beta)$
Shift	$(\sigma,j\|\beta) \Rightarrow (\sigma\|j,\beta)$
Pop	$(\sigma\|i,\beta) \Rightarrow (\sigma,\beta)$
Swap	$(\sigma\|i\|j,\beta) \Rightarrow (\sigma\|j,i\|\beta)$
Swap$_{\text{T}}$	$(\sigma\|i\|j,\beta) \Rightarrow (\sigma\|j\|i,\beta)$

Figure 2
Transitions of the online re-ordering approach.

The complexity of naive spanning is $\theta(n^2)$,[1] because it does nothing to explore the topological properties of a linguistic structure. In other words, the naive graph-spanning idea does not fully take advantages of the **greedy** search of the transition-based parsing architecture. On the contrary, a well-designed transition system for (projective) tree parsing can decode in linear time by exploiting locality among subtrees. Take the arc-eager system presented in Nivre (2008), for example: Only the nodes at the top of the stack and the buffer are allowed to be linked. Such limitation is the key to implement a linear time decoder. In the following, we introduce two ideas to localize a transition action, that is, to allow a transition to manipulate only the frontier items in the data structures of a configuration. By this means, we can decrease the number of possible transitions for each configuration and thus minimize the total decoding time.

2.3 System 1: Online Re-ordering

The online re-ordering approach that we explore is to provide the system with ability to re-order the nodes during parsing in an online fashion. The key idea, as introduced in Titov et al. (2009) and Nivre (2009), is to allow a Swap transition that switches the position of the two topmost nodes on the stack. By changing the linear order of words, the system is able to build crossing arcs for graph spanning. We refer to this approach as **online re-ordering**. We introduce a stack-based transition system with online re-ordering for deep dependency parsing. The obtained oracle parser is complete with respect to the class of all directed graphs without self-loop.

2.3.1 The System. We define a transition system $S_S = (\mathcal{C}, T, c_s, \mathcal{C}_t)$, where a configuration $c = (\sigma, \beta, A) \in \mathcal{C}$ contains a stack σ of nodes, besides β and A. We set the initial configuration for a sentence $x = w_1, \ldots, w_n$ to be $c_s(x) = ([], [1, \ldots, n], \{\})$, and take $\mathcal{C}_t$ to be the set of all configurations of the form $c_t = (\sigma, [], A)$ (for any σ any A). These transitions are shown in Figure 2 and explained as follows.

- Shift (sh) removes the front from the buffer and pushes it onto the stack.
- Left/Right-Arc (la/ra) updates a configuration by adding $(j, i)/(i, j)$ to A where i is the top of the stack, and j is the front of the buffer.
- Pop (pop) updates a configuration by popping the top of the stack.
- Swap (sw) updates a configuration with stack $\sigma|i|j$ by moving i back to the buffer.

[1] We assume that at most one edge exists between two words. This is a reasonable assumption for a linguistic representation.

A variation of transition SWAP is SWAP$_\mathrm{T}$, which updates the configuration by swapping i and j. However, the system of this variation is not complete with respect to directed graphs because the power of transition SWAP$_\mathrm{T}$ is limited, and counterexamples of completeness can be found. For more theoretical discussion about this system (i.e., THMM), see Titov et al. (2009). We also denote Titov et al. (2009)'s system as S_T.

2.3.2 Theoretical Analysis. The soundness of S_S is trivial. To demonstrate the completeness of the system, we give a constructive proof that can derive oracle transitions for any arbitrary graph. To simplify the description, the label attached to transitions are not considered. The idea is inspired by Titov et al. (2009). Given a sentence $x = w_1, \ldots, w_n$ and a graph $G = (V, A)$ on it, we start with the initial configuration $c_0 = c_s(x)$ and compute the oracle transitions step by step. On the i-th step, let p be the top of $\sigma_{c_{i-1}}$, b be the front of $\beta_{c_{i-1}}$; let $L(j)$ be the ordered list of nodes connected to j in $\bar{A}_{c_{i-1}}$ for any node $j \in \sigma_{c_{i-1}}$; let $\mathcal{L}(\sigma_{c_{i-1}}) = [L(j_0), \ldots, L(j_l)]$ if $\sigma_{c_{i-1}} = [j_l, \ldots, j_0]$.

The oracle transition for each configuration is derived as follows. If there is no arc linked to p in $\bar{A}_{c_{i-1}}$, then we set t_i to pop; if there exists $a \in \bar{A}_{c_{i-1}}$ linking p and b, then we set t_i to la or ra correspondingly. When there are only sh and sw left, we see if there is any node q under the top of $\sigma_{c_{i-1}}$ such that $L(q)$ precedes $L(p)$ by the lexicographical order. If so, we set t_i to sw; else we set t_i to sh. An example for when to do sw is shown in Figure 3. Let $c_i = t_i(c_{i-1})$; we continue to compute t_{i+1}, until β_{c_i} is empty.

Lemma 1
If t_i is sh, $\mathcal{L}(\sigma_{c_{i-1}}) = [L(j_0), \ldots, L(j_l)]$ is complete ordered by lexicographical order.

Proof
It cannot be the case that for some $u > 0$, $L(j_u)$ strictly precedes $L(j_0)$, otherwise t_i should be sw. It also cannot be the case that for some $u > v > 0$, $L(j_u)$ strictly precedes $L(j_v)$, because when j_{v-1} is shifted onto the stack, $L(j_v)$ precedes $L(j_u)$ and all the transitions do not change $L(j_v)$ and $L(j_u)$ afterwards. ■

Lemma 2
For $i = 0, \ldots, m$, there is no arc $(j, k) \in \bar{A}_{c_i}$ such that $j, k \in \sigma_i$.

Proof
When $j \in \sigma_{c_i}$ is shifted onto the stack by the w-th transition t_w, there must be no arc (j, k) or (k, j) in $\bar{A}_{c_w}$ such that $k \in \sigma_{c_w}$. Otherwise, by induction every node in $\sigma_{c_{w-1}}$ can only link to nodes in $\beta_{c_{w-1}}$, which implies that $L(k)$ has one of the smallest lexicographical orders, and from Lemma 1 the top of $\sigma_{c_{w-1}}$ must be linked to j. And not sh, but la or ra should be applied. ■

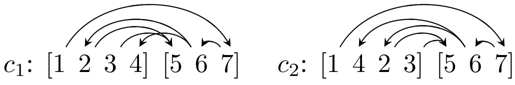


Figure 3
σ, β, and $\bar{A}$ of two configurations c_1 and c_2. In the left graphic, $\mathcal{L}(\sigma_{c_1}) = [[6], [5], [5, 6], [7]]$. Because [5, 6] and [5] precedes [6], we apply two SWAPs and then two SHIFTs, obtaining the right graphic.

Theorem 1
$t_1, \ldots, t_m$ is an oracle sequence of transitions for G.

Proof
From Lemma 2, we can infer that $\bar{A}_{c_m} = \emptyset$, so it suffices to show the sequence of transitions is always finite. We define a **swap sequence** to be a subsequence $t_i, \ldots, t_j$ such that t_i and t_j are sw, t_{i-1} and t_{j+1} are not sw, and a **shift sequence** similarly. It can be seen that a swap sequence is always followed by a shift sequence, the length of which is no less than the swap sequence, and if the two sequences are of the same length, the next transition cannot be sw. Let $\#(t)$ to be the number of transition types t in the sequence, then #(la), #(ra), #(pop), and $\#(\mathsf{sh}) - \#(\mathsf{sw})$ are all finite. Therefore the number of swap sequence is finite, indicating that the transition sequence is finite. ■

2.4 System 2: Two-Stack–Based System

A majority of transition systems organize partial parsing results with a stack. Classical parsers, including arc-standard and arc-eager ones, add dependency arcs only between nodes that are adjacent on the stack or the buffer. A natural idea to produce crossing arcs is to temporarily move nodes that block non-adjacent nodes to an extra memory module, like the two-stack–based system for two-planar graphs (Gómez-Rodríguez and Nivre 2010) and the list-based system (Nivre 2008). In this article, we design a new transition system to handle crossing arcs by using two stacks. This system is also complete with respect to the class of directed graphs without self-loop.

2.4.1 The System. We define the two-stack–based transition system $S_{2S} = (\mathcal{C}, T, c_s, \mathcal{C}_t)$, where a configuration $c = (\sigma, \sigma', \beta, A) \in \mathcal{C}$ contains a primary stack σ and a secondary stack σ'. We set $c_s(x) = ([], [], [1, \ldots, n], \{\})$ for the sentence $x = w_1, \ldots, w_n$, and we take the set $\mathcal{C}_t$ to be the set of all configurations with empty buffers. The transition set T contains six types of transitions, as shown in Figure 4. We only explain Mem and Recall:

- Mem (mem) pops the top element from the primary stack and pushes it onto the secondary stack.
- Recall (rc) moves the top element of the secondary stack back to the primary stack.

Left-Arc	$(\sigma\|i, \sigma', j\|\beta) \Rightarrow (\sigma\|i, \sigma', j\|\beta)$
Right-Arc	$(\sigma\|i, \sigma', j\|\beta) \Rightarrow (\sigma\|i, \sigma', j\|\beta)$
Shift	$(\sigma, \sigma', j\|\beta) \Rightarrow (\sigma\|j, \sigma', \beta)$
Pop	$(\sigma\|i, \sigma', \beta) \Rightarrow (\sigma, \sigma', \beta)$
Mem	$(\sigma\|i, \sigma', \beta) \Rightarrow (\sigma, \sigma'\|i, \beta)$
Recall	$(\sigma, \sigma'\|i, \beta) \Rightarrow (\sigma\|i, \sigma', \beta)$

Figure 4
Transitions of the two-stack–based system.

2.4.2 Theoretical Analysis. The soundness of this system is trivial, and the completeness is also straightforward after we give the construction of an oracle transition sequence for an arbitrary graph. The oracle is computed as follows on the i-th step: We do la, ra, and pop transitions just like in Section 2.3.2. After that, let b be the front of $\beta_{c_{i-1}}$, we see if there is $j \in \sigma_{c_{i-1}}$ or $j \in \sigma'_{c_{i-1}}$ linked to b by an arc in $\bar{A}_{c_{i-1}}$. If $j \in \sigma_{c_{i-1}}$, then we do a sequence of mem to make j the top of $\sigma_{c_{i-1}}$; if $j \in \sigma'_{c_{i-1}}$, then we do a sequence of rc to make j the top of $\sigma_{c_{i-1}}$. When no node in $\sigma_{c_{i-1}}$ or $\sigma'_{c_{i-1}}$ is linked to b, we do sh.

Theorem 2
S_{2S} is complete with respect to directed graphs without self-loop.

Proof
The completeness immediately follows the fact that the computed oracle sequence is finite, and every time a node is shifted onto σ_{c_i}, no arc in $\bar{A}_{c_i}$ links nodes in σ_{c_i} to the shifted node. ■

2.4.3 Related Systems. Gómez-Rodríguez and Nivre (2010, 2013) introduced a two-stack–based transition system for tree parsing. Their study is motivated by the observation that the majority of dependency trees in various treebanks are actually planar or two-planar graphs. Accordingly, their algorithm is specially designed to handle projective trees and two-planar trees, but not all graphs. Because many more crossing arcs exist in deep dependency structures and more sentences are assigned with neither planar nor two-planar graphs, their strategy of utilizing two stacks is not suitable for the deep dependency parsing problem. Different from their system, our new system maximizes the utility of two memory modules and is able to handle any directed graphs.

The list-based systems, such as the basic one introduced by Nivre (2008) and the extended one introduced by Choi and Palmer (2011), also use two memory modules. The function of the secondary memory module of their systems and ours is very different. In our design, only nodes involved in a subgraph that contains crossing arcs may be put into the second stack. In the existing list-based systems, both lists are heavily used, and nodes may be transferred between them many times. The function of the two lists is to simulate one memory module that allows accessing any unit in it.

2.5 Extension

2.5.1 Graphs with Loops. It is easy to extend our system to generate arbitrary directed graphs by adding a new transition:

- Self-Arc adds an arc from the top element of the primary memory module (σ) to itself, but does not update any stack nor buffer.

Theorem 3
S_S and S_{2S} augmented with Self-Arc are complete with respect to directed graphs.

2.5.2 Labeled Parsing and Supertagging. It is also straightforward to adapt the two transition systems for labeled dependency graph generation. To do so, we can parameterize Left-Arc and Right-Arc transitions with dependency relations. For example, a parameterized transition Left-Arc$_r$ tells the system not only that there is an arc between the frontier node of the stack and the frontier node of the buffer but also that this arc holds a relation r. Some linguistic representations assign labels to nodes as well. When a deep grammar is considered to license to representation, node labels are usually called "supertags." To assign supertags to words, namely, nodes in a dependency graph, we can parameterize the SHIFT transition with tag labels.

