Parsing Chinese Sentences with Grammatical Relations

Weiwei Sun
Peking University
Institute of Computer Science and Technology and Center for Chinese Linguistics
ws@pku.edu.cn

Yufei Chen
Peking University
Institute of Computer Science and Technology
yufei.chen@pku.edu.cn

Xiaojun Wan
Peking University
Institute of Computer Science and Technology
wanxiaojun@pku.edu.cn

Meichun Liu
City University of Hong Kong
Department of Linguistics and Translation
meichliu@cityu.edu.hk

We report our work on building linguistic resources and data-driven parsers in the grammatical relation (GR) analysis for Mandarin Chinese. Chinese, as an analytic language, encodes grammatical information in a highly configurational rather than morphological way. Accordingly, it is possible and reasonable to represent almost all grammatical relations as bilexical dependencies. In this work, we propose to represent grammatical information using general directed dependency graphs. Both only-local and rich long-distance dependencies are explicitly represented. To create high-quality annotations, we take advantage of an existing TreeBank, namely, Chinese TreeBank (CTB), which is grounded on the Government and Binding theory. We define a set of linguistic rules to explore CTB's implicit phrase structural information and build deep dependency graphs. The reliability of this linguistically motivated GR extraction procedure is highlighted by manual

Submission received: 23 November 2017; revised version received: 19 September 2018; accepted for publication: 19 November 2018.

doi:10.1162/COLI_a_00343

evaluation. Based on the converted corpus, data-driven, including graph- and transition-based, models are explored for Chinese GR parsing. For graph-based parsing, a new perspective, graph merging, is proposed for building flexible dependency graphs: constructing complex graphs via constructing simple subgraphs. Two key problems are discussed in this perspective: (1) how to decompose a complex graph into simple subgraphs, and (2) how to combine subgraphs into a coherent complex graph. For transition-based parsing, we introduce a neural parser based on a list-based transition system. We also discuss several other key problems, including dynamic oracle and beam search for neural transition-based parsing. Evaluation gauges how successful GR parsing for Chinese can be by applying data-driven models. The empirical analysis suggests several directions for future study.

1. Introduction

Dependency grammar is a longstanding tradition that determines syntactic structures on the basis of word-to-word connections. It names a family of approaches to the linguistic analysis that all share a commitment to the typed relations between ordered pairs of words. Usually, dependencies represent various grammatical relations (GRs), which are exemplified in traditional grammars by the notions of subject, direct/indirect object, etc., and therefore encode rich syntactic information of natural language sentences in an explicit way. In recent years, parsing a sentence to a dependency representation has been well studied and widely applied to many Natural Language Processing (NLP) tasks, for example, Information Extraction and Machine Translation. In particular, the data-driven approaches have made great progress during the past two decades (Kübler, McDonald, and Nivre 2009; Koo et al. 2010; Pitler, Kannan, and Marcus 2013; Chen and Manning 2014; Weiss et al. 2015; Dozat and Manning 2016). Various practical parsing systems have been built, not only for English but also for a large number of typologically different languages, for example, Arabic, Basque, Catalan, Chinese, Czech, Greek, Hungarian, Italian, and Turkish (Nivre et al. 2007).

Previous work on dependency parsing mainly focused on structures that can be represented in terms of directed bilexical dependency trees. Although tree-shaped graphs have several desirable properties from the computational perspective, they do not work well for coordinations, long-range dependencies involved in raising, control, as well as extraction, and many other complicated linguistic phenomena that go beyond the surface syntax. Some well-established and leading linguistic theories, such as Word Grammar (Hudson 1990) and Meaning-Text Theory (Mel'čuk 2001), argue that more general dependency graphs are necessary to represent a variety of syntactic or semantic phenomena.

In this article, we are concerned with parsing Chinese sentences to an enriched deep dependency representation, which marks up a rich set of GRs to specify a linguistically-rich syntactic analysis. Different from the popular *surface*[1] tree-based dependency representation, our GR annotations are represented as general directed graphs that express not only local but also various long-distance dependencies (see Figure 1 for example). To enhance the tree-shaped dependency representation, we borrow the key ideas underlying Lexical Function Grammar (LFG) (Bresnan and Kaplan [1982], Dalrymple [2001]), a well-defined and widely-applied linguistic grammar formalism. In particular, GRs

1 In this work, we arguably take the dependency tree representation as a surface-oriented syntactic analysis.

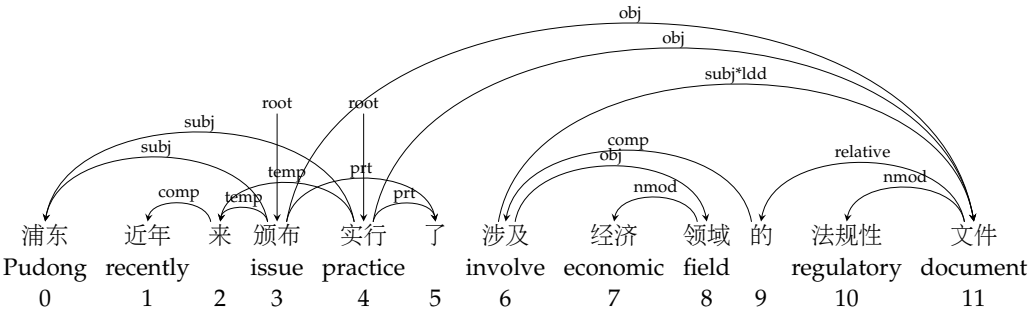


Figure 1
An example: *Pudong recently enacted regulatory documents involving the economic field*. The symbol "*ldd" indicates long-distance dependencies; "subj*ldd" between the word "涉及/involve" and the word "文件/documents" represents a long-range subject-predicate relation. The arguments and adjuncts of the coordinated verbs, namely "颁布/issue" and "实行/practice," are the separately yet distributively linked two heads.

can be viewed as the lexicalized dependency backbone of an LFG analysis that provides general linguistic insights.

To acquire a high-quality GR corpus, we propose a linguistically-motivated algorithm to translate a Government and Binding theory (GB) (Chomsky [1981], Carnie [2007]) grounded phrase structure treebank, namely, Chinese TreeBank (CTB) (Xue et al. [2005]) to a deep dependency bank where the GRs are explicitly represented. A manual evaluation highlights the reliability of our linguistically-motivated GR extraction algorithm: The overall dependency-based precision and recall are 99.17% and 98.87%. These numbers are calculated based on 209 sentences that are randomly selected and manually corrected. The automatically-converted corpus would be of use for a wide variety of NLP tasks, for example, parser training and cross-formalism parser evaluation.

By means of the newly created corpus, we study data-driven models for deep dependency parsing. The key challenge in building GR graphs is the range of relational flexibility. Different from the surface syntax, GR graphs are not constrained to trees. However, the tree structure is a fundamental consideration of the design for the majority of existing parsing algorithms. To deal with this problem, we propose **graph merging**, an innovative perspective for building flexible representations. The basic idea is to decompose a GR graph into several subgraphs, each of which captures most but not necessarily the complete information. On the one hand, each subgraph is *simple* enough to allow efficient construction. On the other hand, the combination of all subgraphs enables the whole target GR structure to be produced.

Based on this design, a practical parsing system is developed with a two-step architecture. In the first step, different graph-based models are applied to assign scores to individual arcs and various tuples of arcs. Each individual model is trained to target a unique type of subgraph. In the second step, a Lagrangian Relaxation-based joint decoder is applied to efficiently produce globally optimal GR graphs according to all graph-based models.

There are two main problems in the graph merging perspective. First, how do we decompose a complex graph into simple subgraphs in a principled way? To deal with this problem, we propose modeling the graph decomposition problem as a constrained optimization problem and leverage appropriate objective functions as well as constraints to reflect essential structure-specific properties of the syntactically-motivated

GR graphs. In particular, we consider the reachability property: In a given GR graph, every node is reachable from the same unique root. This property ensures that a GR graph can be successfully decomposed into a limited number of forests, which in turn can be accurately and efficiently built via tree parsing. Secondly, how can we merge subgraphs into one coherent structure in a principled way? The problem of finding an optimal graph that consistently combines the subgraphs obtained through individual models is non-trivial. We also treat this problem as an optimization problem and employ Lagrangian Relaxation to solve the problem.

Motivated by the recent successes of transition-based approaches to tree parsing, we also explore the transition-based models for building deep dependency structures. We utilize a list-based transition system that uses open lists for organizing partially processed tokens. This transition system is sound and complete with respect to directed graphs without self-loop. With reference to this system, we implement a data-driven parser with a neural classifier based on Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber [1997]). To improve the parsing quality, we look further into the impact of dynamic oracle and beam search on training and/or decoding.

Experiments on CTB 6.0 are conducted to profile the two types of parsers. Evaluation gauges how successful GR parsing for Chinese can be by applying data-driven models. Detailed analysis reveal some important factors that may possibly boost the performance. In particular, the following non-obvious facts should be highlighted:

1. Both the graph merging and transition-based parsers produce high-quality GR analysis with respect to dependency matching, although they do not use any phrase structure information.
2. When gold-standard POS tags are available, the graph merging model obtains a labeled f-score of 84.57 on the test set when coupled with a global linear scorer, and 86.17 when coupled with a neural model. Empirical analysis indicates the effectiveness of the proposed graph decomposition and merging methods.
3. The transition-based parser can be trained with either the dynamic oracle or the beam search method. According to our implementations, the dynamic oracle method performs better. Coupled with dynamic oracle, the transition-based parser reaches a labeled f-score of 85.51 when gold POS tags are utilized.
4. Gold-standard POS tags play a very important role in achieving the above accuracies. However, the automatic tagging quality of state-of-the-art Chinese POS taggers are still far from satisfactory. To deal with this problem, we apply ELMo (Peters et al. 2018), a contextualized word embedding producer, to obtain adequate lexical information. Experiments show that ELMo is extremely effective in harvesting lexical knowledge from raw texts. With the help of the ELMo vectors but not any POS tagging results, the graph merging parser achieves a labeled f-score of 84.90. This is a realistic set-up to evaluate how accurate the GR parsers could be for real-world Chinese Language Processing applications.

The current study expands on the earlier and preliminary results, which have been published in Sun et al. (2014) and Sun, Du, and Wan (2017). The new findings and contributions in this submission include: (1) a detailed comparison between our

GR analysis and other syntactic/semantic analyses, including Universal Dependency, Semantic Role Labeling, and LFG's f-structure, (2) the design of a new neural graph merging parser, (3) the design of a new neural transition-based parser, and (4) more comprehensive evaluations and analyses of the neural parsing models.

The implementation of the corpus conversion algorithm, a corpus visualization tool, and a set of manually labeled sentences are available at `http://www.aclweb.org/anthology/attachments/P/P14/P14-1042.Software.zip`. The implementation of the two neural parsers is available at `https://github.com/pkucoli/omg`.

2. Background

2.1 Deep Linguistic Processing

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, including Combinatory Categorial Grammar (CCG) (Steedman [1996, 2000]), Lexical Functional Grammar (LFG) (Bresnan and Kaplan [1982], Dalrymple [2001]), Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag [1994]), and Tree-adjoining Grammar (TAG) (Joshi and Schabes [1997]). These theories provide not only the description of the syntactic structures, but also the ways in which meanings are composed, and thus parsing in these formalisms provides an elegant way to simultaneously obtain both syntactic and semantic analyses, generating valuable and richer linguistic information. Deep linguistic annotators and processors are strongly demanded in NLP applications, such as machine translation (Oepen et al. 2007; Wu, Matsuzaki, and Tsujii 2010) and text mining (Miyao et al. 2008).