3. Statistical Disambiguation

3.1 Transition Classification

A transition-based parser must decide which transition is appropriate given its parsing environment (i.e., configuration). As with many other data-driven dependency parsers, we use a global linear model for disambiguation. In other words, a discriminative classifier is utilized to approximate the oracle function for a transition system S that maps a configuration c to a transition t that is defined on c. More formally, a transition-based statistical parser tries to find the transition sequence $\mathrm{c}_{0,m}$ that maximizes the following score

$$\textsc{Score}(\mathrm{c}_{0,m}) = \sum_{i=0}^{m-1} \textsc{Score}(c_i, t_{i+1}) \quad (1)$$

Following the state-of-the-art discriminative disambiguation technique for data-driven parsing, we define the score function as a linear combination of features defined over a configuration and a transition, as follows:

$$\textsc{Score}(c_i, t_{i+1}) = \theta^\top \phi(c_i, t_{i+1}) \quad (2)$$

where ϕ defines a vector for each configuration–transition pair and θ is the weight vector for linear combination.

Exact calculation of the maximization is extremely hard without any assumption of ϕ. Even with a proper ϕ for real-word parsing, exact decoding is still impractical for most practical feature designs. In this article, we follow the recent success of using beam search for approximate decoding. During parsing, the parser keeps track of multiple yet a fixed number of partial outputs to avoid making decisions too early. Training a parser in the discriminative setting corresponds to estimating θ associated with rich features. Previous research on dependency parsing shows that structured perceptron (Collins 2002; Collins and Roark 2004) is one of the strongest learning algorithms. In all experiments, we use the averaged perceptron algorithm with early update to estimate parameters. The whole parser is very similar to the transition-based system introduced in Zhang and Clark (2008, 2011b).

3.2 Transition Combination

In either THMM, S_S, or S_{2S}, the LEFT/RIGHT-ARC transition does not modify either the stack or the buffer. Only new edges are added to the target graph. When automatic classifiers are utilized to approximate an oracle, a majority of features for predicting an ARC transition will be overlapped with the features for the successive transition. Empirically, this property significantly decreases the parsing accuracy. A key observation of a linguistically motivated bilexical graph is that there is usually at most one edge between any two words, therefore an ARC transition is not followed by another ARC. As a result, any ARC with its successive transition modifies a configuration much. To practically

LEFT-ARC	$(\sigma\|i,\sigma',j\|\beta) \Rightarrow (\sigma\|i,\sigma',j\|\beta)$
RIGHT-ARC	$(\sigma\|i,\sigma',j\|\beta) \Rightarrow (\sigma\|i,\sigma',j\|\beta)$
SHIFT	$(\sigma,\sigma',j\|\beta) \Rightarrow (\sigma\|j,\sigma',\beta)$
POP	$(\sigma\|i,\sigma',\beta) \Rightarrow (\sigma,\sigma',\beta)$
MEM	$(\sigma\|i,\sigma',\beta) \Rightarrow (\sigma,\sigma'\|i,\beta)$
RECALL	$(\sigma,\sigma'\|i,\beta) \Rightarrow (\sigma\|i,\sigma',\beta)$
LEFT-ARC-SHIFT	$(\sigma\|i,\sigma',j\|\beta) \Rightarrow (\sigma\|i\|j,\sigma',\beta)$
LEFT-ARC-POP	$(\sigma\|i,\sigma',j\|\beta) \Rightarrow (\sigma,\sigma',j\|\beta)$
LEFT-ARC-MEM	$(\sigma\|i,\sigma',j\|\beta) \Rightarrow (\sigma,\sigma'\|i,j\|\beta)$
LEFT-ARC-RECALL	$(\sigma\|i',\sigma'\|i,j\|\beta) \Rightarrow (\sigma\|i'\|i,\sigma',j\|\beta)$
RIGHT-ARC-SHIFT	$(\sigma\|i,\sigma',j\|\beta) \Rightarrow (\sigma\|i\|j,\sigma',\beta)$
RIGHT-ARC-POP	$(\sigma\|i,\sigma',j\|\beta) \Rightarrow (\sigma,\sigma',j\|\beta)$
RIGHT-ARC-MEM	$(\sigma\|i,\sigma',j\|\beta) \Rightarrow (\sigma,\sigma'\|i,j\|\beta)$
RIGHT-ARC-RECALL	$(\sigma\|i',\sigma'\|i,j\|\beta) \Rightarrow (\sigma\|i'\|i,\sigma',j\|\beta)$

Figure 5
Original and combined transitions for the two-stack combined system. Two-cycle is not considered here.

improve the performance of a statistical parser, we combine every pair of two successive transitions starting with ARC and transform the proposed two transition systems into two modified ones. For example, in our two-stack–based system, after combining, we obtain the transitions presented in Figure 5.

The number of edges between any two words could be at most two in real data. If there are two edges between two words w_a and w_b, it must be $w_a \rightarrow w_b$ and $w_b \rightarrow w_a$. We call these two edges a two-cycle, and call this problem the two-cycle problem. In our combined transitions, a LEFT/RIGHT-ARC transition should appear before a non-ARC transition. In order to generate two edges between two words, we have two strategies:

A) Add a new type of transitions to each system, which consist of a LEFT-ARC transition, a RIGHT-ARC transition, and any other non-ARC transition (e.g., LEFT-ARC-RIGHT-ARC-RECALL for S_{2S}).

B) Use a non-directional ARC transition instead of LEFT/RIGHT-ARC. Here, an ARC transition may add one or two edges depends on its label. In detail, we propose two algorithms, namely, ENCODELABEL and DECODELABEL (see Algorithms 1 and 2), to deal with labels for ARC transition.

Algorithm 1 encode label

```
procedure ENCODELABEL(type, lLabel, rLabel)
    if type == LEFT then
        return "left" + lLabel
    else if type == RIGHT then
        return "right" + rLabel
    else
        return "both" + lLabel + "|" + rLabel
    end if
end procedure
```

Algorithm 2 decode combined label, return a pair of left label and right label

```
procedure DecodeLabel($label$)
    if $label.startswith?(''left'')$ then
        return $\{label[4:], nil\}$
    else if $label.startswith?(''right'')$ then
        return $\{nil, label[5:]\}$
    else
        return $\{label[4 : label.index('|')], label[(label.index('|') + 1) :]\}$
    end if
end procedure
```

To our best efforts, the strategy B performs better.

First, let us consider accuracy. Generally speaking, it is harder for transition classification if more target transitions are defined. Using strategy A, we should add additional transitions to handle the two-cycle condition. Based on our experiments, the performance decreases when using more transitions.

Considering efficiency, we can save time by only using labels that appear in training data in strategy B. If we have a total of K possible labels in training data, they will generate K^2 two-cycle types, but only k possible combinations of two-cycle appear in training data ($k \ll K^2$). In strategy A, we must add K^2 transitions to deal with all possible two-cycle types, but most of them do not make sense. Using fewer two-cycle types helps us eliminate the invalid calculation and save time effectively.

Using strategy B, we change the original edges′ labels and use the ARC(label)–non-ARC transition instead of LEFT/RIGHT-ARC(label)–non-ARC. An ARC(label)–non-ARC transition should execute the ARC(label) transition first, then execute the non-ARC transition. ARC(label) generates one or two edges depends on its label. Not only do we encode two-cycle labels, but also LEFT/RIGHT-ARC labels. In practice, we only use those labels that appear in training data. Because labels that do not appear only contribute non-negative weights while training, we can eliminate them without any performance loss.

For each transition system and each dependency graph, we generate an oracle transition, and train our model according to this oracle. The constructive proofs presented in Section 2.3 and 2.4 define two kinds of oracles. However, they are not directly applicable when the transition combination strategy is utilized. The main challenge is the existence of cycles. In this article, we propose three algorithms to derive oracles for THMM, S_S, and S_{2S}, respectively. Algorithms 3 to 5 illustrate the key steps of the procedure of our system, which find the next transition t given a configuration c and gold graph $G_{gold} = (V_x, A_{gold})$ for the three systems. When this key procedure, namely, the EXTRACTONEORACLE method, is well defined, the entire transition system can be derived as follows:

```
ExtractOracle($c_0, A_{gold}$)
oracle = $\emptyset$
while $t \leftarrow$ ExtractOneOracle($c_0, A_{gold}, nil$) do
    $oracle.push\_back = t$
    $c_0 \leftarrow t(c_0)$
end while
```

Algorithm 3 Oracle generation for the THMM system

```
procedure ExtractOneOracle($c, A_{gold}, label$)
    if $c = (\sigma|i,j|\beta,A) \wedge \neg\exists k[k \geqslant j \wedge \exists l[(i,l,k) \in A_{gold}]]$ then
        if $label = nil$ then
            return Reduce
        else
            return Arc($label$) $\circ$ Reduce
        end if
    else if $c = (\sigma|i,j|\beta,A) \wedge \exists l[(i,l,j) \in A_{gold}]$ then
        $A_{gold} \leftarrow A_{gold} - (i,l,j)$
        return ExtractOneOracle($c, A_{gold}, label$)
    else if $c = (\sigma|i_1|i_0,j|\beta,A) \wedge \exists k_0 k_1[k_0 \geqslant j \wedge k_1 \geqslant j \wedge \exists l_0[(i_0,l_0,k_0) \in A_{gold}] \wedge \exists l_1[(i_1,l_1,k_1) \in A_{gold}] \wedge \neg\exists k_0\prime[k_0\prime < k_0 \wedge \exists l_0\prime[(i_0,l_0\prime,k_0\prime) \in A_{gold}]] \wedge \exists l_1\prime[(i_1,l_1\prime,k_1\prime) \in A_{gold}]] \wedge k_0 < k_1] \vee \neg\exists k_1[k_1 \geqslant j \wedge \exists l_1[(i_1,l_1,k_1) \in A_{gold}]]$ then
        if $label = nil$ then
            return Swap
        else
            return Arc($label$) $\circ$ Swap
        end if
    end if
    if $c = (\sigma,j|\beta,A)$ then
        if $label = nil$ then
            return Shift
        else
            return Arc($label$) $\circ$ Shift
        end if
    end if
    return $nil$
end procedure
```

We want to emphasize that, although the EXTRACTORACLE methods initialize the parameter LABEL in EXTRACTONEORACLE as nil, if an arc transition is predicted in the EXTRACTONEORACLE method, it will call EXTRACTONEORACLE recursively to return an ARC(label)–non-ARC transition and assign a value for that LABEL.

3.3 Feature Design

Developing features has been shown to be crucial to advancing the state-of-the-art in dependency parsing (Koo and Collins 2010; Zhang and Nivre 2011). To build accurate deep dependency parsers, we utilize a large set of features for transition classification.

To conveniently define all features, we use the following notation. In a configuration with stack σ and buffer β, we denote the top two nodes in σ by σ_0 and σ_1, and the front of β by β_0. In a configuration of the two-stack–based system with the second stack σ', the top element of σ' is denoted by σ'_0 and the front of β by β_0. The left-most dependent of node n is denoted by $n.lc$, the right-most one by $n.rc$. The left-most parent of node n is denoted by $n.lp$, the right-most one by $n.rp$. Then we denote the word and POS-tag

Algorithm 4 Oracle generation for the online re-ordering system

```
procedure EXTRACTONEORACLE($c, A_{gold}, label$)
    if $c = (\sigma|i, j|\beta, A) \wedge \neg\exists k[k \geqslant j \wedge \exists l[(i, l, k) \in A_{gold}]]$ then
        if $label = nil$ then
            return REDUCE
        else
            return ARC($label$) $\circ$ REDUCE
        end if
    else if $c = (\sigma|i, j|\beta, A) \wedge \exists l[(i, l, j) \in A_{gold}]$ then
        $A_{gold} \leftarrow A_{gold} - (i, l, j)$
        return EXTRACTONEORACLE($c, A_{gold}, label$)
    else if $c = (\sigma|i, j|\beta, A) \wedge \exists i\prime[i\prime < i \wedge i\prime \in \sigma \wedge \exists l\prime[(i\prime, l\prime, j) \in A_{gold}]]$ then
        if $label = nil$ then
            return SWAP
        else
            return ARC($label$) $\circ$ SWAP
        end if
    end if
    if $c = (\sigma, j|\beta, A)$ then
        if $label = nil$ then
            return SHIFT
        else
            return ARC($label$) $\circ$ SHIFT
        end if
    end if
    return $nil$
end procedure
```

of node n by w_n, p_n, respectively. Our parser derives the so-called path features from dependency trees. The path features collect POS tags or the first letter of POS tags along the tree between two nodes. Given two nodes n_1 and n_2, we denote the path feature as $path(n_1, n_2)$ and the coarse-grained path feature as $cpath(n_1, n_2)$. The syntactic head of a node n is denoted as $n.h$.

We use the same feature templates for the online re-ordering and the two-stack–based systems, and they are slightly different from THMM. Figure 6 defines basic feature template functions. All feature templates are described here.