In general, derivations based on deep grammar formalisms may provide a wider variety of syntactic and semantic dependencies in more explicit details. Deep parses arguably contain the most linguistically satisfactory account of the deep dependencies inherent in many complicated linguistic phenomena, for example, coordination and extraction. Among the grammatical models, LFG and HPSG encode grammatical functions directly, and they are adequate for generating predicate–argument analyses (King et al. 2003; Flickinger, Zhang, and Kordoni 2012). In terms of CCG, Hockenmaier and Steedman (2007) introduced a co-indexing method to extract *(predicate, argument)* pairs from CCG derivations; Clark and Curran (2007a) also utilized a similar method to derive LFG-style grammatical relations. These bilexical dependencies can be used to approximate the corresponding semantic structures, such as logic forms in Minimal Recursion Semantics (Copestake et al. 2005).

In recent years, considerable progress has been made in deep linguistic processing for realistic texts. Traditionally, deep linguistic processing has been concerned with grammar development for parsing and generation. During the last decade, techniques for treebank-oriented grammar development (Cahill et al. 2002; Miyao, Ninomiya, and Tsujii 2005; Hockenmaier and Steedman 2007) and statistical deep parsing (Clark and Curran 2007b; Miyao and Tsujii 2008) have been well studied for English. Statistical models that are estimated on large-scale treebanks play an essential role in boosting the parsing performance.

It is generally believed that the representation format for parser outputs may greatly affect its impact on applications. Predicate–argument structures extracted from deep parses have been shown very helpful for NLP tasks such as information extraction (Miyao et al. 2008; Yakushiji et al. 2005). Partly due to their importance in information processing, deep dependencies are the de facto standard to evaluate deep parsers (Kaplan et al.

2004; Briscoe and Carroll 2006; Clark and Curran 2007a; Bender et al. 2011), including C&C[2] and Enju.[3] If the interpretable predicate–argument structures are so valuable, then the question arises: Why cannot we directly build deep dependency graphs?

2.2 Data-Driven Dependency Parsing

Data-driven, grammar-free dependency parsing has received an increasing amount of attention in the past decade. Such approaches, for example, transition-based (Yamada and Matsumoto 2003; Nivre 2008) and graph-based (McDonald et al. 2005; McDonald, Crammer, and Pereira 2005) models have attracted the most attention in dependency parsing in recent works. Transition-based parsers utilize transition systems to derive dependency trees together with treebank-induced statistical models for predicting transitions. This approach was pioneered by Yamada and Matsumoto (2003) and Nivre, Hall, and Nilsson (2004). Specifically, deep learning techniques have been successfully applied to enhance the parsing accuracy (Chen and Manning 2014; Weiss et al. 2015; Andor et al. 2016; Kiperwasser and Goldberg 2016; Dozat and Manning 2016).

Chen and Manning (2014) made the first successful attempt at incorporating deep learning into a transition-based dependency parser. A number of other researchers have attempted to address some limitations of their parser by augmenting it with additional complexity: Later it was enhanced with a more principled decoding algorithm, namely beam search, as well as a conditional random field loss objective function (Weiss et al. 2015; Watanabe and Sumita 2015; Zhou et al. 2015; Andor et al. 2016).

A graph-based system explicitly parameterizes models over substructures of a dependency tree, and formulates parsing as a Maximum Spanning Tree problem (McDonald et al. 2005). Similar to the transition-based approach, it also utilizes treebank annotations to train strong statistical models to assign scores to word pairs. Many researchers focus on developing principled decoding algorithms to resolve the Maximum Spanning Tree problem involved, for both projective (McDonald and Pereira 2006; Koo and Collins 2010) and non-projective structures (Koo et al. 2010; Kuhlmann 2010; Pitler, Kannan, and Marcus 2013). Deep learning has also been proved to be powerful for disambiguation and remained a hot topic in recent research (Kiperwasser and Goldberg 2016; Dozat and Manning 2016).

Kiperwasser and Goldberg (2016) proposed a simple yet effective architecture to implement neural dependency parsers. In particular, a bidirectional-LSTM (Bi-LSTM) is utilized as a powerful *feature extractor* to assist a dependency parser. Mainstream data-driven dependency parsers, including both transition- and graph-based ones, can apply useful word vectors provided by a Bi-LSTM to calculate scores. Following Kiperwasser and Goldberg (2016)'s experience, we implemented such a parser to evaluate the impact of empty categories on surface parsing.

Most research concentrated on surface syntactic structures, and the majority of existing approaches are limited to producing only trees. We notice several exceptions. Sagae and Tsujii (2008) and Henderson et al. (2013) individually introduced two transition systems that can generate specific graphs rather than trees. Unfortunately, the two transition systems only cover 51.0% and 76.5%, respectively, of graphs in our data, highlighting the high complexity of our graph representation and the inadequacy of existing transition systems. McDonald and Pereira (2006) presented a graph-based

2 http://svn.ask.it.usyd.edu.au/trac/candc/
3 http://kmcs.nii.ac.jp/enju/

parser that can generate dependency graphs in which a word may depend on multiple heads. Encouraged by their work, we study new data-driven models to build deep dependency structures for Mandarin Chinese.

2.3 Initial Research on Chinese Deep Linguistic Processing

In the last few years, study on deep linguistic processing for Chinese has been initialized. Treebank annotation for individual formalisms is prohibitively expensive. To quickly construct deep annotations, corpus-driven grammar engineering has been developed. Phrase structure trees in CTB have been semi-automatically converted to deep derivations in the CCG (Tse and Curran 2010), LFG (Guo, van Genabith, and Wang 2007), and HPSG (Yu et al. 2010) formalisms. To semi-automatically extract a large-scale HPSG grammar, Yu et al. (2010) defined a skeleton, including the structure of sign, grammatical principles, and schemata, based on which, the CTB trees are converted into HPSG-style trees. The treebank conversion under the CCG formalism is relatively easier. Tse and Curran (2010) followed the method proposed for English Penn Treebank (Hockenmaier and Steedman 2007) to process CTB. The main steps include (1) distinguishing head, argument, and adjunct, (2) binarizing CTB trees, and (3) redefining syntactic categories. To more robustly construct f-structure for CTB trees, Guo, van Genabith, and Wang (2007) proposed a dependency-based model, which extracts functional information from dependency trees.

With the use of converted fine-grained linguistic annotations, successful English deep parsers, such as C&C (Clark and Curran 2007b) and Enju (Miyao and Tsujii 2008), have been evaluated on the Chinese annotations (Yu et al. 2011; Tse and Curran 2012). Although the discriminative learning architecture of both C&C and Enju parsers makes them relatively easy to be adapted to solve multilingual parsing, their performance on Chinese sentences is far from satisfactory. Yu et al. (2011) and Tse and Curran (2012) analyze the challenges and difficulties in Chinese deep parsing. In particular, some language-specific properties account for a large number of errors.

3. Representing Deep Linguistic Information Using Dependency Graphs

In this section, we discuss the construction of the GR annotations. Basically, the annotations are automatically converted from a GB-grounded phrase structure treebank, namely CTB. Conceptually, this conversion is similar to the conversions from CTB structures to representations in deep grammar formalisms (Xia 2001; Guo, van Genabith, and Wang 2007; Tse and Curran 2010; Yu et al. 2010). However, our work is grounded in GB, which is the linguistic basis of the construction of CTB. We argue that this theoretical choice makes the conversion process more compatible with the original annotations and therefore more accurate. We use directed graphs to explicitly encode bilexical dependencies involved in coordination, raising/control constructions, extraction, topicalization, and many other complicated phenomena. Figure 2 shows the original CTB annotation of the sentence in Figure 1.

3.1 Linguistic Basis

GRs are encoded in different ways in different languages and languages may employ mixed or multiple strategies for grammatical function encoding. In some languages, for example, Turkish, grammatical function is encoded by means of morphological marking, and there may be no uniform position where a particular grammatical function must

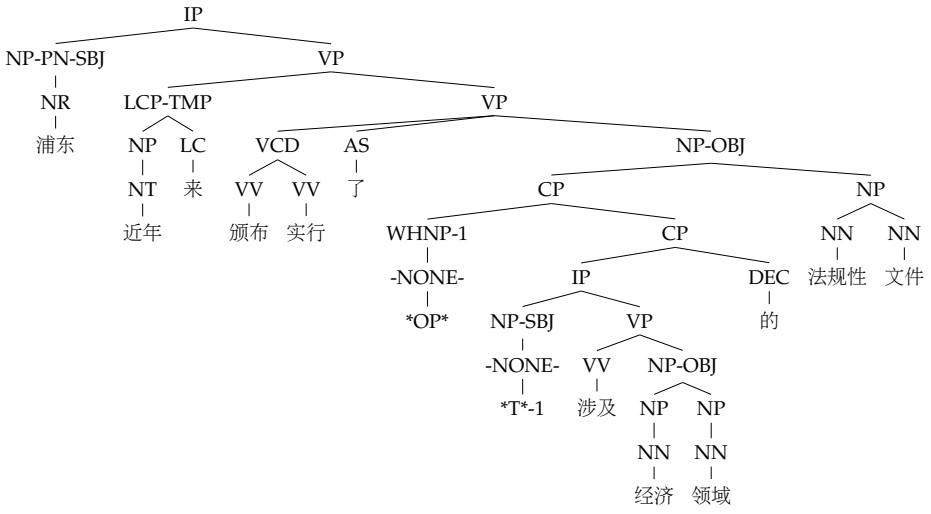


Figure 2
The original CTB annotation of the running example.

appear. In highly *configurational* languages, for example, Chinese, however, the grammatical function of a phrase is heavily determined by its constituent structure position. Dominant Chomskyan theories, including GB, have defined GRs as configurations at phrase structures. Following this principle, CTB groups words into constituents through the use of a limited set of fundamental grammatical functions. For example, one bracket represents only one grammatical relation in CTB, providing phrase structure annotations with specifications of particular grammatical functions. Transformational grammar utilizes empty categories to represent long-distance dependencies. In CTB, traces are provided by relating displaced linguistic material to where it should be interpreted semantically. By exploiting configurational information, traces, and functional tag annotations, GR information can hopefully be derived from CTB trees with high accuracy.

3.2 The Extraction Algorithm

Our treebank conversion algorithm borrows key insights from LFG. LFG posits two levels of representation: c(onstituent)-structure and f(unctional)-structure minimally. C-structure is represented by phrase structure trees, and captures surface syntactic configurations such as word order, whereas f-structure encodes grammatical functions. It is easy to extract a dependency backbone that approximates basic predicate–argument–adjunct structures from f-structures. The construction of the widely used PARC DepBank (King et al. 2003) is a good example.