- THMM system: $\mathrm{f}_{\mathrm{uni}}(\sigma_0)$, $\mathrm{f}_{\mathrm{uni}}(\sigma_1)$, $\mathrm{g}_{\mathrm{uni}}(\beta_0)$, $\mathrm{f}_{\mathrm{context}}(\sigma_0)$, $\mathrm{f}_{\mathrm{context}}(\beta_0)$, $\mathrm{f}_{\mathrm{pair-1}}(\sigma_0, \beta_0)$, $\mathrm{f}_{\mathrm{pair-1}}(\sigma_1, \beta_0)$, $\mathrm{f}_{\mathrm{pair}}(\sigma_0, \sigma_1)$, $\mathrm{f}_{\mathrm{tri}}(\sigma_0, \beta_0, \sigma_1)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_0, \beta_0, \sigma_0.lp)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_0, \beta_0, \sigma_0.rp)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_0, \beta_0, \sigma_0.lc)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_0, \beta_0, \sigma_0.lc)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_0, \beta_0, \beta_0.lp)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_0, \beta_0, \beta_0.lc)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_1, \beta_0, \sigma_1.lp)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_1, \beta_0, \sigma_1.rp)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_1, \beta_0, \sigma_1.lc)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_1, \beta_0, \sigma_1.lc)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_1, \beta_0, \beta_0.lp)$, $\mathrm{f}_{\mathrm{tri-1}}(\sigma_1, \beta_0, \beta_0.lc)$, $\mathrm{f}_{\mathrm{quar-1}}(\sigma_0, \beta_0, \sigma_0.rp, \sigma_0.rc)$, $\mathrm{f}_{\mathrm{quar-1}}(\sigma_0, \beta_0, \sigma_0.lc, \sigma_0.lc2)$, $\mathrm{f}_{\mathrm{quar-1}}(\sigma_0, \beta_0, \sigma_0.rc, \sigma_0.rc2)$, $\mathrm{f}_{\mathrm{quar-1}}(\sigma_0, \beta_0, \beta_0.lp, \beta_0.lc)$, $\mathrm{f}_{\mathrm{quar-1}}(\sigma_0, \beta_0, \beta_0.lc, \beta_0.lc2)$, $\mathrm{f}_{\mathrm{quar-1}}(\sigma_1, \beta_0, \sigma_1.rp, \sigma_1.rc)$, $\mathrm{f}_{\mathrm{quar-1}}(\sigma_1, \beta_0, \sigma_1.lc, \sigma_1.lc2)$, $\mathrm{f}_{\mathrm{quar-1}}(\sigma_1, \beta_0, \sigma_1.rc, \sigma_1.rc2)$,

Algorithm 5 Oracle generation for the two-stack–based system

```
procedure ExtractOneOracle($c, A_{gold}, label$)
    if $c = (\sigma|i, \sigma_s, j|\beta, \mathbf{A}) \wedge \neg\exists k[k \geqslant j \wedge \exists l[(i, l, k) \in A_{gold}]]$ then
        if $label = nil$ then
            return Reduce
        else
            return Arc($label$) $\circ$ Reduce
        end if
    else if $c = (\sigma|i, \sigma_s, j|\beta, A) \wedge \exists l[(i, l, j) \in A_{gold}]$ then
        $A_{gold} \leftarrow A_{gold} - (i, l, j)$
        return ExtractOneOracle($c, \sigma_s, A_{gold}, label$)
    else if $c = (\sigma|i, \sigma_s, j|\beta, A) \wedge \exists i\prime[i\prime < i \wedge i\prime \in \sigma \wedge \exists l\prime[(i\prime, l\prime, j) \in A_{gold}]]$ then
        if $label = nil$ then
            return Mem
        else
            return Arc($label$) $\circ$ Mem
        end if
    else if $c = (\sigma|i, \sigma_s|i_s, j|\beta, A)$ then
        if $label = nil$ then
            return Recall
        else
            return Arc($label$) $\circ$ Recall
        end if
    end if
    if $c = (\sigma, \sigma_s, j|\beta, A)$ then
        if $label = nil$ then
            return Shift
        else
            return Arc($label$) $\circ$ Shift
        end if
    end if
    return $nil$
end procedure
```

$f_{quar-1}(\sigma_1, \beta_0, \beta_0.lp, \beta_0.lc)$, $f_{quar-1}(\sigma_1, \beta_0, \beta_0.lc, \beta_0.lc2)$, $f_{path}(\sigma_0, \beta_0)$, $f_{path}(\sigma_1, \beta_0)$, $f_{char}(\sigma_0)$, $f_{char}(\beta_0)$,

- Online re-ordering/two stack system: $f_{uni}(\sigma_0)$, $f_{uni}(\sigma_1)$, $f_{uni}(\sigma_0\prime)$, $g_{uni}(\beta_0)$, $f_{context}(\sigma_0)$, $f_{context}(\beta_0)$, $f_{pair-1}(\sigma_0, \beta_0)$, $f_{pair-1}(\sigma_1, \beta_0)$, $f_{pair-1}(\sigma_0\prime, \beta_0)$, $f_{pair}(\sigma_0, \sigma_1)$, $f_{pair}(\sigma_0, \sigma_0\prime)$, $f_{tri}(\sigma_0, \beta_0, \sigma_1)$, $f_{tri}(\sigma_0, \beta_0, \sigma_0\prime)$, $f_{tri-1}(\sigma_0, \beta_0, \sigma_0.lp)$, $f_{tri-1}(\sigma_0, \beta_0, \sigma_0.rp)$, $f_{tri-1}(\sigma_0, \beta_0, \sigma_0.lc)$, $f_{tri-1}(\sigma_0, \beta_0, \sigma_0.lc)$, $f_{tri-1}(\sigma_0, \beta_0, \beta_0.lp)$, $f_{tri-1}(\sigma_0, \beta_0, \beta_0.lc)$, $f_{tri-1}(\sigma_1, \beta_0, \sigma_1.lp)$, $f_{tri-1}(\sigma_1, \beta_0, \sigma_1.rp)$, $f_{tri-1}(\sigma_1, \beta_0, \sigma_1.lc)$, $f_{tri-1}(\sigma_1, \beta_0, \sigma_1.lc)$, $f_{tri-1}(\sigma_1, \beta_0, \beta_0.lp)$, $f_{tri-1}(\sigma_1, \beta_0, \beta_0.lc)$, $f_{tri-1}(\sigma_0\prime, \beta_0, \sigma_0\prime.lp)$, $f_{tri-1}(\sigma_0\prime, \beta_0, \sigma_0\prime.rp)$, $f_{tri-1}(\sigma_0\prime, \beta_0, \sigma_0\prime.lc)$, $f_{tri-1}(\sigma_0\prime, \beta_0, \sigma_0\prime.lc)$, $f_{tri-1}(\sigma_0\prime, \beta_0, \beta_0.lp)$, $f_{tri-1}(\sigma_0\prime, \beta_0, \beta_0.lc)$, $f_{quar-1}(\sigma_0, \beta_0, \sigma_0.rp, \sigma_0.rc)$, $f_{quar-1}(\sigma_0, \beta_0, \sigma_0.lc, \sigma_0.lc2)$, $f_{quar-1}(\sigma_0, \beta_0, \sigma_0.rc, \sigma_0.rc2)$,

$\mathrm{f_{uni}}(X)$: $X.w,\ X.p,\ X.w \circ X.lp.l,\ X.w \circ X.rp.l,\ X.w \circ X.lc.l,\ X.w \circ X.rc.l,\ X.w \circ X.lp.a,\ X.w \circ X.rp.a,\ X.w \circ X.lc.a,\ X.w \circ X.rc.a,\ X.w \circ X.p.a,\ X.w \circ X.c.a,\ X.w \circ X.lc.set,\ X.p \circ X.lc.set,\ X.w \circ X.rc.set,\ X.p \circ X.rc.set$
$\mathrm{g_{uni}}(X)$: $X.w, X.p, X.w \circ X.lp.l, X.p \circ X.lp.l, X.w \circ X.lc.l, X.p \circ X.lc.l, X.w \circ X.lp.a, X.p \circ X.lp.a, X.w \circ X.lc.a, X.p \circ X.lc.a\ X.w \circ X.lc.set, X.p \circ X.lc.set$
$\mathrm{f_{context}}(X)$: $X_{-2}.w,\ X_{-1}.w,\ X_{+1}.w,\ X_{+2}.w,\ X_{-2}.p,\ X_{-1}.p,\ X_{+1}.p,\ X_{+2}.p,\ X_{-2}.w \circ X_{-1}.w,\ X_{-1}.w \circ X_{+1}.w, X_{+1}.w \circ X_{+2}.w, X_{-2}.p \circ X_{-1}.p, X_{-1}.p \circ X_{+1}.p, X_{+1}.p \circ X_{+2}.p$
$\mathrm{f_{pair}}(X,Y)$: $X.wp \circ Y.wp, X.wpY.w, X.wp \circ Y.p, X.w \circ Y.wp, X.p \circ Y.wp, X.w \circ Y.w, X.w \circ Y.p, X.p \circ Y.w, X.p \circ Y.p$
$\mathrm{f_{pair-l}}(X,Y)$: $X.wp \circ Y.wp,\ X.wpY.w,\ X.wp \circ Y.p,\ X.w \circ Y.wp,\ X.p \circ Y.wp,\ X.w \circ Y.w,\ X.w \circ Y.p,\ X.p \circ Y.w,\ X.p \circ Y.p,\ X.w \circ Y.w \circ X.rc.a,\ X.w \circ Y.w \circ Y.lc.a,\ X.w \circ Y.w \circ \langle X,Y \rangle.d,\ X.p \circ Y.p \circ \langle X,Y \rangle.d,\ X.w \circ Y.p \circ \langle X,Y \rangle.d,\ X.p \circ Y.w \circ \langle X,Y \rangle.d,\ X.p \circ Y.p \circ X.lc.set,\ X.p \circ Y.p \circ X.rc.set,\ X.p \circ Y.p \circ Y.lc.set$
$\mathrm{f_{tri}}(X,Y,Z)$: $X.w \circ Y.p \circ Z.p, X.p \circ Y.w \circ Z.p, X.p \circ Y.p \circ Z.w, X.p \circ Y.p \circ Z.p$
$\mathrm{f_{tri-l}}(X,Y,Z)$: $X.w \circ Y.p \circ Z.p \circ \langle X,Z \rangle.l,\ X.p \circ Y.w \circ Z.p \circ \langle X,Z \rangle.l,\ X.p \circ Y.p \circ Z.w \circ \langle X,Z \rangle.l,\ X.p \circ Y.p \circ Z.p \circ \langle X,Z \rangle.l$
$\mathrm{f_{quar-l}}(X,Y,Z,W)$: $X.p \circ Y.p \circ Z.p \circ W.p \circ \langle X,Z \rangle.l \circ \langle X,W \rangle.l$
$\mathrm{f_{path}}(X,Y)$: $\langle X,Y \rangle.path,\ \langle X,Y \rangle.cpath,\ X.p \circ Y.p \circ X.tp.w,\ X.p \circ Y.w \circ X.tp.p,\ X.w \circ Y.p \circ X.tp.p,\ X.p \circ Y.p \circ Y.tp.w, X.p \circ Y.w \circ Y.tp.p, X.w \circ Y.p \circ Y.tp.p$
$\mathrm{f_{char}}(X)$: $X_{[-1,-1]}.w, X_{[-2,-1]}.w, X_{[-3,-1]}.w, X_{[+1,+1]}.w, X_{[+1,+2]}.w, X_{[+1,+3]}.w$

Figure 6
Feature template functions.

$$\begin{aligned}
&\mathrm{f_{quar-l}}(\sigma_0, \beta_0, \beta_0.lp, \beta_0.lc), \mathrm{f_{quar-l}}(\sigma_0, \beta_0, \beta_0.lc, \beta_0.lc2),\\
&\mathrm{f_{quar-l}}(\sigma_1, \beta_0, \sigma_1.rp, \sigma_1.rc), \mathrm{f_{quar-l}}(\sigma_1, \beta_0, \sigma_1.lc, \sigma_1.lc2),\\
&\mathrm{f_{quar-l}}(\sigma_1, \beta_0, \sigma_1.rc, \sigma_1.rc2), \mathrm{f_{quar-l}}(\sigma_1, \beta_0, \beta_0.lp, \beta_0.lc),\\
&\mathrm{f_{quar-l}}(\sigma_1, \beta_0, \beta_0.lc, \beta_0.lc2), \mathrm{f_{quar-l}}(\sigma_0\prime, \beta_0, \sigma_0\prime.rp, \sigma_0\prime.rc),\\
&\mathrm{f_{quar-l}}(\sigma_0\prime, \beta_0, \sigma_0\prime.lc, \sigma_0\prime.lc2), \mathrm{f_{quar-l}}(\sigma_0\prime, \beta_0, \sigma_0\prime.rc, \sigma_0\prime.rc2),\\
&\mathrm{f_{quar-l}}(\sigma_0\prime, \beta_0, \beta_0.lp, \beta_0.lc), \mathrm{f_{quar-l}}(\sigma_0\prime, \beta_0, \beta_0.lc, \beta_0.lc2), \mathrm{f_{path}}(\sigma_0, \beta_0),\\
&\mathrm{f_{path}}(\sigma_1, \beta_0), \mathrm{f_{path}}(\sigma_0\prime, \beta_0), \mathrm{f_{char}}(\sigma_0), \mathrm{f_{char}}(\beta_0)
\end{aligned}$$

4. Tree Approximation

Tree structures exhibit many computationally good properties, and parsing techniques for tree-structured representations are quite mature to some extent. When we consider semantics-oriented graphs, such as the representations for semantic role labeling

(SRL; Surdeanu et al. 2008; Hajič et al. 2009), CCG-grounded functor–argument (Clark, Hockenmaier, and Steedman 2002) analysis, HPSG-grounded predicate–argument analysis (Miyao, Ninomiya, and ichi Tsujii 2004), and reduction of MRS (Ivanova et al. 2012), syntactic trees can provide very useful features for semantic disambiguation (Punyakanok, Roth, and Yih 2008). Our parser also utilizes a *path* feature template (as defined in Section 3.3) to incorporate syntactic information for disambiguation.

In case syntactic tree information is not available, we introduce a tree approximation technique to induce tree backbones from deep dependency graphs. Such tree backbones can be utilized to train a tree parser which provides *pseudo* path features. In particular, we introduce an algorithm to associate every graph with a projective dependency tree, which we call **weighted conversion**. The tree reflects partial information about the corresponding graph. The key idea underlying this algorithm is to assign heuristic weights to all ordered pairs of words, and then find the tree with maximum weights. That means a tree frame of a given graph is automatically derived as an alternative for syntactic analysis.

We assign weights to all the possible edges (i.e., all pairs of words) and then determine which edges are to be kept by finding the maximum spanning tree. More formally, given a set of nodes V, each possible edge (i,j), where $i,j \in V$, is assigned a heuristic weight $\omega(i,j)$. Among all trees (denoted as $\mathcal{T}$) over V, the maximum spanning tree $T^{\max}$ contains the maximum sum of values of edges:

$$T^{\max} = \arg \max_{(V,A_T)\in\mathcal{T}} \sum_{(i,j)\in A_T} t(i,j)\omega(i,j) \tag{3}$$

We separate the $\omega(i,j)$ into three parts ($\omega(i,j) = A(i,j) + B(i,j) + C(i,j)$) that are as defined here.

- $A(i,j) = a \cdot \max\{y(i,j), y(j,i)\}$: a is the weight for the existing edge on graph ignoring direction.
- $B(i,j) = b \cdot y(i,j)$: b is the weight for the forward edge on the graph.
- $C(i,j) = n - |i-j|$: This term estimates the importance of an edge where n is the length of the given sentence. For dependency parsing, we consider edges with short distance to be more important because those edges can be predicted more accurately in future parsing processes.
- $a \gg b \gg n$ or $a > bn > n^2$: The converted tree should contain as many arcs as possible in original graph, and the direction of the arcs should not be changed if possible. The relationship of a, b, and c guarantees this.

After all edges are weighted, we can use maximum spanning tree algorithms to obtain the converted tree. To obtain the projective tree, we choose Eisner's algorithm. For any graph, we can call this algorithm and get a corresponding tree. However, the tree is informative only when the given graph is dense enough. Fortunately, this condition holds for semantic dependency parsing.

5. Empirical Evaluation

5.1 Set-up

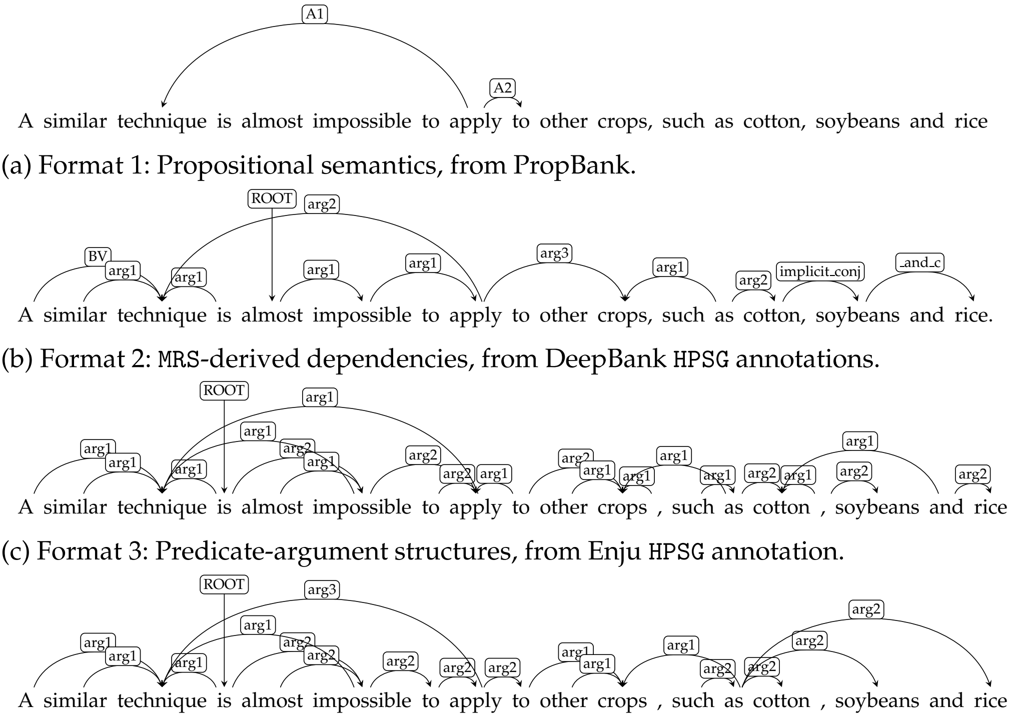
We present empirical evaluation of different incremental graph spanning algorithms for CCG-style functor–argument analysis, LFG-style grammatical relation analysis, and HPSG-style semantic dependency analysis for English and Chinese. Linguistically speaking, these types of syntacto-semantic dependencies directly encode information such as coordination, extraction, raising, control, as well as many other long-range dependencies. Experiments for a variety of formalisms and languages profile different aspects of transition-based deep dependency parsing models.

Figure 7 visualizes cross-format annotations assigned to the English sentence: *A similar technique is almost impossible to apply to other crops, such as cotton, soybeans, and rice.* This running example illustrates a range of linguistic phenomena such as coordination, verbal chains, argument and modifier prepositional phrases, complex noun phrases, and the so-called **tough** construction. The first format is from the popular corpus PropBank, which is widely used by various SRL systems. We can clearly see that compared with SRL, SDP uses dense graphs to represent much more syntacto–semantic information. This difference suggests to us that we should explore different algorithms for producing SRL and SDP graphs. Another thing worth noting is that, for the same phenomenon, annotation schemes may not agree with each other. Take the coordination construction, for example. For more details about the difference among different data sets, please refer to Ivanova et al. (2012).

For CCG analysis, we conduct experiments on English and Chinese CCGBank (Hockenmaier and Steedman 2007; Tse and Curran 2010). Following previous experimental set-up for English CCG parsing, we use Section 02-21 as training data, Section 00 as the development data, and Section 23 for testing. To conduct Chinese parsing experiments, we use data setting **C** of Tse and Curran (2012). For grammatical relation analysis, we conduct experiments on Chinese GRBank data (Sun et al. 2014). The selection for training, development, and test data is also according to Sun et al.'s (2014) experiments.

We also evaluate all parsing models using more HPSG-grounded semantics-oriented data, namely, DeepBank[2] (Flickinger, Zhang, and Kordoni 2012) and EnjuBank (Miyao, Ninomiya, and ichi Tsujii 2004). Different from Penn Treebank–converted corpus, DeepBank's annotations are essentially based on the parsing results given a large-scale linguistically precise HPSG grammar, namely, LingGO English resource grammar (ERG; Flickinger 2000), and manually disambiguated. As part of the full HPSG sign, the ERG also makes available a logical-form representation of propositional semantics, in the framework of minimal recursion semantics (MRS; Copestake et al. 2005). Such semantic information is reduced into variable-free bilexical dependency graphs (Oepen and Lønning 2006; Ivanova et al. 2012). In summary, DeepBank gives the *reduction of logical-form meaning representations* with respect to MRS. EnjuBank (Miyao, Ninomiya, and ichi Tsujii 2004) provides another corpus for semantic dependency parsing. This type of annotation is somehow shallower than DeepBank, given that only basic predicate–argument structures are concerned. Different from DeepBank but similar to CCGBank and GRBank, EnjuBank is semi-automatically converted from Penn Treebank–style annotations with linguistic heuristics. To conduct HPSG experiments, we use Sections 00 to 19 as training data and Section 20 as development data to tune parameters. For final

[2] http://moin.delph-in.net/DeepBank.


A1
A2
A similar technique is almost impossible to apply to other crops, such as cotton, soybeans and rice
ROOT
arg2
BV
arg1
arg1
arg1
arg1
arg3
arg1
arg2
implicit_conj
_and_c
A similar technique is almost impossible to apply to other crops, such as cotton, soybeans and rice.
ROOT
arg1
arg1
arg1
arg1
arg1
arg2
arg1
arg2
arg2
arg1
arg2
arg1
arg1
arg1
arg1
arg1
arg2
arg1
arg2
arg2
A similar technique is almost impossible to apply to other crops , such as cotton , soybeans and rice
ROOT
arg3
arg1
arg1
arg1
arg1
arg2
arg2
arg2
arg2
arg2
arg1
arg1
arg1
arg2
arg2
arg2
arg2
A similar technique is almost impossible to apply to other crops , such as cotton , soybeans and rice

(a) Format 1: Propositional semantics, from PropBank.

(b) Format 2: MRS-derived dependencies, from DeepBank HPSG annotations.

(c) Format 3: Predicate-argument structures, from Enju HPSG annotation.

(d) Format 4: Functor-argument structures, from CCGBank.

Figure 7
Dependency representations in (a) PropBank, (b) DeepBank, (c) Enju HPSGBank, and (d) CCGBank formats.

evaluation, we use Sections 00 to 20 as training data and section 21 as test data. The DeepBank and EnjuBank data sets are from SemEval 2014 Task 8 (Oepen et al. 2014), and the data splitting policy follows the shared task. Table 1 gives a summary of the data sets for experiments.

Experiments for English CCG-grounded analysis were performed using automatically assigned POS-tags that are generated by a symbol-refined generative HMM tagger[3] (SR-HMM; Huang, Harper, and Petrov 2010). Experiments for English HPSG-grounded analysis used POS-tags provided by the shared task. For the experiments on Chinese CCGBank and GRBank, we use gold-standard POS tags.

We use the averaged perceptron algorithm with early update to estimate parameters, and beam search for decoding. We set the beam size to 16 and the number of iterations to 20 for all experiments. The measure for comparing two dependency graphs is precision and recall of tokens that are defined as $\langle w_h, w_d, l \rangle$ tuples, where w_h is the head, w_d is the dependent, and l is the relation. Labeled precision/recall (LP/LR) is the ratio of tuples correctly identified by the automatic generator, and unlabeled precision/recall (UP/UR) is the ratio regardless of l. F-score is a harmonic mean of precision and recall. These measures correspond to attachment scores (LAS/UAS) in dependency tree parsing and also used by the SemEval 2014 Task 8. The de facto standard

[3] http://www.code.google.com/p/umd-featured-parser/.

Table 1
Data sets for experiments. Columns "Training" and "Test" present the number of sentences in training and test sets, respectively.

Language	Formalism	Data	Training	Test
English	CCG	CCGBank	39,604	2,407
	HPSG	DeepBank	34,003	1,348
	HPSG	EnjuBank	34,003	1,348
Chinese	CCG	CCGBank	22,339	2,813
	LFG	GRBank	22,277	2,557

to evaluate CCG parsers also considers supertags. Because no supertagging is performed in our experiments, only the unlabeled precision/recall/F-score is comparable to the results reported in other papers. And the labeled performance reported here only considers the labels assigned to dependency arcs that indicate the argument types. For example, an arc label *arg1* denotes that the dependent is the first argument of the head.