LFG relates c-structure and f-structure through f-structure annotations, which compositionally map every constituent to a corresponding f-structure. Borrowing this key idea, we translate CTB trees to dependency graphs by first augmenting each constituency with f-structure annotations, then propagating the head words of the head or conjunct daughter(s) upwards to their parents, and finally creating a dependency graph. The details are presented step-by-step as follows:

3.2.1 Tapping Implicit Information. A systematic study to tap the implicit functional information of CTB has been introduced by Xue (2007). This gives us a very good

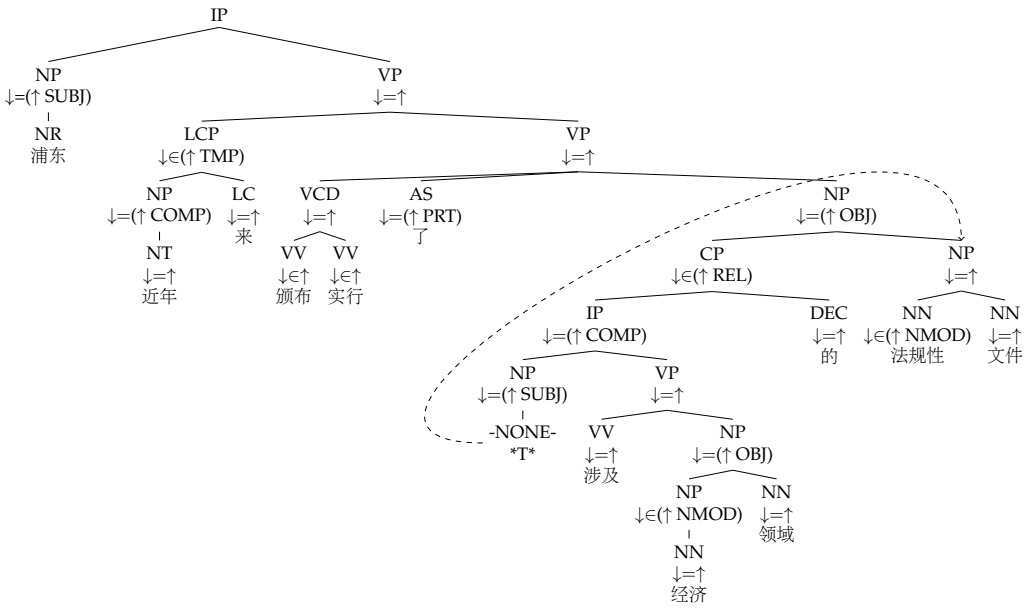


Figure 3
The original CTB annotation augmented with LFG-like functional annotations of the running example.

start to extract GRs. We slightly modify their method to enrich a CTB tree with f-structure annotations: Each node in a resulting tree is annotated with one and only one corresponding equation (see Figure 3 for an example). Comparing the original and enriched annotations, we can see that the functionality of this step is to explicitly represent and regulate grammatical functions[4] for every constituent. We enrich the CTB trees with function information by using the conversion tool[5] that generated the Chinese data sets for the CoNLL 2009 Shared Task on multilingual dependency parsing and semantic role labeling.

3.2.2 Beyond CTB Annotations: Tracing More. Natural languages do not always interpret linguistic materials locally. In order to obtain accurate and complete GR, predicate–argument, or logical form representations, a hallmark of deep grammars is that they usually involve a non-local dependency resolution mechanism. CTB trees utilize empty categories and co-indexed materials to represent long-distance dependencies. An empty category is a nominal element that does not have any phonological content and is therefore unpronounced. There are two main types of empty categories: null pronounce and trace, and each type can be further classified into different subcategories. Two kinds of anaphoric empty categories, that is, big PRO and trace, are annotated in CTB. Theoretically speaking, only trace is generated as the result of *movement* and therefore annotated with antecedents in CTB. We carefully checked the annotation and found that considerable amounts of antecedents are not labeled, and hence a large sum of important non-local information is missing. In addition, because the big PRO is also

4 See the Appendix for the list of major grammatical functions defined by our algorithm.
5 `www.cs.brandeis.edu/~clp/ctb/ctb.html`

anaphoric, it is possible to find co-indexed components sometimes. Such non-local information is also valuable in marking the dependency relation.

Beyond CTB annotations, we introduce a number of phrase structure patterns to extract more non-local dependencies. The method heavily leverages linguistic rules to exploit structural information. We take into account both theoretical assumptions and analyzing practices to enrich co-indexation information according to phrase structure patterns. In particular, we try to link an anaphoric empty category *e* with its c-commonders if no non-empty antecedent has already been co-indexed with *e*. Because the CTB is influenced deeply by the X-bar syntax, which highly regulates constituent analysis, the number of linguistic rules we have is quite modest. For the development of conversion rules, we used the first 9 files of CTB, which contain about 100 sentences. Readers can refer to the well-documented Perl script for details (see Figure 3 for an example). The noun phrase "法规性文件/regulatory documents" is related to the trace "*T*." This co-indexation is not labeled in the original annotation.

3.2.3 Passing Head Words and Linking Empty Categories. Based on an enriched tree, our algorithm propagates the head word of the head daughter upwards to their parents, linking co-indexed units, and finally creating a GR graph. The partial result after head word passing of the running example is shown in Figure 4. There are two differences in the head word passing between our GR extraction and a "normal" dependency tree extraction. First, the GR extraction procedure may pass multiple head words to their parent, especially in a coordination construction. Secondly, long-distance dependencies are created by linking empty categories and their co-indexed phrases.

The coding of long-distance dependency relations is particularly important for processing grammatical relations in Chinese. Compared to English, Chinese allows more flexibility in word order and a wider range of "gap-and-filler" relations. Among the common long-distance dependencies such as co-referential indexing, cleft-focus, prepositional phrase, and coordination, relativization in Chinese is of particular interest. Chinese relativization is structurally head-final, with the relative clause, marked by *de* 的 (the grammatical marker of nominal modifier), occurring before the head noun. The head noun may hold any kind of semantic relation with the proceeding relative

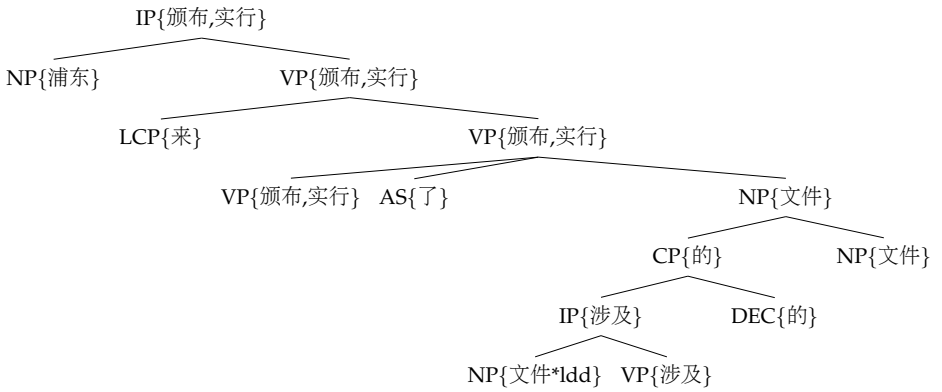


Figure 4
An example of lexicalized tree after head word upward passing. Only partial result is shown. The long-distance dependency between "涉及/involve" and "文件/document" is created through copying the dependent to a coindexed anaphoric empty category position.

Table 1
Manual evaluation of 209 sentences.

	Precision	Recall	F_1
Unlabeled	99.48	99.17	99.32
Labeled	99.17	98.87	99.02

clause. In other words, Chinese relative structure will have the "gap" occurring before the "filler" and there is little restriction on the semantic roles of the relativized head noun. Chinese-specific structural peculiarities may give rise to unexpected difficulties in sentence processing. Applying an augmented version of dependency notations, our system is able to handle such complicated issues in processing Chinese sentences.

3.3 Manual Evaluation

To have a precise understanding of whether our extraction algorithm works well, we have selected 20 files that contain 209 sentences in total for manual evaluation. Linguistic experts carefully examine the corresponding GR graphs derived by our extraction algorithm and correct all errors. In other words, a *gold-standard* GR annotation set is created. The measure for comparing two dependency graphs is precision/recall of GR tokens, which 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, while 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. To evaluate our GR parsing models that will be introduced later, we also report these metrics.

The overall performance is summarized in Table 1. We can see that the automatic GR extraction achieves relatively high performance. There are two sources of errors in treebank conversion: (1) inadequate conversion rules and (2) wrong or inconsistent original annotations. During the creation of the gold-standard corpus, we find that rule-based errors are mainly caused by complicated unbounded dependencies and the lack of internal structure for some phrases. Such problems are very hard to solve through rules only, if even possible, since original annotations do not provide sufficient information. The latter problem is more scattered and unpredictable, which requires manual correction.

3.4 Comparing to the De Facto LFG Analysis

Our extraction algorithm integrates the key idea underlying LFG and the practice of corpus annotation of CTB. Therefore, the structures obtained are highly comparable to Dalrymple (2001)'s de facto LFG analysis. Take the running sentence in Figure 1, for example. LFG separates syntactic analysis into two parallel structures: c-structure and f-structure. The c-structure provides basic analysis of the constituent structure. A *standard* c-structure for the running example is shown in Figure 5 and this analysis can be compared to the result—as shown in Figure 3—of the second step of our extraction algorithm. A number of functional schemata are introduced to augment the phrase structure analysis. By instantiating these schemata, we are able to generate the

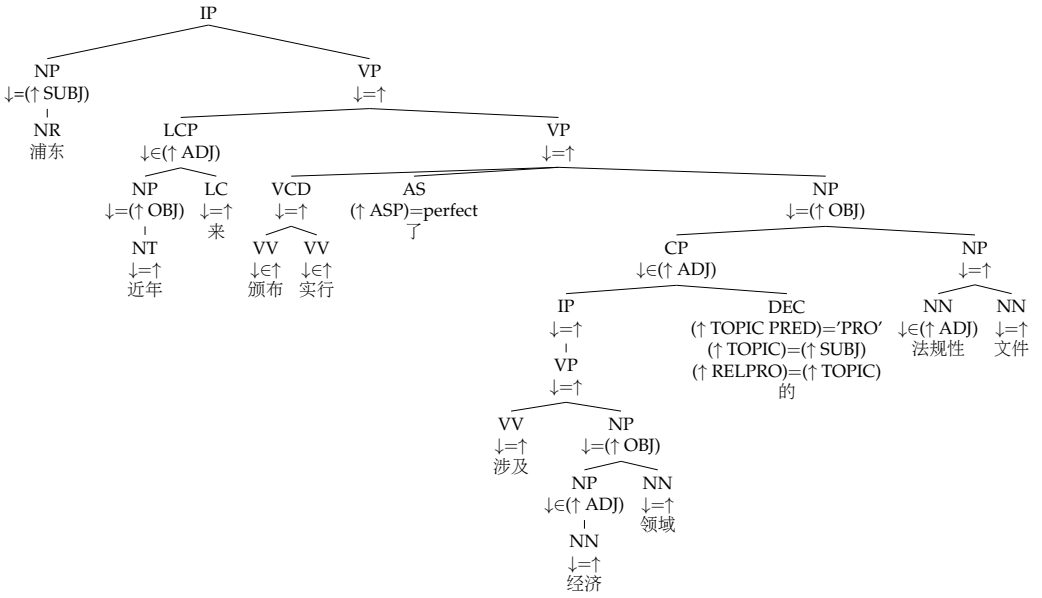


Figure 5
A de facto c-structure of the running example. The c-structure is augmented by a number of functional schemata.

corresponding functional description. Finally, we solve the simultaneous equations of these functional descriptions and then construct the minimal f-structure that satisfies them. Figure 6 shows the f-structure of the running sentence. According to the linguistic assumptions under LFG, this f-structure is more linguistically universal and has a tighter relationship with semantics.

Comparing the two trees in Figures 3 and 5, we can see that most functional schemata are similar. There are two minor differences between the LFG and our GR analysis.

1. LFG utilizes functional schemata that are assigned to phrase structures rather than empty categories to analyze a number of complex linguistic phenomena. Therefore, in the standard LFG c-structure tree, there is no "-NONE- *T*"-style terminals. Instead, specific constructions or function words take the responsibility. Our corpus is based on the original annotations of CTB, which is based on the GB theory. As a result, our grammatical function-augmented trees include unpronounced nodes, and there are not many functional schemata associated with constructions or function words.