5.2 Parsing Efficiency

We evaluate the real running time of our final trained parser using realistic data. The test sentences are collected from English Wikipedia and Chinese Gigaword (LDC2005T14). First, we show the influence of beam size in Figure 8. In this experiment, the DeepBank trained models are used for test. We can see that the parsers run in nearly linear time regardless of the beam width in realistic situations. Second, we report the the averaged real running time of models trained on different data sets in Figure 9. Again, we can see that the parser runs in close to linear time for a variety of linguistically motivated representations. The results also suggest that our proposed transition-based parsers can automatically learn the complexity of linguistically motivated dependency structures from an annotated corpus. Note that although within the deep parsing framework, the study of formal grammars is partially relevant for data-driven dependency parsing,

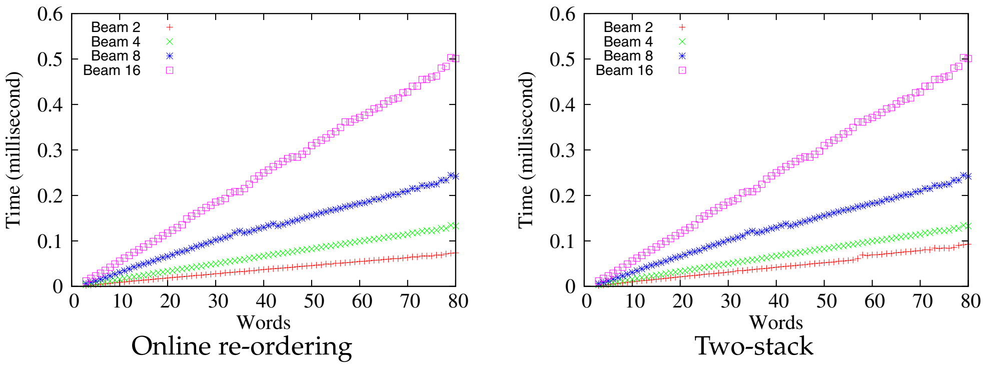
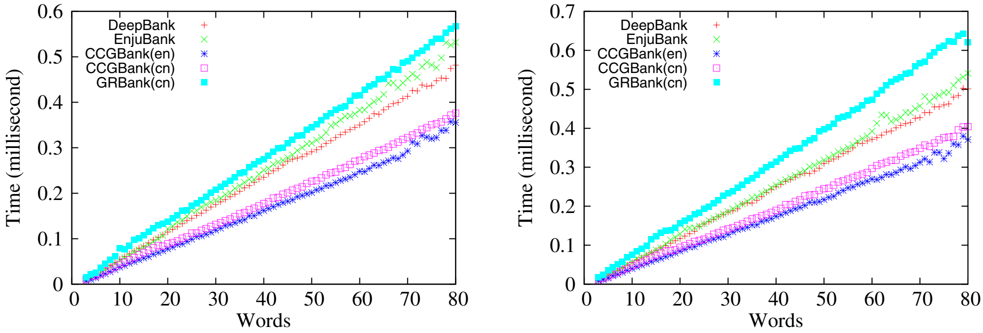


Figure 8
Real running time relative to beam size. Tested using DeepBank-trained models.



Online re-ordering

Two-stack

Figure 9
Real running time relative to models trained on different data sets.

where our parsers rely on inductive inference from treebank data, and only *implicitly* use a grammar.

5.3 Importance of Transition Combination

Figure 10 and Table 2 summarize the labeled parsing results on all of the five data sets. In this experiment, we distinguish parsing models with and without transition combination. All models take only the surface word form and POS tag information and do not derive features from any syntactic analysis. The importance of transition combination is highlighted by the comparative evaluation on parsers using this mechanism or not. Significant improvements are observed over a wide range of conditions: Parsers based on different transition systems for different languages and different formalisms almost always benefit. This result suggests a necessary strategy for designing transition systems for producing deep dependency graphs: Configurations should be essentially modified by every transition.

Because of the importance of transition combination, all the following experiments utilize the transition combination strategy.

5.4 Model Diversity and Parser Ensemble

5.4.1 Model Diversity. For model ensemble, besides the accuracy of each single model, it is also essential that the models to be integrated should be very different. We argue that heterogeneous parsing models can be built by varying the underlying transition systems. By reversing the sentence from right to left, we can build other model variants with the same transition system. To evaluate the differences between two models A and B, we define the following metric:

$$\frac{2 * |\mathcal{D}_A \cap \mathcal{D}_B|}{|\mathcal{D}_A| + |\mathcal{D}_B|}$$

where $\mathcal{D}_X$ denotes the set of dependencies related to held out sentences returned by model X. Tables 3 and 4 show the model diversity evaluated on English and Chinese data, respectively. We can see that parsing models built upon different transition systems do vary. Even for one specific transition system, different processing directions yield quite different parsing results.

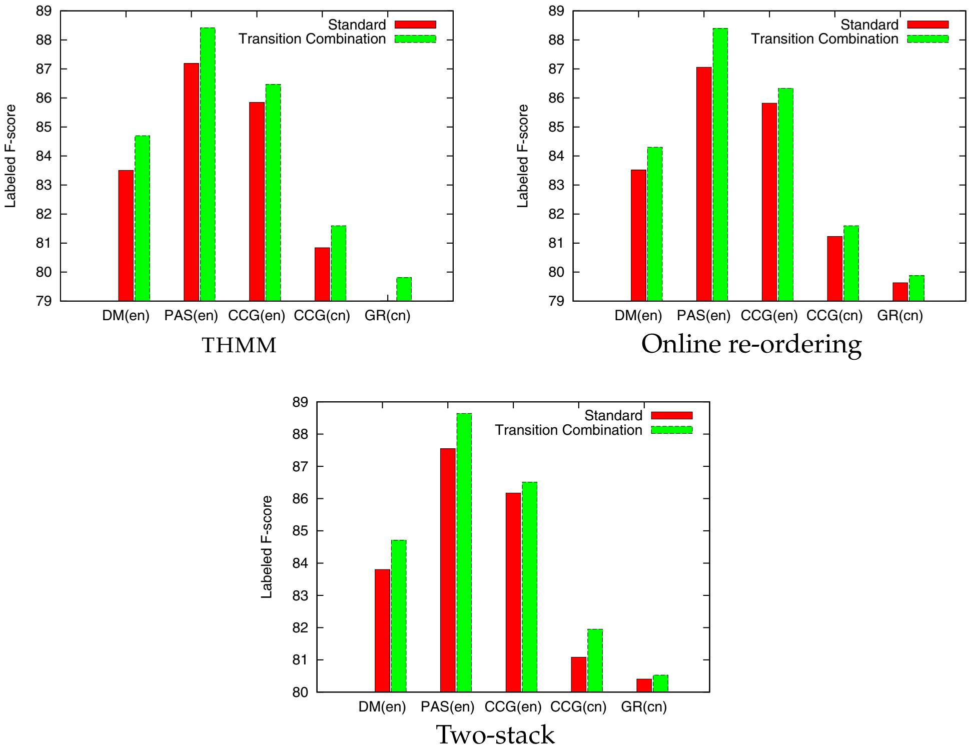


Figure 10
Labeled parsing F-scores of different transition system with and without transition combination. "Standard" denotes the *standard* systems, which do not combine an ARC transition with its following transition.

5.4.2 Parser Ensemble. Parser ensemble has been shown very effective to boost the performance of data-driven tree parsers (Nivre and McDonald 2008; Surdeanu and Manning 2010; Sun and Wan 2013). Empirically, the two proposed systems together with the existing THMM system exhibit complementary prediction powers, and their combination yields superior accuracy. We present a simple yet effective voting strategy for parser ensemble. For each pair of words in each sentence, we count the number of models that give positive predictions. If the number is greater than a threshold (we set it to half the number of models in this work), we put this arc to the final graph, and label the arc with the most common label of what the models give.

Table 5 presents the parsing accuracy of the combined model where six base models are utilized for voting. We can see that a system ensemble is quite helpful. Given that our graph parsers all run in expected linear time, the combined system also runs very efficiently.

5.5 Impact of Syntactic Parsing

5.5.1 Effectiveness of Syntactic Features. Syntactic parsing, especially the full one, has been shown very important for boosting the performance of SRL, a well studied shallow semantic parsing task (Punyakanok, Roth, and Yih 2008). According to the comprehensive evaluation presented in Punyakanok, Roth, and Yih (2008) and Zhuang and Zong

Table 2
Performance of different transition system with and without transition combination on the test set of the DeepBank/EnjuBank data, on the development set of the English and Chinese CCGBank data, and on the development set of the Chinese GRBank data. S_x^{std} denotes the *standard* system, which does not combine an ARC transition with its following transition.

English

	DeepBank			EnjuBank			CCGBank		
	LP	LR	LF	LP	LR	LF	LP	LR	LF
S_T^{std}	82.71%	84.32%	83.51	6.89%	87.48%	87.19	85.88%	85.81%	85.85
S_T	85.00%	84.40%	84.70	88.66%	88.17%	88.42	87.00%	85.93%	86.46
S_S^{std}	82.60%	84.46%	83.52	86.72%	87.38%	87.06	85.60%	86.04%	85.82
S_S	84.63%	83.98%	84.30	88.58%	88.20%	88.39	86.63%	86.04%	86.33
S_{2S}^{std}	82.97%	84.65%	83.80	87.25%	87.75%	87.55	86.06%	86.29%	86.17
S_{2S}	85.01%	84.41%	84.71	88.80%	88.48%	88.64	86.77%	86.25%	86.51

Chinese

	CCGBank			GRBank		
	LP	LR	LF	LP	LR	LF
S_T^{std}	80.93%	80.75%	80.84	80.10%	77.95%	79.01
S_T	82.04%	81.16%	81.60	81.28%	78.40%	79.81
S_S^{std}	80.86%	81.60%	81.23	80.32%	78.96%	79.63
S_S	81.71%	81.48%	81.60	80.30%	79.46%	79.88
S_{2S}^{std}	80.81%	81.35%	81.08	80.58%	80.23%	80.41
S_{2S}	82.09%	81.81%	81.95	80.88%	80.18%	80.53

(2010) (see Table 6), there is an essential gap between full and shallow parsing-based SRL systems. If we consider a system that takes only word form and POS tags as input, the performance gap will be larger.

When we consider semantics-oriented deep dependency structures, including the representations for CCG-grounded functor–argument (Clark, Hockenmaier, and Steedman 2002) analysis, HPSG-grounded predicate–argument analysis (Miyao, Ninomiya, and ichi Tsujii 2004), and reduction of MRS (Ivanova et al. 2012), syntactic parses can also provide very useful features for disambiguation. To evaluate the impact of syntactic tree parsing, we include more features, namely, path features, to our parsing models. The detailed description of syntactic features are presented in Section 3.3. In this work, we apply syntactic dependency parsers rather than phrase-structure parsers. Figure 11 summarizes the impact of features derived from syntactic trees. We can clearly see that syntactic features are effective to enhance semantic dependency parsing. These informative features lead to on average 1.14% and 1.03% absolute improvements for English and Chinese CCG parsing. Compared with SRL, the improvement brought by syntactic parsing is smaller. We think one main reason for this difference is the information density of different types of graphs. SRL graphs usually annotate only on verbal predicates and their nominalization, whereas the semantic graphs grounded by CCG and HPSG target all words. In other words, SRL provides partial analysis and semantic dependency parsing provides full analysis. Accordingly, SRL needs structural information generated by a syntactic parser much more than semantic dependency parsing.

Table 3
Model diversity between different models on the test set of the DeepBank/EnjuBank data and on the development set of the English CCGBank data. S_x^{rev} means processing a sentence with system S_x but in the right-to-left word order.

			DeepBank		
	S_S	S_{2S}	S_T^{rev}	S_S^{rev}	S_{2S}^{rev}
S_T	0.9285	0.9285	0.8788	0.8796	0.8797
S_S		0.9385	0.8748	0.8776	0.8773
S_{2S}			0.8772	0.8802	0.8790
S_T^{rev}				0.9390	0.9364
S_S^{rev}					0.9413

			EnjuBank		
	S_S	S_{2S}	S_T^{rev}	S_S^{rev}	S_{2S}^{rev}
S_T	0.9504	0.9481	0.9045	0.9038	0.9043
S_S		0.9503	0.9046	0.9060	0.9055
S_{2S}			0.9066	0.9087	0.9076
S_T^{rev}				0.9562	0.9565
S_S^{rev}					0.9584

			CCGBank		
	S_S	S_{2S}	S_T^{rev}	S_S^{rev}	S_{2S}^{rev}
S_T	0.9547	0.9532	0.9155	0.9164	0.9182
S_S		0.9575	0.9166	0.9179	0.9187
S_{2S}			0.9200	0.9197	0.9205
S_T^{rev}				0.9586	0.9575
S_S^{rev}					0.9617

Table 4
Model diversity between different models on the development set of the Chinese CCGBank/GRBank data.

			CCGBank		
	S_S	S_{2S}	S_T^{rev}	S_S^{rev}	S_{2S}^{rev}
S_T	0.9261	0.9262	0.8668	0.8614	0.8658
S_S		0.9314	0.8667	0.8593	0.8663
S_{2S}			0.8694	0.8624	0.8683
S_T^{rev}				0.9130	0.9107
S_S^{rev}					0.9230

			GRBank		
	S_S	S_{2S}	S_T^{rev}	S_S^{rev}	S_{2S}^{rev}
S_T	0.8918	0.8861	0.8398	0.8301	0.8328
S_S		0.9058	0.8455	0.8378	0.8414
S_{2S}			0.8460	0.8391	0.8441
S_T^{rev}				0.8969	0.8984
S_S^{rev}					0.9111

Table 5
Performance of base and combined models on the test set of the DeepBank/EnjuBank data, on the development set of the English and Chinese CCGBank data and on the development set of the Chinese GRBank data. Note that the labeled results for CCG parsing do not consider supertags.

English

DeepBank	UP	UR	UF	LP	LR	LF
S_T	87.03%	86.42%	86.72	85.00%	84.40%	84.70
S_T^{rev}	88.16%	88.16%	88.16	86.12%	86.11%	86.12
S_S	86.57%	85.91%	86.24	84.63%	83.98%	84.30
S_S^{rev}	88.53%	88.27%	88.40	86.63%	86.38%	86.51
S_{2S}	86.99%	86.38%	86.68	85.01%	84.41%	84.71
S_{2S}^{rev}	88.12%	88.11%	88.11	86.28%	86.26%	86.27
Combined	88.29%	90.27%	89.27	86.46%	88.40%	87.42
EnjuBank	UP	UR	UF	LP	LR	LF
S_T	89.98%	89.47%	89.72	88.66%	88.17%	88.42
S_T^{rev}	91.93%	91.92%	91.92	90.67%	90.66%	90.67
S_S	89.86%	89.48%	89.67	88.58%	88.20%	88.39
S_S^{rev}	92.12%	92.04%	92.08	90.86%	90.79%	90.82
S_{2S}	90.07%	89.75%	89.91	88.80%	88.48%	88.64
S_{2S}^{rev}	92.09%	92.01%	92.05	90.88%	90.80%	90.84
Combined	91.31%	93.62%	92.45	90.15%	92.43%	91.28
CCGBank	UP	UR	UF	LP	LR	LF
S_T	91.10%	89.98%	90.54	87.00%	85.93%	86.46
S_T^{rev}	91.23%	91.27%	91.25	87.25%	87.28%	87.27
S_S	90.80%	90.18%	90.49	86.63%	86.04%	86.33
S_S^{rev}	91.12%	91.31%	91.22	87.35%	87.53%	87.44
S_{2S}	90.85%	90.30%	90.58	86.77%	86.25%	86.51
S_{2S}^{rev}	91.57%	91.63%	91.60	87.83%	87.89%	87.86
Combined	91.43%	92.83%	92.13	87.76%	89.10%	88.42

Chinese

CCGBank	UP	UR	UF	LP	LR	LF
S_T	86.24%	85.31%	85.77	82.04%	81.16%	81.60
S_T^{rev}	85.20%	85.13%	85.16	80.97%	80.90%	80.94
S_S	85.86%	85.62%	85.74	81.71%	81.48%	81.60
S_S^{rev}	84.65%	85.90%	85.27	80.55%	81.74%	81.14
S_{2S}	86.17%	85.87%	86.02	82.09%	81.81%	81.95
S_{2S}^{rev}	85.14%	86.32%	85.73	81.05%	82.18%	81.61
Combined	86.63%	89.05%	87.82	82.83%	85.14%	83.97
GRBank	UP	UR	UF	LP	LR	LF
S_T	83.38%	80.43%	81.88	81.28%	78.40%	79.81
S_T^{rev}	85.03%	84.05%	84.54	82.93%	81.98%	82.45
S_S	82.35%	81.49%	81.92	80.30%	79.46%	79.88
S_S^{rev}	83.74%	85.04%	84.39	81.77%	83.05%	82.41
S_{2S}	82.93%	82.20%	82.56	80.88%	80.18%	80.53
S_{2S}^{rev}	83.84%	85.02%	84.43	81.80%	82.94%	82.37
Combined	86.05%	87.14%	86.59	84.06%	85.12%	84.59

Table 6
Performance of English and Chinese SRL achieved by representative full and shallow parsing-based systems. The results are copied from Punyakanok, Roth, and Yih (2008) and Zhuang and Zong (2010).

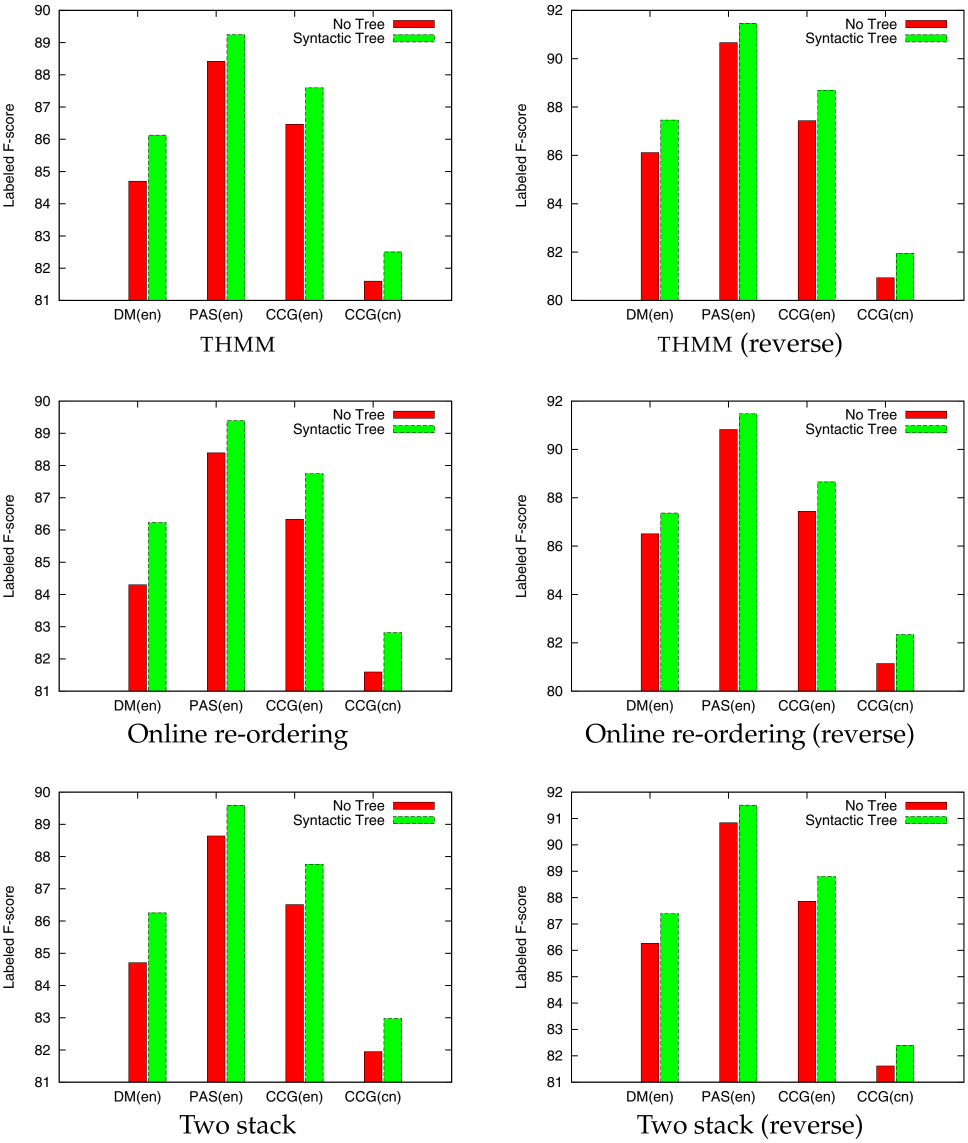
		Precison	Recall	F-score
English	Full parsing	77.09%	75.51%	76.29
	Shallow parsing	75.48%	67.13%	71.06
Chinese	Full parsing	79.17%	72.09%	75.47
	Shallow parsing	72.57%	67.02%	69.68

5.5.2 Comparison of Different Tree Parsers. There are two dominant data-driven approaches to syntactic dependency tree parsing: transition-based (Yamada and Matsumoto 2003; Nivre 2008) and graph-based (McDonald 2006; Torres Martins, Smith, and Xing 2009). In terms of overall per token prediction, the transition-based and graph-based tree parsers achieve comparable performance (Suzuki et al. 2009; Weiss et al. 2015). To evaluate the impact of the two tree parsing approaches on semantic dependency parsing, we use two tree parsers to serve our graph parser. The first one is our in-house implementation of the algorithm presented in Zhang and Nivre (2011), and the second one is a second-order graph-based parser[4] (Bohnet 2010). The tree parsers are trained with the unlabeled tree annotations provided by the English and Chinese CCGBank data. For both English and Chinese experiments, 5-fold cross validation is performed to parse the training data to avoid overfitting. The accuracy of tree parsers is shown in Table 7. Results presented in Figure 12 indicate that the two parsers are also equivalently effective for producing semantic analysis. This result is somehow non-obvious given that the combination of a graph-based and transition-based parser usually gives significantly better parsing performance (Nivre and McDonald 2008; Torres Martins et al. 2008).

5.6 Effectiveness of Tree Approximation

In case syntactic information is not available, we propose a tree approximation technique to induce tree backbones from deep dependency graphs. In particular, our technique guarantees that the automatically derived trees are projective, which is a necessary condition for a number of effective tree parsing algorithms. We can utilize these *pseudo* trees as an alternative to syntactic analysis. To evaluate the effectiveness of tree approximation, we compare the contribution to semantic dependency parsing of syntactic trees and pseudo trees. In this experiment, we use a transition-based tree parser to generate automatic analysis. Figure 13 presents the results. Generally speaking, *pseudo* trees contribute to semantic dependency parsing equally well as syntactic trees. Sometimes, they perform even better. There is a considerable drop when DeepBank data are applied. We think the main reason is the density of DeepBank graphs. Because there are fewer edges in the original graphs, it is harder to extract informative pseudo trees. As a result, the final graph parsing benefits less.

[4] `http://www.code.google.com/p/mate-tools/`.



THMM

THMM (reverse)

Online re-ordering

Online re-ordering (reverse)

Two stack

Two stack (reverse)

Figure 11
Parsing accuracy with and without syntactic features. The syntactic trees for experiments on DeepBank and EnjuBank data sets are provided by the SemEval 2014 shared task, and they are automatically generated by the Stanford Parser. The syntactic trees for experiments on English and Chinese CCG data sets are generated by our in-house implementation of the model introduced in Zhang and Nivre (2011).

It is also possible to build a parser ensemble on pseudo tree enhanced models. However, the effectiveness of system combination is not as effective as integrating non-tree models. Table 8 summarizes the detailed parsing accuracy. We can see that system ensemble is still helpful, though the improvement is limited.

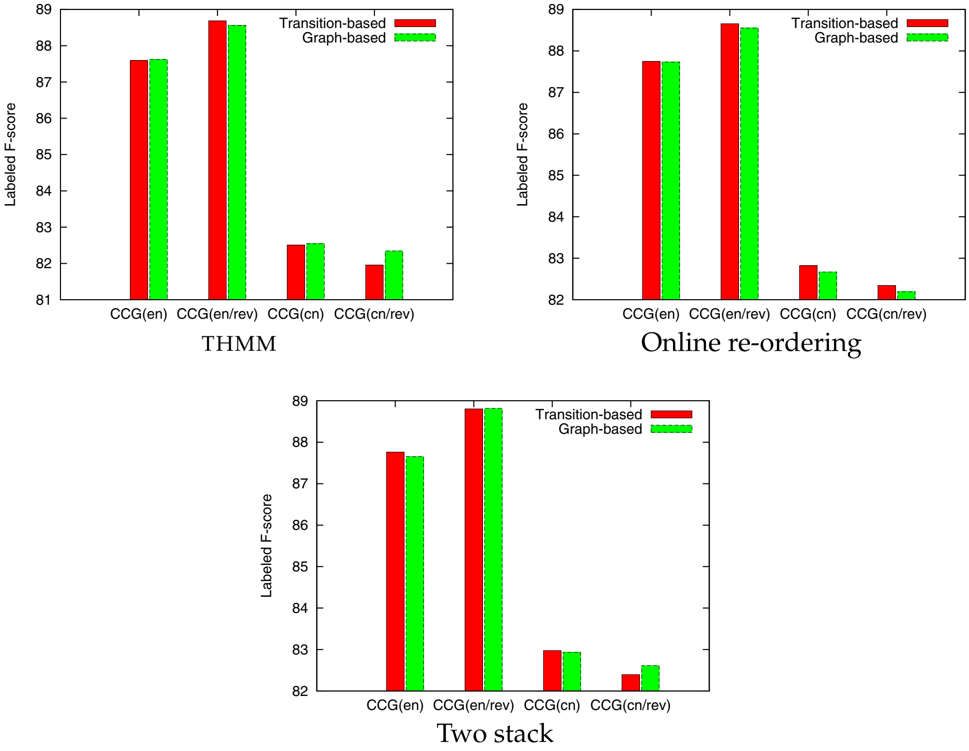
Table 7
Accuracy of preprocessing on the development data for CCG analysis. *Tr* and *Gr*, respectively, denote transition-based and graph-based tree parsers.

	UAS(Tr)	UAS(Gr)
English	93.48%	93.47%
Chinese	80.97%	80.81%

5.7 Comparison with Other Parsers

5.7.1 Comparison with Grammar-Based Parsers. We compare our parser with several representative Treebank-guided, grammar-based parsers that achieve state-of-the-art performance for CCG and HPSG analysis. The grammar-based parsers selected represent two different architectures.