2. The labels that indicate GRs, e.g. subject, objects, adjuncts, are slightly different. This is also due to the annotation practice of CTB. The labels in Figure 3 are designed to maximally reflect implicit functional information of the CTB annotations. Nevertheless, the two labeling systems are highly comparable. For example, the "TMP" label in Figure 3 corresponds to the "ADJ" label in Figure 5 because in most cases a temporal expression is taken as an adjunct.

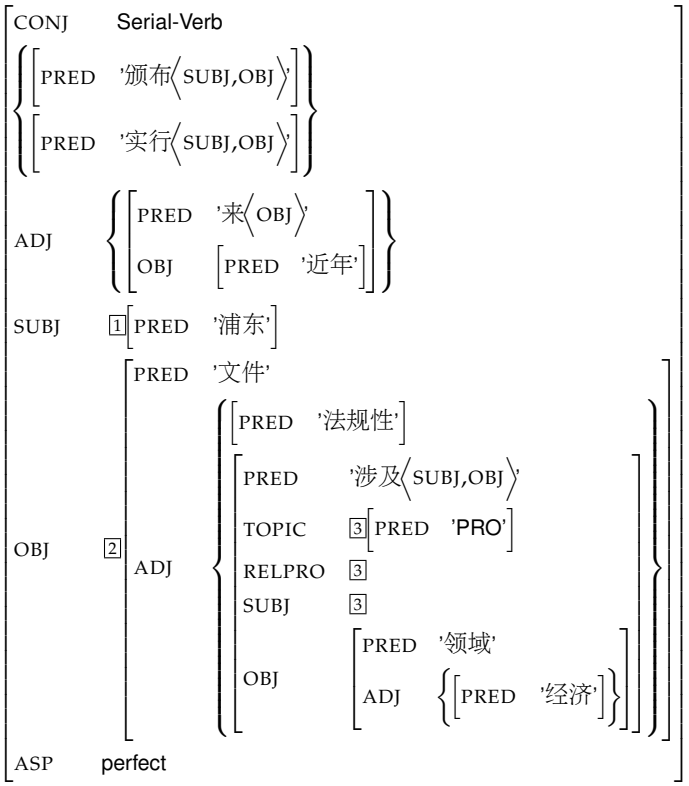


Figure 6
A de facto f-structure of the running example.

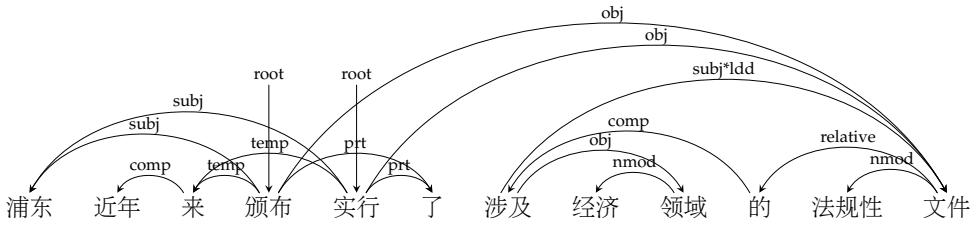
Some annotations may be different due to theoretical considerations. Take relative clause for example. The analysis in Figures 5 and 6 is based on the solution provided by Dalrymple (2001) and Bresnan (2001). The predicate–argument relation between "涉及/involve" and "文件/document" is not included because LFG treats this as an anaphoric problem. However, we argue that this relation is triggered by the function word *de* "的" as a marker of modification and is therefore put into our GR graph. Another motivation of this design is to let our GR graph represent more semantic information. We will continue to discuss this topic in the next subsection.

3.5 Comparing to Other Dependency Representations

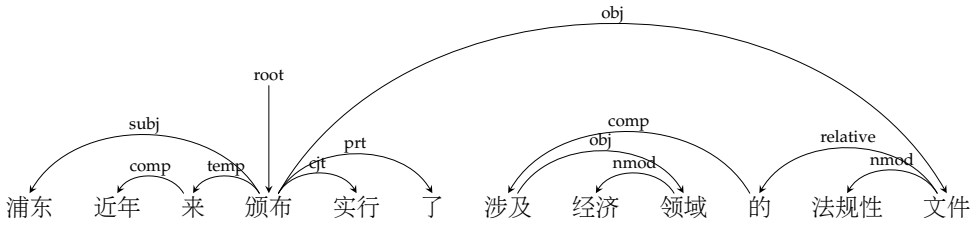
In this section, we consider the differences between our GR representation and other popular dependency-based syntactic and semantic analyses. Figure 7 visualizes four types of cross-representation annotations assigned to the sentence in Figure 1:

1. our GR graph,
2. the CTB-dependency tree,
3. the Universal Dependency, and
4. PropBank-style Semantic Role Labeling.

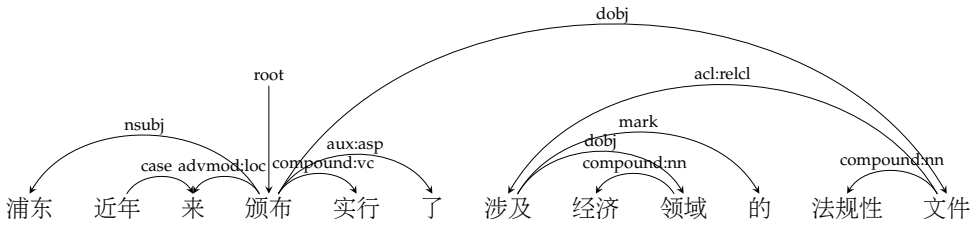
Two of the dependency representations are based on tree structures that are representative of Chinese Language Processing. Both are converted from CTB by applying heuristic rules. The CTB-dependency tree is the result of transformation introduced by Xue (2007), with a constituency-to-dependency transformation algorithm that explores the implicit functional information of CTB. This dependency corpus is used by the



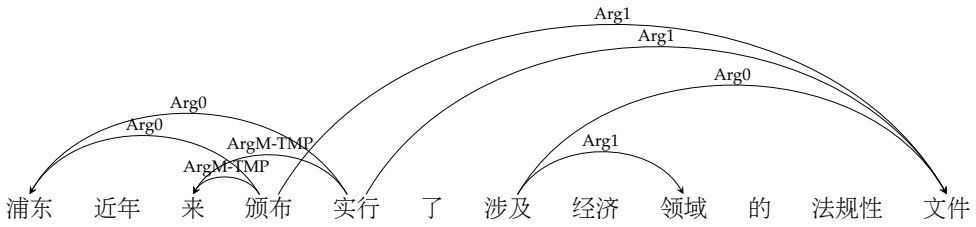
(a) The GR graph.



(b) The CTB-dependency.



(c) Universal dependency.



(d) PropBank-style Semantic role labeling.

Figure 7
Dependency representations in (a) our GR graph, (b) CTB-dependency, (c) Universal Dependency, and (d) PropBank-style semantic roles.

CoNLL 2009 shared task. Because this transformation algorithm is proposed by the developer and maintainer, we call it CTB-dependency. The Universal Dependency[6] corpus for Chinese is also based on CTB but with different constituency-to-dependency transformation rules. Both the language- and treebank-specific properties and comparability to resources of other languages are considered. The last representation is motivated by semantic processing and the dependencies reflect basic predicate–argument structures of verbs. The annotations are from Chinese PropBank (Xue and Palmer 2009), a sister resource to CTB. The head words are verbs or their normalizations, while the dependent words take semantic roles pertaining to the related verbs.

It can be clearly seen that compared to the tree-shaped dependency analysis, our GR representation represents much more syntactic information though more general graphs. In addition, the syntactic information of our analysis is more transparent with respect to (compositional) semantics. This property is highlighted by the structural parallelity between our GR graphs and the semantic role labeling results.

- Take, for example, the serial verb construction, a special type of coordination (颁布实行了 "issued-practiced"). According to the properties of distributive features, the shared subject, object, and aspect marker hold the same grammatical relations regarding both *conjuncts*. Limited by the single-head constraint, a dependency tree cannot represent all of these dependencies in an explicit way.
- Take the relative clause construction for another example (涉及经济领域的法规性文件 "the regulatory document that involves economic field"). There is a long-distance predicate–argument dependency between "涉及/involve" and "文件/document." The addition of this dependency to a tree will bring in a cycle for the CTB-dependency analysis. The Universal Dependency analysis includes this dependency, because its annotations are driven by not only syntax but also semantics. Nevertheless, the related label lacks the information that a subject is displaced.

3.6 A Quantitative Analysis

In the previous two subsections, we presented a phenomenon-by-phenomenon comparison to show the similarity and dissimilarity between our GR analysis and other syntactic/semantic analyses, for example, Universal Dependency, Semantic Role Labeling, and LFG's f-structure. In this subsection, we present a quantitative analysis based on overall statistics of the derived dependency corpus and a quantitative comparison with the CTB-dependency corpus. Table 2 shows how many dependencies are shared by both representations. The majority of grammatical relations involve local dependencies, and therefore the intersection of both representations are quite large. Nevertheless, a considerable number of dependencies of the GR representation do not appear in the tree representations. In principle, the GR representation removes semantically irrelevant dependencies, and thus it contains fewer arcs. Figure 8 summarizes the distribution of governable and non-governable GRs with respect to the tree and graph corpora.

6 http://universaldependencies.org

Table 2
The distribution of the unlabeled dependencies. The "Tree" and "Graph" rows indicate how many dependencies, that is, arcs, in total are included by the CTB-dependency and our graph representations. Sentences are selected from CTB 6.0 and have been used by the CoNLL 2009 shared task. "Graph∩Tree" means the dependencies in common; "Graph–Tree" means the dependencies that appear in the GR graphs but not the CTB-dependency trees; "Tree–Graph" means the dependencies that appear in the CTB-dependency trees but not the GR graphs.

	#Arc
Tree	731,833
Graph	669,910
Graph∩Tree	554,240
Graph–Tree	115,670
Tree–Graph	177,593

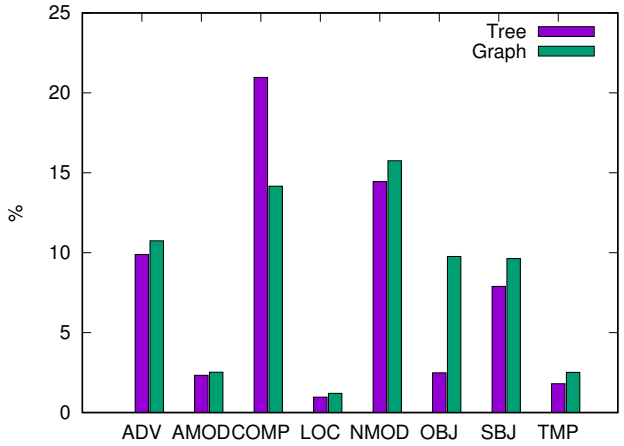


Figure 8
The distribution of the dependency relations.

4. Graph-Based Parsing via Graph Merging

4.1 The Proposal

The key proposal of this work is to construct a complex structure via constructing simple partial structures. Each partial structure is *simple* in the sense that it allows efficient construction. To construct each partial structure, we can employ mature parsing techniques. To obtain the final target output, it requires the total of all partial structures as they enable the whole target structure to be produced. This article aims to exemplify the above idea by designing a new parser for obtaining GR graphs. Take the GR graph in Figure 1 for example. It can be decomposed into two tree-like subgraphs, as shown in Figure 9. If we can parse the sentence into subgraphs and combine them in a principled way, we are able to obtain the original GR graph.

Under this premise, we need to develop a principled method to decompose a complex structure into simple structures, which allows us to generate data to train simple solvers. We also need to develop a principled method to integrate partial structures,

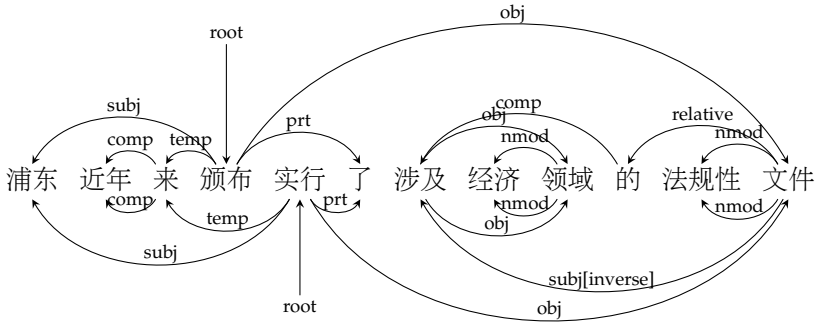


Figure 9
A graph decomposition for the GR graph in Figure 1. The two subgraphs are shown on two sides of the sentence, respectively. The subgraph on the upper side of the sentence is exactly a well-formed tree, while the one on the lower side is slightly different. The edge from the word "文件/document" to "涉及/involve" is tagged "[inverse]" to indicate that the direction of the edge in the subgraph is in fact opposite to that in the original graph.

which allows us to produce coherent structures as outputs. The techniques we developed to solve the two problems are demonstrated in the following sections.

4.2 Decomposing GR Graphs

4.2.1 Graph Decomposition as Optimization. Given a sentence $s = w_0 w_1 w_2 \cdots w_n$ of length n (where w_0 denotes the virtual root), a vector $\boldsymbol{y}$ of length $n(n+1)$ is used to denote a graph on the sentence. Indices i and j are then assigned to index the elements in the vector, $\boldsymbol{y}(i,j) \in \{0,1\}$, denoting whether there is an arc from w_i to w_j ($0 \leq i \leq n$, $1 \leq j \leq n$).

Given a graph $\boldsymbol{y}$, there may be m subgraphs $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m$, each of which belongs to a specific class of graphs $\mathcal{G}_k$ ($k = 1, 2, \cdots, m$). Each class should allow efficient construction. For example, we may need a subgraph to be a tree or a non-crossing dependency graph. The combination of all $\boldsymbol{y}_k$ gives enough information to construct $\boldsymbol{y}$. Furthermore, the graph decomposition procedure is utilized to generate training data for building submodels. Therefore, we hope each subgraph $\boldsymbol{y}_k$ is informative enough to train a good scoring model. To do so, for each $\boldsymbol{y}_k$, we define a score function s_k that indicates the "goodness" of $\boldsymbol{y}_k$. Integrating all ideas, we can formalize graph decomposition as an optimization problem,

$$\begin{aligned} \text{max.} \ & \textstyle\sum_k s_k(\boldsymbol{y}_k) \\ \text{s.t.} \ & \boldsymbol{y}_i \text{ belongs to } \mathcal{G}_i \\ & \textstyle\sum_k \boldsymbol{y}_k(i,j) \geq \boldsymbol{y}(i,j), \forall i,j \end{aligned}$$

The last condition in this optimization problem ensures that all edges in $\boldsymbol{y}$ appear at least in one subgraph.

For a specific graph decomposition task, we should define good score functions s_k and graph classes $\mathcal{G}_k$, according to key properties of the target structure $\boldsymbol{y}$.

4.2.2 Decomposing GR Graphs into Tree-Like Subgraphs. One key property of GR graphs is their reachability: Every node is either reachable from a unique root or is, by itself,

an independent connected component. This property allows a GR graph to be decomposable into a limited number of *tree-like* subgraphs. By tree-like, we mean that if we treat a graph on a sentence as undirected, it is either a tree, or a subgraph of some tree on the sentence. The advantage of tree-like subgraphs is that they can be effectively built by adapting data-driven tree parsing techniques. Take the sentence in Figure 1, for example. For every word, there is at least one path linking the virtual root and the target word. Furthermore, we can decompose the graph into two tree-like subgraphs, as shown in Figure 9. In such decomposition, one subgraph is exactly a tree, and the other is very close to a tree.

In practice, we restrict the number of subgraphs to 3. The reasoning is that we use one tree to capture long distance information and the other two to capture coordination information.[7] In other words, we decompose each given graph $\boldsymbol{y}$ into three tree-like subgraphs $\boldsymbol{g}_1, \boldsymbol{g}_2$, and $\boldsymbol{g}_3$ for each subgraph to carry important information of the graph as well as cover all edges in $\boldsymbol{y}$. The optimization problem can be written as

$$\begin{aligned} &\text{max. } s_1(\boldsymbol{g}_1) + s_2(\boldsymbol{g}_2) + s_3(\boldsymbol{g}_3) \\ &\text{s.t. } \quad \boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3 \text{ are tree-like} \\ &\qquad \boldsymbol{g}_1(i,j) + \boldsymbol{g}_2(i,j) + \boldsymbol{g}_3(i,j) \geq \boldsymbol{y}(i,j), \forall i,j \end{aligned}$$

Scoring a Subgraph. We score a subgraph in a first order *arc-factored* way, which first scores the edges separately and then adds up the scores. Formally, the score function is $s_k(\boldsymbol{g}) = \sum \omega_k(i,j)\boldsymbol{g}_k(i,j)$ $(k = 1, 2, 3)$ where $\omega_k(i,j)$ is the score of the edge from i to j. Under this score function, we can use the Maximum Spanning Tree (MST) algorithm (Chu and Liu 1965; Edmonds 1967; Eisner 1996) to decode the tree-like subgraph with the highest score.

After the score function is defined, extracting a subgraph from a GR graph works in the following way: We first assign heuristic weights $\omega_k(i,j)$ $(1 \leq i,j \leq n)$ to the potential edges between all the pairs of words, then compute a best projective tree $\boldsymbol{g}_k$ using the Eisner's Algorithm (Eisner 1996):

$$\boldsymbol{g}_k = \arg\max_{\boldsymbol{g}} s_k(\boldsymbol{g}) = \arg\max_{\boldsymbol{g}} \sum \omega_k(i,j)\boldsymbol{g}(i,j).$$

$\boldsymbol{g}_k$ is not exactly a subgraph of $\boldsymbol{y}$, because there may be some edges in the tree but not in the graph. To guarantee a meaningful subgraph of the original graph, we add labels to the edges in trees to encode necessary information. We label $\boldsymbol{g}_k(i,j)$ with the original label, if $\boldsymbol{y}(i,j) = 1$; with the original label appended by "∼R" if $\boldsymbol{y}(j,i) = 1$; with "None" else. With this labeling, we can have a function $t2g$ to transform the extracted trees into tree-like graphs. $t2g(\boldsymbol{g}_k)$ is not necessarily the same as the original graph $\boldsymbol{y}$, but it must be a subgraph of it.

Three Variations of Scoring. With different weight assignments, different trees can be extracted from a graph, obtaining different subgraphs. We devise three variations of

7 In this article, we employ projective parsers. The minimal number of sub-graphs is related to the pagenumber of GR graphs. The pagenumber of 90.96% GR graphs is smaller than or equal to 2, whereas the pagenumber of 98.18% GR graphs is at most 3. That means 3 projective trees are perhaps good enough to handle Chinese sentences, but 2 projective trees are not. Due to the empirical results in Table 3, using three projective trees can handle 99.55% GR arcs. Therefore, we think three is suitable for our problem.

weight assignment: ω_1, ω_2, and ω_3. Each of the ω's consists of two parts. One is shared by all, denoted by S, and the other is different from each other, denoted by V. Formally, $\omega_k(i,j) = S(i,j) + V_k(i,j)$ ($k = 1,2,3$ and $1 \leq i,j \leq n$).

Given a graph $\boldsymbol{y}$, S is defined as $S(i,j) = S_1(i,j) + S_2(i,j) + S_3(i,j) + S_4(i,j)$, where

$$S_1(i,j) = \begin{cases} w_1 \text{ if } \boldsymbol{y}(i,j) = 1 \text{ or } \boldsymbol{y}(j,i) = 1 \\ 0 \quad \text{else} \end{cases} \tag{1}$$

$$S_2(i,j) = \begin{cases} w_2 \text{ if } \boldsymbol{y}(i,j) = 1 \\ 0 \quad \text{else} \end{cases} \tag{2}$$

$$S_3(i,j) = w_3(n - |i-j|) \tag{3}$$

$$S_4(i,j) = w_4(n - l_p(i,j)) \tag{4}$$

In the definitions above, w_1, w_2, w_3, and w_4 are coefficients, satisfying $w_1 \gg w_2 \gg w_3$, and l_p is a function of i and j. $l_p(i,j)$ is the length of shortest path from i to j that either i is a child of an ancestor of j or j is a child of an ancestor of i. That is to say, the paths are in the form $i \leftarrow n_1 \leftarrow \cdots \leftarrow n_k \rightarrow j$ or $i \leftarrow n_1 \rightarrow \cdots \rightarrow n_k \rightarrow j$. If no such path exists, then $l_p(i,j) = n$. The reasoning behind the design is illustrated below.

S_1 indicates whether there is an edge between i and j, and it is meant for optimal effect;
S_2 indicates whether the edge is from i to j, and we want the edge with the correct direction more likely to be selected;
S_3 indicates the distance between i and j, and we like the edge with short distance because it is easier to predict;
S_4 indicates the length of certain types of path between i and j that reflects c-commanding relationships, and the coefficient remains to be tuned.

The score V is meant to capture different information from the GR graph. In GR graphs, we have an additional piece of information (as denoted as "*ldd" in Figure 1) for long-distance dependency edges. Moreover, we notice that conjunction is another important structure, which can be derived from the GR graph. Assume that we tag the edges relating to conjunctions with "*cjt." The three variation scores, that is, V_1, V_2, and V_3, reflect long distance and the conjunction information in different ways.

V_1. First for edges $\boldsymbol{y}(i,j)$ whose label is tagged with *ldd, we assign $V_1(i,j) = d$.[8] Whenever we come across a parent p with a set of conjunction children $cjt_1, cjt_2, \cdots, cjt_n$, we look for the rightmost child gc_{1r} of the leftmost child in conjunction cjt_1, and add d to each $V_1(p, cjt_1)$ and $V_1(cjt_1, gc_{1r})$. The edges in conjunction to which additional d's are added are shown in blue in Figure 10.

V_2. Different from V_1, for edges $\boldsymbol{y}(i,j)$ whose label is tagged with *ldd, we assign $V_2(j,i) = d$. Then for each conjunction structure with a parent p and a set of conjunction children $cjt_1, cjt_2, \cdots, cjt_n$, we find the leftmost child gc_{nl} of the rightmost child in

8 d is a coefficient to be tuned on validation data.

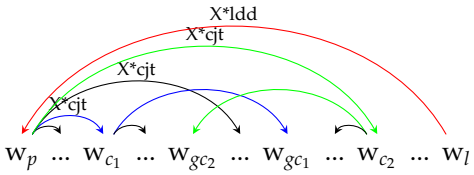


Figure 10
Examples to illustrate the additional weights.

conjunction cjt_n, and add d to each $V_2(p, cjt_n)$ and $V_2(cjt_n, gc_{nl})$. The concerned edges in conjunction are shown in green in Figure 10.

V_3. We do not assign d's to the edges with tag *ldd. For each conjunction with parent p and conjunction children $cjt_1, cjt_2, \cdots, cjt_n$, we add d to $V_3(p, cjt_1)$, $V_3(p, cjt_2)$, $\cdots$, and $V_3(p, cjt_n)$.