- The first type of parser implements a shift-reduce parsing architecture and also uses beam search for practical decoding. In particular, we compare our parser with the state-of-the-art CCG parser introduced in Xu, Clark, and

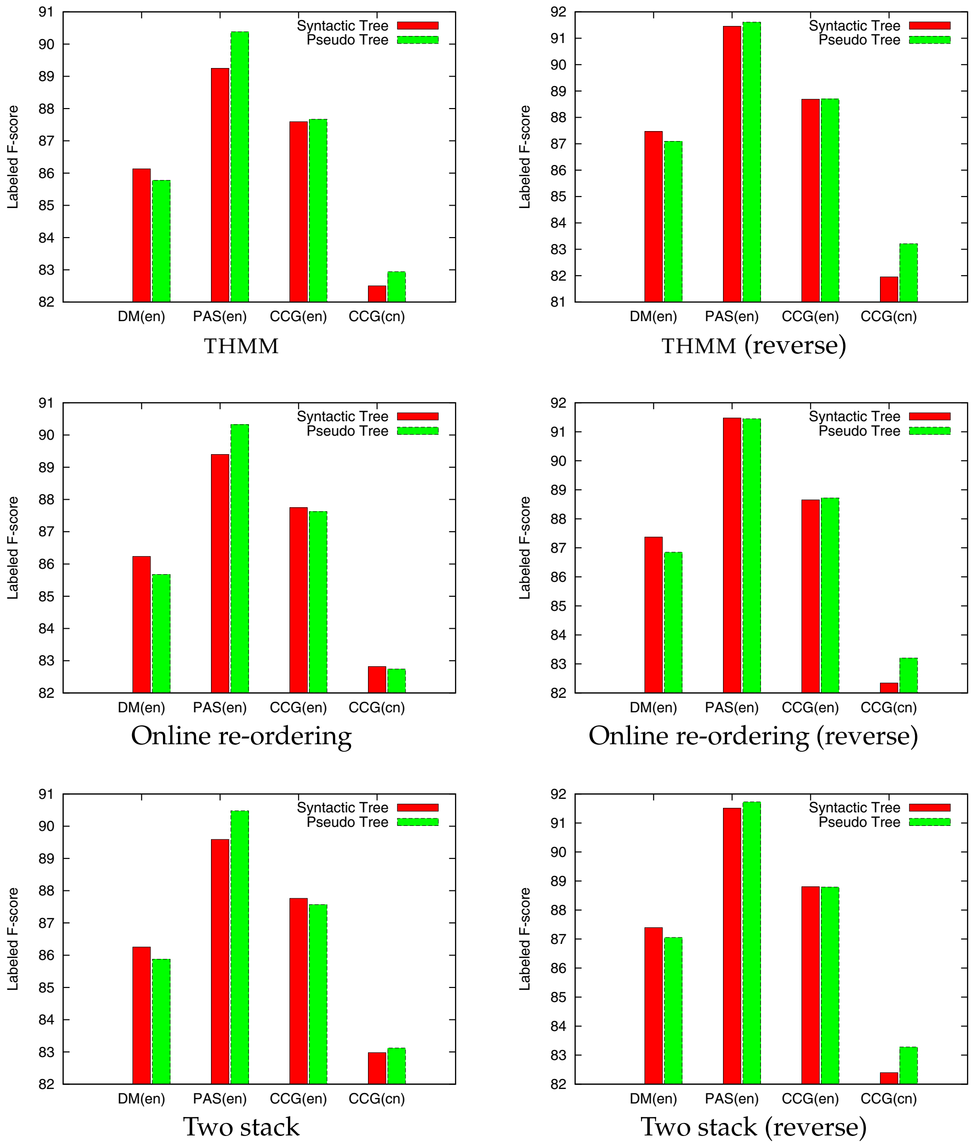


THMM

Online re-ordering

Two stack

Figure 12
Labeled F-scores with respect to different tree parsing techniques. Results shown here are from experiments for English and Chinese CCG parsing.



THMM

THMM (reverse)

Online re-ordering

Online re-ordering (reverse)

Two stack

Two stack (reverse)

Figure 13
Parsing accuracy based on syntactic and pseudo tree features. All trees are generated by our in-house implementation of the model introduced in Zhang and Nivre (2011).

Zhang (2014).[5] This parser extends a shift-reduce CFG parser (Zhang and Clark 2011a) with a dependency model.

- The second type of parser implements the chart parsing architecture with some refinements. For CCG analysis, we focus on the parser proposed by

[5] The unlabeled parsing results are not reported in the original paper. The figures presented in Table 9 are provided by Wenduan Xu.

Table 8
Performance of base and combined models on the test set of the DeepBank/EnjuBank data and on the development set of the English and Chinese CCGBank data. Features extracted from *pseudo* trees are utilized for disambiguation.

			English			
DeepBank	UP	UR	UF	LP	LR	LF
S_T	87.99%	87.64%	87.81	85.95%	85.61%	85.78
S_T^{rev}	88.94%	88.98%	88.96	87.07%	87.11%	87.09
S_S	87.76%	87.45%	87.60	85.83%	85.52%	85.67
S_S^{rev}	88.72%	88.65%	88.69	86.88%	86.82%	86.85
S_{2S}	87.92%	87.60%	87.76	86.03%	85.72%	85.87
S_{2S}^{rev}	89.04%	88.85%	88.95	87.15%	86.96%	87.05
Combined	88.54%	90.25%	89.39	86.65%	88.32%	87.48
EnjuBank	UP	UR	UF	LP	LR	LF
S_T	91.88%	91.45%	91.66	90.60%	90.17%	90.38
S_T^{rev}	92.82%	92.83%	92.83	91.61%	91.61%	91.61
S_S	91.76%	91.39%	91.58	90.50%	90.14%	90.32
S_S^{rev}	92.66%	92.65%	92.65	91.45%	91.44%	91.45
S_{2S}	91.85%	91.54%	91.70	90.63%	90.33%	90.48
S_{2S}^{rev}	92.92%	92.83%	92.87	91.77%	91.68%	91.73
Combined	92.47%	93.52%	92.99	91.34%	92.38%	91.86
CCGBank	UP	UR	UF	LP	LR	LF
S_T	92.15%	91.05%	91.60	88.20%	87.15%	87.67
S_T^{rev}	92.46%	92.27%	92.37	88.78%	88.61%	88.70
S_S	91.91%	91.18%	91.54	87.97%	87.28%	87.62
S_S^{rev}	92.34%	92.43%	92.39	88.67%	88.76%	88.72
S_{2S}	91.86%	91.13%	91.49	87.92%	87.22%	87.57
S_{2S}^{rev}	92.53%	92.41%	92.47	88.85%	88.73%	88.79
Combined	92.38%	93.20%	92.79	88.92%	89.71%	89.31

			Chinese			
CCGBank	UP	UR	UF	LP	LR	LF
S_T	87.44%	86.41%	86.93	83.44%	82.45%	82.94
S_T^{rev}	87.11%	87.04%	87.07	83.24%	83.17%	83.21
S_S	86.76%	86.52%	86.64	82.85%	82.63%	82.74
S_S^{rev}	86.46%	87.54%	87.00	82.69%	83.72%	83.20
S_{2S}	87.09%	86.91%	87.00	83.21%	83.03%	83.12
S_{2S}^{rev}	86.57%	87.69%	87.13	82.75%	83.82%	83.28
Combined	87.27%	89.00%	88.12	83.57%	85.23%	84.39

Auli and Lopez (2011b). The basic system architecture follows the well-engineered C&C Parser,[6] and additionally applies a number of advanced machine learning and optimization techniques, including belief propagation, dual decomposition Auli and Lopez (2011a), and parameter estimation with softmax-margin loss (Auli and Lopez 2011b), to enhance the results. For HPSG analysis, we compare with the well-studied Enju

[6] http://svn.ask.it.usyd.edu.au/trac/candc.

Table 9
Parsing results on test sets obtained by representative parsers. State-of-the-art results on these data sets, as reported in Oepen et al. (2014), Martins and Almeida (2014), Xu, Clark, and Zhang (2014), Auli and Lopez (2011b), Du, Sun, and Wan (2015), Sun et al. (2014), are included.

DeepBank		LP	LR	LF
Our system	S_{2S}^{rev}	86.28%	86.26%	86.27
	Combined	86.46%	88.40%	87.42
	S_{2S}^{rev}+Pseudo Tree	87.15%	86.96%	87.05
	Combined+Pseudo Tree	86.65%	88.32%	87.48
Factorization (Turbo)	(Martins and Almeida 2014)	88.82%	87.35%	88.08
EnjuBank		LP	LR	LF
Our system	S_{2S}^{rev}	90.88%	90.80%	90.84
	Combined	90.15%	92.43%	91.28
	S_{2S}^{rev}+Pseudo Tree	91.77%	91.68%	91.73
	Combined+Pseudo Tree	91.34%	92.38%	91.86
Chart parsing (Enju)	(Oepen et al. 2014)	92.09%	92.02%	92.06
Factorization (Turbo)	(Martins and Almeida 2014)	91.95%	89.92%	90.93
English CCGBank		UP	UR	UF
Our system	S_{2S}^{rev}	91.84%	91.75%	91.80
	Combined	92.06%	93.14%	92.60
	S_{2S}^{rev}+Pseudo Tree	92.49%	92.30%	92.40
	Combined+Pseudo Tree	92.52%	93.13%	92.82
Shift-reduce	(Xu, Clark, and Zhang 2014)	93.15%	91.06%	92.09
Chart parsing	(Auli and Lopez 2011b)	93.08%	92.44%	92.76
Factorization	(Du, Sun, and Wan 2015)	93.03%	92.03%	92.53
Chinese GRBank		LP	LR	LF
Our system	S_{2S}^{rev}	82.28%	83.11%	82.69
	Combined	84.92%	85.28%	85.10
Transition-based	(Sun et al. 2014)	83.93%	79.82%	81.82
Chinese CCGBank		UP	UR	UF
Our system	S_{2S}^{rev}	85.07%	86.02%	85.54
	Combined	86.35%	88.85%	87.58
	S_{2S}^{rev}+Pseudo Tree	86.65%	87.34%	86.99
	Combined+Pseudo Tree	87.14%	88.60%	87.86

Parser,[7] which develops a number of advanced techniques for discriminative deep parsing—for example, maximum entropy estimation with feature forest (Miyao and Tsujii 2008) and efficient decoding with supertagging and CFG-filtering (Matsuzaki, Miyao, and Tsujii 2007).

Table 9 shows the final results on the test data for each data set. The representative shift-reduce parser for comparison utilizes a very similar learning and decoding architectures to our system. Similar to our parser, Xu, Clark, and Zhang's (2014) parser incrementally processes a sentence and uses a beam decoder that performs an inexact

[7] http://kmcs.nii.ac.jp/enju/?lang=en.

search. Xu, Clark, and Zhang's parser sets beam width to 128, while ours is 16. It also uses the structured prediction algorithm for parameter estimation. The major difference is that the shift-reduce CCG parser explicitly utilizes a core grammar to guide decoding, whereas our parser excludes all such information. Actually, our models reported here also exclude all syntactic information because no syntactic parse is used for feature extraction. We can see that our individual system based on the two stack transition system achieves equivalent performance to the CCG-driven parser. Moreover, when this individual system is augmented with tree approximation, the accuracy is significantly improved. Note that the individual system with both settings does not rely on any explicit syntactic information. This result on one hand indicates the effectiveness of adapting syntactic parsing techniques for full semantic parsing, and on the other hand suggests the possibility of using semantically structural (not syntactically structural) information only to achieve high-accuracy semantic parsing.

Statistical parsers based on chart parsing are able to perform a more principled search and therefore usually achieve better parsing accuracy than a *normal* shift-reduce parser. We also compare our parsing models with two state-of-the-art chart parsers, namely, the Enju Parser (Miyao and Tsujii 2008) and Auli and Lopez's (2011b) parser. Different from Xu, Clark, and Zhang's (2014) shift-reduce parser and our models, Auli and Lopez's (2011b) parser does not guarantee to produce analysis for arbitrary sentences. Usually, the numerical performance evaluated on all sentences is lower than the results obtained on sentences that can be parsed. Note that Auli and Lopez (2011b) only reported results on sentences that are covered, whereas Oepen et al. (2014) reported results on all sentences, which is achieved by Enju Parser. From Table 9, we can clearly see that our graph-spanning models are very competitive. The best individual and combined models outperform the Enju Parser and perform equally well to Auli and Lopez's (2011b) parser. It is worth noting that strictly less information is used by our parsers.

5.7.2 Comparison with Other Data-Driven Parsers. We also compare our parser with recently developed data-driven, factorization models (Martins and Almeida 2014; Du, Sun, and Wan 2015). Different from projective but similar to non-projective tree parsing, decoding for factorization models where very basic second-order sibling factors are incorporated is NP-hard. See the proof presented in our early work (Du, Sun, and Wan 2015) for details. To perform principled decoding, dual decomposition is used and achieves good empirical results (Martins and Almeida 2014; Du, Sun, and Wan 2015).

From Table 9, we can see that the transition-based approach augmented with tree approximation is comparable to the factorization approach in general. Compared with the Turbo Parser, our individual and hybrid models perform significantly worse on DeepBank but significantly better on EnjuBank. We think one main reason is because of the annotation styles. Though both corpora are based on HPSG, the annotations in question are quite different. DeepBank graphs are more sparse than EnjuBank, which makes tree approximation less effective. It seems that the transition-based parser suffers more when fewer output edges are targeted. The two approaches achieve equivalent performance for CCG parsing.

6. Related Work

Deep linguistic processing is concerned with NLP approaches that aim at modeling the complexity of natural languages in rich linguistic representations. Such approaches are typically related to a particular computational linguistic theory (e.g., CCG, LFG, and

HPSG). Parsing in these formalisms provides an elegant way to generate deep syntacto-semantic dependency structures with high quality (Clark and Curran 2007; Miyao, Sagae, and Tsujii 2007; Miyao and Tsujii 2008). The incremental shift-reduce parsing architecture has been implemented for CCG parsing (Zhang and Clark 2011a; Ambati et al. 2015). Besides using phrase-structure rules only, a shift-reduce parser can be enhanced by incorporating a dependency model (Xu, Clark, and Zhang 2014). Our parser and the two above parsers have some essential resemblances, including learning and decoding algorithms. The main difference is the usage of syntactic and grammatical information. The comparison in Section 5.7 gives a rough idea of the impact of explicitly using grammatical constraints. A deep-grammar-guided parsing model usually cannot produce full coverage and the time complexity of the corresponding parsing algorithms is very high. Some NLP applications may favor lightweight solutions to build deep dependency structures.

Different from grammar-guided approaches, data-driven approaches make essential use of machine learning from linguistic annotations in order to parse new sentences. Such approaches, for example, transition-based (Yamada and Matsumoto 2003; Nivre 2008) and graph-based (McDonald 2006; Torres Martins, Smith, and Xing 2009) models, have attracted the most attention of dependency parsing in recent years. Several successful parsers (e.g., MST, Mate, and Malt parsers) have been built and applied to many NLP applications. Recently, two advanced techniques have been studied to enhance a transition-based parser. First, developing features has been shown crucial to advancing parsing accuracy and a very rich feature set is carefully evaluated by Zhang and Nivre (2011). Second, beyond deterministic greedy search, beam search and principled dynamic programming strategies have been used to explore more possible hypotheses (Zhang and Clark 2008; Huang and Sagae 2010). When we implement our graph parser, we also leverage rich features and beam search to obtain good parsing accuracy.

Most research concentrated on surface dependency structures, and the majority of existing approaches are limited to producing only tree-shaped graphs. We notice three distinguished exceptions in early work. Sagae and Tsujii (2008) proposed a DAG parser that is able to handle projective directed dependency graphs, and that uses the pseudo-projective parsing technique (Nivre and Nilsson 2005) to build crossing arcs. Titov et al. (2009) and Henderson et al. (2013) introduced non-planar parsing to parse PropBank (Palmer, Gildea, and Kingsbury 2005) structures. However, neither technique handles crossing arcs fully well. There have been a number of papers trying to build non-projective trees, which inspired the design of our transition systems. Especially, we borrow key ideas from Nivre (2009), Gómez-Rodríguez and Nivre (2010), and Gómez-Rodríguez and Nivre (2013). In addition to the investigation on the transition-based approach, McDonald and Pereira (2006) presented a factorization parser that can generate dependency graphs in which a word may depend on multiple heads, and evaluated it on the Danish Treebank. Very recently, the dual decomposition technique has been adopted to achieve principled decoding for factorization models. High-accuracy models have been introduced in Martins and Almeida (2014) and Du, Sun, and Wan (2015).

7. Conclusion

We study transition-based approaches that produce general dependency graphs directly from input sequences of words, in a way nearly as simple as tree parsers. We introduce two new graph-spanning algorithms to generate arbitrary directed graphs, which suit deep dependency parsing well. We also introduce transition combination and tree approximation for statistical disambiguation. Statistical parsers built upon these new techniques have been evaluated with dependency structures that are extracted from linguistically deep CCG, LFG, and HPSG derivations. Our models achieve state-of-the-art performance on five representative data sets for English and Chinese parsing. Experiments demonstrate the effectiveness of grammar-free, transition-based approaches to dealing with complex linguistic phenomena beyond surface syntax.

In addition to deep dependency parsing, many other NLP tasks (e.g., quantifier scope disambiguation [Manshadi, Gildea, and Allen 2013] and event extraction [Li, Ji, and Huang 2013]), can be formulated as graph spanning problems. We think such tasks can benefit from algorithms that span general graphs rather than trees, and our new transition-based parsers can provide practical solutions to these tasks.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under grants 61300064 and 61331011, and the National High-Tech R&D Program under grant 2015AA015403. We are very grateful to the anonymous reviewers for their insightful and constructive comments and suggestions.

References

Ambati, Bharat Ram, Tejaswini Deoskar, Mark Johnson, and Mark Steedman. 2015. An incremental algorithm for transition-based CCG parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 53–63, Denver, CO.

Auli, Michael and Adam Lopez. 2011a. A comparison of loopy belief propagation and dual decomposition for integrated CCG supertagging and parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 470–480, Portland, OR. Association for Computational Linguistics.

Auli, Michael and Adam Lopez. 2011b. Training a log-linear parser with loss functions via softmax-margin. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 333–343, Edinburgh.

Bohnet, Bernd. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing.

Bresnan, J. and R. M. Kaplan. 1982. Introduction: Grammars as mental representations of language. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA, pages xvii–lii.

Choi, Jinho D. and Martha Palmer. 2011. Getting the most out of transition-based dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 687–692, Portland, OR.

Clark, Stephen and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

Clark, Stephen, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures using a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 327–334. Philadelphia, PA.

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

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

Copestake, Ann, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2005. Minimal recursion semantics: An introduction. *Research on Language and Computation*, 3:281–332.

Covington, Michael A. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102. Athens, GA.

Du, Yantao, Weiwei Sun, and Xiaojun Wan. 2015. A data-driven, factorization parser for CCG dependency structures. In *Proceedings of the 53rd Annual Meeting of the*

Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 1545–1555, Beijing.

Du, Yantao, Fan Zhang, Weiwei Sun, and Xiaojun Wan. 2014. Peking: Profiling syntactic tree parsing techniques for semantic graph parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 459–464, Dublin.

Du, Yantao, Fan Zhang, Xun Zhang, Weiwei Sun, and Xiaojun Wan. 2015. Peking: Building semantic dependency graphs with a hybrid parser. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 927–931, Denver, CO.

Flickinger, Dan. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6(1):15–28.

Flickinger, Daniel, Yi Zhang, and Valia Kordoni. 2012. Deepbank: A dynamically annotated treebank of the Wall Street journal. In *Proceedings of the Eleventh International Workshop on Treebanks and Linguistic Theories*, pages 85–96, Lisbon.

Gómez-Rodríguez, Carlos and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501, Uppsala.

Gómez-Rodríguez, Carlos and Joakim Nivre. 2013. Divisible transition systems and multiplanar dependency parsing. *Computational Linguistics*, 39(4):799–845.

Hajič, Jan, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CONLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18, Boulder, CO.

Henderson, James, Paola Merlo, Ivan Titov, and Gabriele Musillo. 2013. Multilingual joint parsing of syntactic and semantic dependencies with a latent variable model. *Computational Linguistics*, 39(4):949–998.

Hockenmaier, Julia and Mark Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn treebank. *Computational Linguistics*, 33(3):355–396.

Huang, Liang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala.

Huang, Zhongqiang, Mary Harper, and Slav Petrov. 2010. Self-training with products of latent variable grammars. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 12–22, Cambridge, MA.

Ivanova, Angelina, Stephan Oepen, Lilja Øvrelid, and Dan Flickinger. 2012. Who did what to whom? A contrastive study of syntacto-semantic dependencies. In *Proceedings of the Sixth Linguistic Annotation Workshop*, pages 2–11, Jeju Island.

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

Li, Qi, Heng Ji, and Liang Huang. 2013. Joint event extraction via structured prediction with global features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 73–82, Sofia.

Manshadi, Mehdi, Daniel Gildea, and James Allen. 2013. Plurality, negation, and quantification: Towards comprehensive quantifier scope disambiguation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 64–72, Sofia.

Martins, André F. T. and Mariana S. C. Almeida. 2014. Priberam: A turbo semantic parser with second order features. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 471–476, Dublin.

Matsuzaki, Takuya, Yusuke Miyao, and Jun'ichi Tsujii. 2007. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of the 20th International Joint Conference on Artificial intelligence*, pages 1671–1676, San Francisco, CA.

McDonald, Ryan. 2006. *Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

McDonald, Ryan and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In

Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)), volume 6, pages 81–88, Trento.

McDonald, Ryan T. and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230.

Miyao, Yusuke, Takashi Ninomiya, and Jun'ichi Tsujii. 2004. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the penn treebank. In *IJCNLP*, pages 684–693, Hainan Island.

Miyao, Yusuke, Rune Sætre, Kenji Sagae, Takuya Matsuzaki, and Jun'ichi Tsujii. 2008. Task-oriented evaluation of syntactic parsers and their representations. In *Proceedings of ACL-08: HLT*, pages 46–54, Columbus, OH.

Miyao, Yusuke, Kenji Sagae, and Jun'ichi Tsujii. 2007. Towards framework-independent evaluation of deep linguistic parsers. In *Proceedings of the GEAF 2007 Workshop*, pages 238–258, Stanford, CA.

Miyao, Yusuke and Jun'ichi Tsujii. 2008. Feature forest models for probabilistic HPSG parsing. *Computational Linguistics*, 34(1):35–80.

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

Nivre, Joakim. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec.

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

Nivre, Joakim and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, MI.

Oepen, Stephan, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 63–72, Dublin.

Oepen, Stephan and Jan Tore Lønning. 2006. Discriminant-based MRS banking. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC-2006)*, Genoa.

Palmer, Martha, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31:71–106.

Pollard, Carl and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago.

Punyakanok, Vasin, Dan Roth, and Wen-tau Yih. 2008. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287.

Reddy, Siva, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics (TACL)*, 2:377–392.

Sagae, Kenji and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132, Stroudsburg, PA.

Sagae, Kenji and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 753–760, Manchester.

Steedman, Mark. 2000. *The Syntactic Process*. MIT Press, Cambridge, MA.

Sun, Weiwei, Yantao Du, Xin Kou, Shuoyang Ding, and Xiaojun Wan. 2014. Grammatical relations in Chinese: GB-ground extraction and data-driven parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 446–456, Baltimore.

Sun, Weiwei and Xiaojun Wan. 2013. Data-driven, PCFG-based and pseudo-PCFG-based models for Chinese dependency parsing. *Transactions of the Association for Computational Linguistics (TACL)*, 1:301–314.

Surdeanu, Mihai, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CONLL 2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177, Manchester.

Surdeanu, Mihai and Christopher D. Manning. 2010. Ensemble models for dependency parsing: Cheap and good?

In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 649–652, Los Angeles, CA.

Suzuki, Jun, Hideki Isozaki, Xavier Carreras, and Michael Collins. 2009. An empirical study of semi-supervised structured conditional models for dependency parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 551–560, Singapore.

Titov, Ivan, James Henderson, Paola Merlo, and Gabriele Musillo. 2009. Online graph planarisation for synchronous parsing of semantic and syntactic dependencies. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1562–1567, San Francisco, CA.

Torres Martins, Andre, Noah Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 342–350, Suntec.

Torres Martins, André Filipe, Dipanjan Das, Noah A. Smith, and Eric P. Xing. 2008. Stacking dependency parsers. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 157–166, Honolulu, HI.

Tse, Daniel and James R. Curran. 2010. Chinese CCGbank: Extracting CCG derivations from the Penn Chinese treebank. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1083–1091, Beijing.

Tse, Daniel and James R. Curran. 2012. The challenges of parsing Chinese with combinatory categorial grammar. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 295–304, Montréal.

Weiss, David, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing.

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

Yamada, Hiroyasu and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *8th International Workshop of Parsing Technologies (IWPT2003)*, pages 195–206, Nancy.

Zhang, Hui, Min Zhang, Chew Lim Tan, and Haizhou Li. 2009. K-best combination of syntactic parsers. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1552–1560, Singapore.

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

Zhang, Yue and Stephen Clark. 2011a. Shift-reduce CCG parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 683–692, Portland, OR.

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

Zhang, Yue and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, OR.

Zhuang, Tao and Chengqing Zong. 2010. A minimum error weighting combination strategy for Chinese semantic role labeling. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1362–1370, Beijing.