4.2.3 Lagrangian Relaxation with Approximation. As soon as we identify three trees $\boldsymbol{g}_1, \boldsymbol{g}_2$, and $\boldsymbol{g}_3$, there are three subgraphs $\boldsymbol{g}_1 = t2g(\boldsymbol{g}_1)$, $\boldsymbol{g}_2 = t2g(\boldsymbol{g}_2)$, and $\boldsymbol{g}_3 = t2g(\boldsymbol{g}_3)$. As stated above, each edge in a graph $\boldsymbol{y}$ needs to be covered by at least one subgraph, and the goal is to maximize the sum of the edge weights of all trees. Note that the inequality in the constrained optimization problem above can be replaced by a maximization, written as

$$\begin{aligned} \text{max. } & s_1(\boldsymbol{g}_1) + s_2(\boldsymbol{g}_2) + s_3(\boldsymbol{g}_3) \\ \text{s.t. } & \boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3 \text{ are trees} \\ & \max\{t2g(\boldsymbol{g}_1)(i,j), t2g(\boldsymbol{g}_2)(i,j), \\ & t2g(\boldsymbol{g}_3)(i,j)\} = \boldsymbol{y}(i,j), \forall i,j \end{aligned}$$

where $s_k(\boldsymbol{g}_k) = \sum \omega_k(i,j)\boldsymbol{g}_k(i,j)$

Let $\boldsymbol{g}_m = \max\{t2g(\boldsymbol{g}_1), t2g(\boldsymbol{g}_2), t2g(\boldsymbol{g}_3)\}$, and by $\max\{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3\}$ we mean to take the maximum of three vectors pointwisely. The Lagrangian of the problem is

$$\mathcal{L}(\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3; u) = s_1(\boldsymbol{g}_1) + s_2(\boldsymbol{g}_2) + s_3(\boldsymbol{g}_3) + u^\top(\boldsymbol{g}_m - \boldsymbol{y})$$

where u is the Lagrangian multiplier.

Then the dual is

$$\begin{aligned} \mathcal{L}(u) &= \max_{\boldsymbol{g}_1 \boldsymbol{g}_2 \boldsymbol{g}_3} \mathcal{L}(\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3; u) \\ &= \max_{\boldsymbol{g}_1}(s_1(\boldsymbol{g}_1) + \frac{1}{3}u^\top \boldsymbol{g}_m) + \max_{\boldsymbol{g}_2}(s_2(\boldsymbol{g}_2) + \frac{1}{3}u^\top \boldsymbol{g}_m) + \max_{\boldsymbol{g}_3}(s_3(\boldsymbol{g}_3) + \frac{1}{3}u^\top \boldsymbol{g}_m) - u^\top \boldsymbol{y} \end{aligned}$$

According to the duality principle, $\max_{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3; u} \min_u \mathcal{L}(\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3) = \min_u \mathcal{L}(u)$, we can find the optimal solution for the problem if we can find $\min_u \mathcal{L}(u)$. However, it is very hard to compute $\mathcal{L}(u)$, not to mention $\min_u \mathcal{L}(u)$. The challenge is that g_m in the three maximizations must be consistent.

Initialization: set $u^{(0)}$ to 0
for $k = 0$ to K:
  $\boldsymbol{g}_1 \leftarrow \arg\max_{\boldsymbol{g}_1} s_1(\boldsymbol{g}_1) + u^{(k)\top}\boldsymbol{g}_1$
  $\boldsymbol{g}_2 \leftarrow \arg\max_{\boldsymbol{g}_2} s_2(\boldsymbol{g}_2) + u^{(k)\top}\boldsymbol{g}_2$
  $\boldsymbol{g}_3 \leftarrow \arg\max_{\boldsymbol{g}_3} s_3(\boldsymbol{g}_3) + u^{(k)\top}\boldsymbol{g}_3$
  if $\max\{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3\} = y$ **then**
    return $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3$
  $u^{(k+1)} \leftarrow u_{(k)} - \alpha^{(k)}(\max\{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3\} - \boldsymbol{y})$
return $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3$

Figure 11
The Tree Extraction Algorithm.

The idea is to separate the overall maximization into three maximization problems by approximation. It is observed that g_1, g_2, and g_3 are very close to g_m, so we can approximate $\mathcal{L}(u)$ by

$$\begin{aligned}\mathcal{L}'(u) &= \max_{\boldsymbol{g}_1\boldsymbol{g}_2\boldsymbol{g}_3} \mathcal{L}(\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3; u) \\ &= \max_{\boldsymbol{g}_1}(s_1(\boldsymbol{g}_1) + \frac{1}{3}u^\top\boldsymbol{g}_1) + \max_{\boldsymbol{g}_2}(s_2(\boldsymbol{g}_2) + \frac{1}{3}u^\top\boldsymbol{g}_2) + \max_{\boldsymbol{g}_3}(s_3(\boldsymbol{g}_3) + \frac{1}{3}u^\top\boldsymbol{g}_3) - u^\top\boldsymbol{y}\end{aligned}$$

In this case, the three maximization problems can be decoded separately, and we can try to find the optimal u using the subgradient method.

4.2.4 The Algorithm. Figure 11 gives the tree decomposition algorithm, in which a subgradient method is used to identify $\min_u \mathcal{L}'(u)$ iteratively, and K is the maximum of iterations. In each iteration, we first compute $\boldsymbol{g}_1, \boldsymbol{g}_2$, and $\boldsymbol{g}_3$ to find $\mathcal{L}'(u)$, then update u until the graph is covered by the subgraphs. The coefficient $\frac{1}{3}$'s can be merged into the steps $\alpha^{(k)}$, so we omit them. The three separate problems $\boldsymbol{g}_k \leftarrow \arg\max_{\boldsymbol{g}_k} s_k(\boldsymbol{g}_k) + u^\top\boldsymbol{g}_k$ ($k = 1, 2, 3$) can be solved using Eisner's Algorithm (Eisner 1996), similar to solving $\arg\max_{\boldsymbol{g}_k} s_k(\boldsymbol{g}_k)$. Intuitively, the Lagrangian multiplier u in our algorithm can be regarded as additional weights for the score function. The update of u is to increase weights to the edges that are not covered by any tree-like subgraph, so it is more likely for them to be selected in the next iteration.

4.3 Graph Merging

As explained above, the extraction algorithm gives three classes of trees for each graph. The algorithm is applied to the graph training set to deliver three training tree sets. After that, three parsing models can be trained with the three tree sets. The parsers used in this study to train models and parse trees include Mate (Bohnet 2010), a second-order graph-based dependency parser, and our implementation of the first-order factorization model proposed in Kiperwasser and Goldberg (2016).

If the scores used by the three models are f_1,f_2, f_3, respectively, then the parsers can find trees with the highest scores for a sentence. That solves the following optimization problems: $\arg\max_{\boldsymbol{g}_1} f_1(\boldsymbol{g}_1)$, $\arg\max_{\boldsymbol{g}_2} f_2(\boldsymbol{g}_2)$, and $\arg\max_{\boldsymbol{g}_2} f_3(\boldsymbol{g}_3)$. We can

parse a given sentence with the three models, obtain three trees, and then transform them into subgraphs. We combine them together to obtain the graph parse of the sentence by putting all the edges in the three subgraphs together. That is to say, graph $\boldsymbol{y} = \max\{t2g(\boldsymbol{g}_1), t2g(\boldsymbol{g}_2), t2g(\boldsymbol{g}_3)\}$. This process is called **simple merging**.

4.3.1 Capturing the Hidden Consistency. However, the simple merging process lacks the consistency of extracting the three trees from the same graph, thus losing some important information. More specifically, when we decompose a graph into three subgraphs, some edges tend to appear in certain classes of subgraphs at the same time, and this information is lost in the simple merging process. It is more desirable to retain the co-occurrence relationship of the edges when doing parsing and merging. To retain the hidden consistency, instead of decoding the three models separately, we must do *joint* decoding.

In order to capture the hidden consistency, we add consistency tags to the labels of the extracted trees to represent the co-occurrence. The basic idea is to use additional tags to encode the relationship of the edges in the three trees. The tag set is $\mathcal{T} = \{0, 1, 2, 3, 4, 5, 6\}$. Given a tag $t \in \mathcal{T}$, $t\&1$, $t\&2$, $t\&4$, denote whether the edge is contained in $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3$, respectively, where the operator "&" is the bitwise AND operator. Since we do not need to consider the first bit of the tags of edges in $\boldsymbol{g}_1$, the second bit in $\boldsymbol{g}_2$, and the third bit in $\boldsymbol{g}_3$, we always assign 0 to these tags. For example, if $\boldsymbol{y}(i,j) = 1$, $\boldsymbol{g}_1(i,j) = 1, \boldsymbol{g}_2(j,i) = 1, \boldsymbol{g}_3(i,j) = 0$, and $t_3(j,i) = 0$, we tag $\boldsymbol{g}_1(i,j)$ as 2 and $\boldsymbol{g}_2(j,i)$ as 1.

When it comes to parsing, it is important to obtain labels with consistency information. Our goal is to guarantee that the tags in those edges of the parse trees for the same sentence are consistent throughout graph merging. Since the consistency tags emerge, we index the graph and tree vector representation using three indices for convenience. Thus, $\boldsymbol{g}(i,j,t)$ denotes whether there is an edge from word w_i to word w_j with tag t in graph g.

The joint decoding problem can be written as a constrained optimization problem as

$$
\begin{aligned}
&\max. \ f_1(\boldsymbol{g}_1) + f_2(\boldsymbol{g}_2) + f_3(\boldsymbol{g}_3) \\
&\text{s.t.} \quad \boldsymbol{g}'_1(i,j,2) + \boldsymbol{g}'_1(i,j,6) \le \textstyle\sum_t \boldsymbol{g}'_2(i,j,t) \\
&\qquad \boldsymbol{g}'_1(i,j,4) + \boldsymbol{g}'_1(i,j,6) \le \textstyle\sum_t \boldsymbol{g}'_3(i,j,t) \\
&\qquad \boldsymbol{g}'_2(i,j,1) + \boldsymbol{g}'_2(i,j,5) \le \textstyle\sum_t \boldsymbol{g}'_1(i,j,t) \\
&\qquad \boldsymbol{g}'_2(i,j,4) + \boldsymbol{g}'_2(i,j,5) \le \textstyle\sum_t \boldsymbol{g}'_3(i,j,t) \\
&\qquad \boldsymbol{g}'_3(i,j,1) + \boldsymbol{g}'_3(i,j,3) \le \textstyle\sum_t \boldsymbol{g}'_1(i,j,t) \\
&\qquad \boldsymbol{g}'_3(i,j,2) + \boldsymbol{g}'_3(i,j,3) \le \textstyle\sum_t \boldsymbol{g}'_2(i,j,t) \\
&\qquad \forall i,j
\end{aligned}
$$

where $\boldsymbol{g}'_k = t2g(\boldsymbol{g}_k)(k = 1, 2, 3)$.

The inequality constraints in the problem are the consistency constraints. Each of them gives the constraint between two classes of trees. For example, the first inequality says that an edge in $\boldsymbol{g}_1$ with tag $t\&2 \neq 0$ exists only when the same edge in $\boldsymbol{g}_2$ exists. If all of these constraints are satisfied, the subgraphs achieve the desired consistency.

4.3.2 Lagrangian Relaxation with Approximation. To solve the constrained optimization problem mentioned above, we perform some transformations and then apply the Lagrangian Relaxation to it with approximation.

Let $\boldsymbol{a}_{12}(i,j) = \boldsymbol{g}_1(i,j,2) + \boldsymbol{g}_1(i,j,6)$; then the first constraint can be written as an equity constraint

$$\boldsymbol{g}_1(:,:,2) + \boldsymbol{g}_1(:,:,6) = \boldsymbol{a}_{12}. * (\sum_t \boldsymbol{g}_2(:,:,t))$$

where ":" is to take out all the elements in the corresponding dimension, and ".∗" is to do multiplication point-wisely. Other inequality constraints can be rewritten in the same way. If we take $\boldsymbol{a}_{12}, \boldsymbol{a}_{13}, \cdots, \boldsymbol{a}_{32}$ as constants, then all the constraints are linear. Thus, the constraints can be written as

$$A_1\boldsymbol{g}_1 + A_2\boldsymbol{g}_2 + A_3\boldsymbol{g}_3 = \mathbf{0}$$

where A_1, A_2, and A_3 are matrices that can be constructed from $\boldsymbol{a}_{12}, \boldsymbol{a}_{13}, \cdots, \boldsymbol{a}_{32}$.

The Lagrangian of the optimization problem is

$$\mathcal{L}(\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3; u) = f_1(\boldsymbol{g}_1) + f_2(\boldsymbol{g}_2) + f_3(\boldsymbol{g}_3) + u^\top(A_1\boldsymbol{g}_1 + A_2\boldsymbol{g}_2 + A_3\boldsymbol{g}_3)$$

where u is the Lagrangian multiplier. Then the dual is

$$\begin{aligned}\mathcal{L}(u) &= \max_{\boldsymbol{g}_1\boldsymbol{g}_2\boldsymbol{g}_3} \mathcal{L}(\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3; u) \\ &= \max_{\boldsymbol{g}_1}(f_1(\boldsymbol{g}_1) + u^\top A_1\boldsymbol{g}_1) + \max_{\boldsymbol{g}_2}(f_2(\boldsymbol{g}_2) + u^\top A_2\boldsymbol{g}_2) + \max_{\boldsymbol{g}_3}(f_3(\boldsymbol{g}_3) + u^\top A_3\boldsymbol{g}_3)\end{aligned}$$

Again, we use the subgradient method to minimize $\mathcal{L}(u)$. During the deduction, $\boldsymbol{a}_{12}, \boldsymbol{a}_{13}, \cdots, \boldsymbol{a}_{32}$ are taken as constants, but unfortunately they are not. We propose an approximation for the $\boldsymbol{a}$'s in each iteration: Using the $\boldsymbol{a}$'s obtained in the previous iteration instead. It is a reasonable approximation given that the u's in two consecutive iterations are similar and so are the $\boldsymbol{a}$'s.

4.3.3 The Algorithm. The pseudocode of our algorithm is shown in Figure 12. It is well known that the score functions f_1, f_2, and f_3 each consists of scores that are first-order and higher. So they can be written as

$$f_k(\boldsymbol{g}) = s_k^{1st}(\boldsymbol{g}) + s_k^h(\boldsymbol{g})$$

```
1   Initialization: set u^(0), A1, A2, A3 to 0,
2   if k = 0 to K then
3     g1 ← arg max_g1 f1(g1) + u^(k)⊤ A1 g1
4     g2 ← arg max_g2 f2(g2) + u^(k)⊤ A2 g2
5     g3 ← arg max_g3 f3(g3) + u^(k)⊤ A3 g3
6     UPDATE A1, A2, A3
7     if A1 g1 + A2 g2 + A3 g3 = 0 then
8       return g1, g2, g3
9     u^(k+1) ← u^(k) − α^(k)(A1 g1 + A2 g2 + A3 g3)
10  return g1, g2, g3
```

Figure 12
The Joint Decoding Algorithm.

where $s_k^{1st}(\boldsymbol{g}) = \sum \omega_k(i,j)\boldsymbol{g}(i,j)$ $(k = 1,2,3)$. With this property, each individual problem $\boldsymbol{g}_k \leftarrow \arg\max_{\boldsymbol{g}_k} f_k(\boldsymbol{g}_k) + u^\top A_k \boldsymbol{g}_k$ can be decoded easily, with modifications to the first-order weights of the edges in the three models. Specifically, let $\mathbf{w}_k = u^\top A_k$, then we can modify the ω_k in s_k to ω'_k, such that $\omega'_k(i,j,t) = \omega_k(i,j,t) + \mathbf{w}_k(i,j,t) + \mathbf{w}_k(j,i,t)$.

The update of $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$ can be understood in an intuitive way. Consider the following situation: One of the constraints, say, the first one for edge $\boldsymbol{y}(i,j)$, is not satisfied, without loss of generality. We know $\boldsymbol{g}_1(i,j)$ is tagged to represent that $\boldsymbol{g}_2(i,j) = 1$, but it is not the case. So we increase the weight of that edge with all kinds of tags in $\boldsymbol{g}_2$, and decrease the weight of the edge with the tag representing $\boldsymbol{g}_2(i,j) = 1$ in $\boldsymbol{g}_1$. After the update of the weights, consistency is more likely to be achieved.

4.3.4 Labeled Parsing. For the sake of formal concision, we illustrate our algorithms omitting the labels. It is straightforward to extend the algorithms to labeled parsing. In the joint decoding algorithm, we just need to extend the weights $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$ for every label that appears in the three tree sets, and the algorithm can be deduced similarly.

4.4 Global Linear Model Based Scorer

A majority of dependency parsers have explored the framework of global linear models with encouraging success (Nivre, Hall, and Nilsson 2004; McDonald et al. 2005; McDonald, Crammer, and Pereira 2005; Torres Martins, Smith, and Xing 2009; Koo et al. 2010). The dependency parsing problem can be formalized as a structured linear model as follows:

$$\boldsymbol{g}^*(s) = \arg\max_{\boldsymbol{g}\in\mathcal{T}(s)} \textsc{Score}(s,\boldsymbol{g}) = \arg\max_{\boldsymbol{g}\in\mathcal{T}(s)} \theta^\top \Phi(s,\boldsymbol{g}) \tag{5}$$

In brief, given a sentence s, its parse $\boldsymbol{g}^*(s)$ is computed by searching for the highest-scored dependency graph in the set of compatible trees $\mathcal{T}(s)$. Scores, namely $\textsc{Score}(x,\boldsymbol{g})$, are assigned using a linear model where $\Phi(s,\boldsymbol{g})$ is a feature-vector representation of the event that tree $\boldsymbol{g}$ is the analysis of sentence s, and θ is parameter vector containing associated weights. In general, performing a direct maximization over the set $\mathcal{T}(s)$ is infeasible, and a common solution used in many parsing approaches is to introduce a part-wise factorization:

$$\textsc{Score}(s,\boldsymbol{g}) = \sum_{p\in\textsc{Part}(\boldsymbol{g})} \textsc{ScorePart}(s,p) \tag{6}$$

Considering linear models, we can define a factorization model as follows,

$$\theta^\top \Phi(s,\boldsymbol{g}) = \sum_{p\in\textsc{Part}(\boldsymbol{g})} \theta^\top \phi(s,p)$$

In the above, we have assumed that the output $\boldsymbol{g}$ can be factored into a set of parts p through a function $\textsc{Part}$, each of which represents a small substructure of $\boldsymbol{g}$. For example, $\boldsymbol{g}$ might be factored into the set of its component bilexical dependencies. Each part can be evaluated using a part-wise feature-vector mapping $\phi(x,p)$. The factorization is able to establish implicit independent relationships among parts, while keeping the search for the best result efficient.

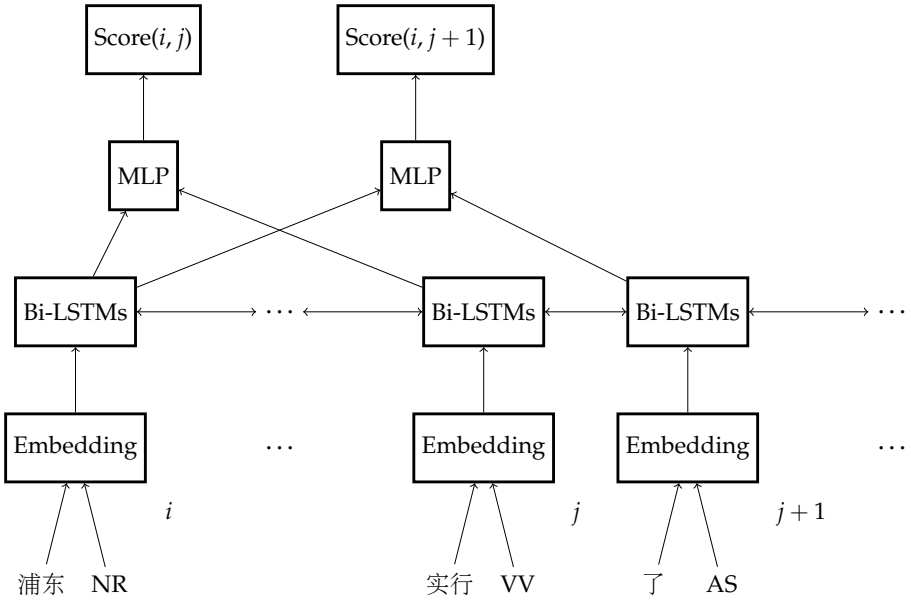


Figure 13
The architecture of the neural network model employed.

4.5 Bi-LSTM Based Scorer

In the above architecture, we can assign scores, namely SCORE(x,$\boldsymbol{g}$) in (5), using neural network models. A simple yet effective design is selected among a rich set of choices. Following Kiperwasser and Goldberg (2016)'s experience, we employ a bidirectional-LSTMs (Bi-LSTMs) based neural model to perform data-driven parsing. A vector is associated with each word or POS-tag to transform them into continuous and dense representations. The concatenation of word embedding and POS-tag embedding of each word in a specific sentence is used as the input of Bi-LSTMs to extract context-related feature vectors r_i. The two feature vectors of each word pair are scored with a nonlinear transformation.

$$\text{SCOREPART}(\boldsymbol{g}, i, j) = \boldsymbol{W}_2 \cdot \text{ReLU}\left(\boldsymbol{W}_{1,1} \cdot \boldsymbol{r}_i + \boldsymbol{W}_{1,2} \cdot \boldsymbol{r}_j + \boldsymbol{b}\right) \tag{7}$$

Figure 13 shows the architecture of this design.

We can see here the *local* score function explicitly utilizes the word positions of the head and the dependent. It is similar to first-order factorization as defined in the linear model. We use the first-order Eisner Algorithm (Eisner 1996) to get coherent projective subtrees.

5. Transition-Based Parsing

5.1 Background Notions

To precisely illustrate our transition-based parser, we employ traditional graph-theoretic notations to define a transition system. A dependency graph $G = (V, A)$ is a

labeled directed graph in the standard graph-theoretic sense and consists of nodes, V, and arcs, A, such that for sentence $x = w_0 w_1 \dots w_n$ and label set R the following holds:

- $V = \{0, 1, \dots, n\}$,
- $A \subseteq V \times V \times R$.

The vertex -1 denotes a virtual root. The arc set A represents the labeled dependency relations of the particular analysis G. Specifically, an arc $(w_i, w_j, r) \in A$ represents a dependency relation from head w_i to dependent w_j labeled with relation type r. A dependency graph G is thus a set of labeled dependency relations between the words of x.

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 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 $\beta = [0, \dots, n]$, and
4. $\mathcal{C}_t \subseteq \mathcal{C}$ is a set of terminal configurations.

Given a sentence $x = w_1, \dots, w_n$ and a graph $G = (V, A)$ on it, if there is a sequence of transitions $t_1, \dots, t_m$ and a sequence of configurations $c_0, \dots, c_m$ such that $c_0 = c_s(x)$, $t_i(c_{i-1}) = c_i (i = 1, \dots, m)$, $c_m \in \mathcal{C}_t$, 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 in c_i. In a typical transition-based parsing process, the input words are put into a queue and partially built structures are organized by one or more memory module(s). A set of transition actions are performed sequentially to consume words from the queue and update the partial parsing results, organized by the stack.

5.2 A List-Based Transition System

Nivre (2008) proposed a list-based algorithm to produce non-projective dependency trees. This algorithm essentially implements a very simple idea that is conceptually introduced by Covington (2001): making use of a list to store partially processed tokens. It is straightforward to use this strategy to handle any kind of directed graphs. In this work, we use such a system S_L. In fact, it is simpler to produce a graph as compared to a tree. The main difference between Nivre's (2008) tree-parsing system and S_L is that at each transition step, S_L does not need to check the multiheaded condition. This appears to be simpler and more efficient.

5.2.1 The System. In $S_L = (\mathcal{C}, T, c_s, \mathcal{C}_t)$, we take $\mathcal{C}$ to be the set of all quadruples $c = (\lambda, \lambda', \beta, A)$ where λ and λ' are lists of nodes. The initial configuration $c_s(x)$ is $([], [][1, \dots, n], , \{\})$, and a terminal configuration c_t is of the form $(\lambda, \lambda', [], A)$ (for any

Transitions	
LEFT-ARC$_l$	$(\lambda_1\|i, \lambda_2, j\|\beta, A) \Rightarrow (\lambda_1\|i, \lambda_2, j\|\beta, A \cup \{(j, l, i)\})$
RIGHT-ARC$_l$	$(\lambda_1\|i, \lambda_2, j\|\beta, A) \Rightarrow (\lambda_1\|i, \lambda_2, j\|\beta, A \cup \{(i, l, j)\})$
SHIFT	$(\lambda_1, \lambda_2, j\|\beta, A) \Rightarrow (\lambda_1.\lambda_2\|j, [], \beta, A)$
	(The operation $\lambda_1.\lambda_2$ is to concatenate the two lists)
NO-ARC	$(\lambda_1\|i, \lambda_2, \beta, A) \Rightarrow (\lambda_1, i\|\lambda_2, \beta, A)$
*SELF-ARC$_r$	$(\lambda\|i, \lambda', \beta) \Rightarrow (\lambda\|i, \lambda', \beta \cup (i, r, i))$

Figure 14
Transitions of the list-based system.

list and any arc set). T contains five types of transitions, shown in Figure 14, and is illustrated as follows:

- LEFT-ARC$_r$ updates a configuration by adding (j, r, i) to A where i is the top element of the list λ, j is the front of the buffer, and r is the dependency relation.
- RIGHT-ARC$_r$ updates a configuration similarly by adding (i, r, j) to A.
- SHIFT concatenates λ' to λ, clears λ', and then moves the front element of β to current left list.
- NO-ARC removes the right most node from λ and adds it onto the left most of λ'.

5.2.2 Theoretical Analysis. **Theorem 1**
S_L is sound and complete[9] with respect to the class of directed graphs without self-loop.

Proof 1
The soundness of S_L is relatively trivial. The completeness of S_L is obvious from the construction of the **oracle** sequence as follows: For each step on an initial configuration, we first construct all the arcs in $\bar{A}_{c_i}$ that link the nodes in λ_{c_i} to the front node of β_{c_i} by applying LEFT-ARC$_r$, RIGHT-ARC$_r$, and NO-ARC. If no other transition is allowed, SHIFT is applied. ■

5.2.3 Extension. It is easy to extend our system to generate arbitrary directed graphs by adding a new transition SELF-ARC:

- SELF-ARC adds an arc from the top element of λ to itself, but does not update any list nor the buffer.

9 The notations of soundness and completeness are adopted from Nivre (2008). Let S be a transition system for dependency parsing.

- S is sound for a class $\mathcal{G}$ of dependency graphs if and only if, for every sentence x and every transition sequence $c_0, \ldots, c_m$ for x in S, the parse $G_{c_m} \in \mathcal{G}$.
- S is complete for a class $\mathcal{G}$ of dependency graphs if and only if, for every sentence x and every dependency graph G_x for x in $\mathcal{G}$, there is a transition sequence $c_0, \ldots, c_m$ for x in S such that $G_{c_m} = G_x$.

Transition	Configuration			
	([-1],	[],	[0,...,11],	$\emptyset$)
SHIFT	([-1,0],	[],	[1,...,11],	$\emptyset$)
SHIFT	([-1,0,1],	[],	[2,...,11],	$\emptyset$)
LEFT-ARC$_{\text{COMP}}$	([-1,0,1],	[],	[2,...,11],	$A_1 = \{(2, 1, \text{COMP})\}$)
SHIFT	([-1,...,2],	[],	[3,...,12],	A_1)
LEFT-ARC$_{\text{TEMP}}$	([-1,...,2],	[],	[3,...,11],	$A_2 = A_1 \cup \{(3, 2, \text{TEMP})\}$)
NO-ARC	([-1,0,1],	[2],	[3,...,11],	A_2)
NO-ARC	([-1,0],	[1,2],	[3,...,11],	A_2)
LEFT-ARC$_{\text{SUBJ}}$	([-1,0],	[1,2],	[3,...,11],	$A_3 = A_2 \cup \{(3, 0, \text{SUBJ})\}$)
SHIFT	([-1,...,3],	[],	[4,...,11],	A_3)
NO-ARC	([-1,...,2],	[3],	[4,...,11],	A_3)
LEFT-ARC$_{\text{TEMP}}$	([-1,...,2],	[3],	[4,...,11],	$A_4 = A_3 \cup \{(4, 2, \text{TEMP})\}$)
NO-ARC$_{\text{SUBJ}}$	([-1,0,1],	[2,3],	[4,...,11],	$A_5 = A_4 \cup \{(4, 0, \text{SUBJ})\}$)
NO-ARC$_{\text{SUBJ}}$	([-1,0],	[1,2,3],	[4,...,11],	$A_5 = A_4 \cup \{(4, 0, \text{SUBJ})\}$)
LEFT-ARC$_{\text{SUBJ}}$	([-1,0],	[3],	[4,...,11],	$A_5 = A_4 \cup \{(4, 0, \text{SUBJ})\}$)
SHIFT	([-1,...,4],	[],	[5,...,11],	A_5)
RIGHT-ARC$_{\text{PRT}}$	([-1,...,4],	[],	[5,...,11],	$A_6 = A_5 \cup \{(4, 5, \text{PRT})\}$)
NO-ARC	([-1,...,3],	[4],	[5,...,11],	A_6)
RIGHT-ARC$_{\text{PRT}}$	([-1,...,3],	[4],	[5,...,11],	$A_7 = A_6 \cup \{(3, 5, \text{PRT})\}$)
SHIFT	([-1,...,5],	[],	[6,...,11],	A_7)

Figure 15
A prefix of the oracle transition sequence for the running example.

Linguistic dependencies usually exclude self-loop, and therefore the basic list-based system is satisfactory in most cases. We use the basic list-based system, namely S_L, as the core engine of our parser.

5.2.4 An Example. Figure 15 shows the first transitions needed to parse the running example of Figure 1. It can be seen, from this example, that the key step to produce crossing arcs is to temporarily move nodes that block non-adjacent nodes to the secondary memory module, namely λ'. Another key property of the oracle is building arcs as soon as possible to avoid further complication.

5.3 Bi-LSTM Based Scorer

The neural model, which acts as a classifier of actions in this transition system, is similar to previous neural models. The Bi-LSTMs play the same role, but the feature vectors of the front of the buffer and the top of list λ are used to assign scores for actions. The structure is shown in Figure 16.

$$\text{Scores} = \boldsymbol{W}_2 \cdot \text{ReLU}\left(\boldsymbol{W}_1 \cdot (\boldsymbol{r}_{\text{list}\lambda\text{top}} \oplus \boldsymbol{r}_{\text{bufferfront}}) + \boldsymbol{b}_1\right) + \boldsymbol{b}_2$$

Two search methods, that is, dynamic oracle (Goldberg and Nivre 2012) and beam search, can be used with this transition system.

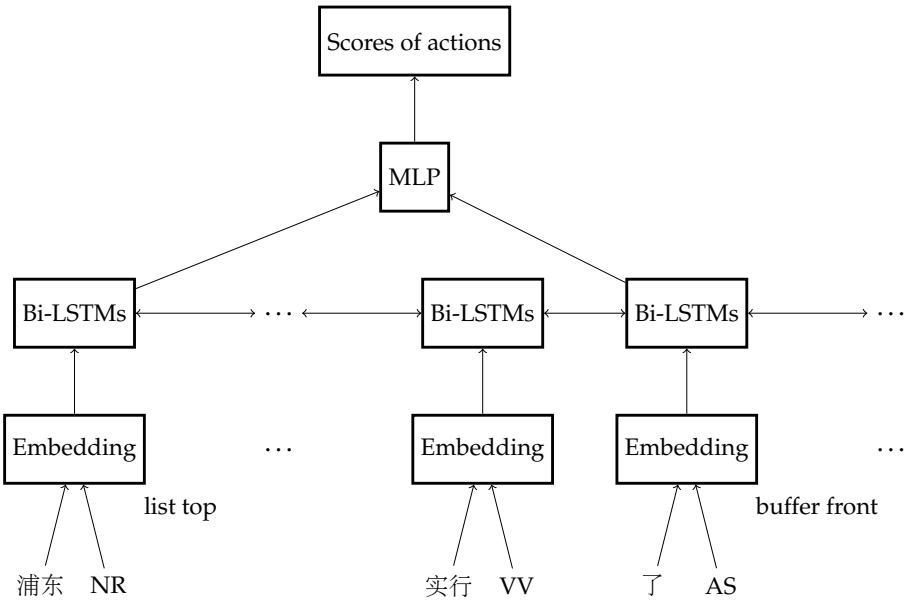


Figure 16
The neural network structure for parsing the running sentence. We select the top element of list λ and top front of the buffer as features.

6. Empirical Evaluation

6.1 Experimental Setup

CTB is a segmented, part-of-speech (POS) tagged, and fully bracketed corpus in the constituency formalism, and very popularly used to evaluate fundamental NLP tasks, including word segmentation (Sun and Xu 2011), POS tagging (Sun and Uszkoreit 2012), constituent parsing (Wang, Sagae, and Mitamura 2006; Zhang and Clark 2009), and dependency parsing (Zhang and Clark 2008; Huang and Sagae 2010; Li et al. 2011). This corpus was collected during different time periods from different sources with a diverse range of topics. We used CTB 6.0 and defined the training, development, and test sets according to the CoNLL 2009 shared task. Table 3 gives a summary of the data sets for experiments.

Evaluation on this benchmark data allows us to directly compare our parsers and other parsers in the literature, according to numeric performance. The measure for comparing two dependency graphs is precision/recall of bilexical dependencies, which 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, while unlabeled metrics (UP/UR) is the ratio regardless of l. F-score (UF/LF) is a harmonic mean of precision and recall. These measures correspond to attachment scores (LAS/UAS) in dependency tree parsing. To evaluate the ability to recover non-local dependencies, the recall ($\mathrm{UR_{NL}}$/$\mathrm{LR_{NL}}$) of such dependencies is reported. We also consider the correctness with respect to the whole graph and report unlabeled and labeled complete match (UCM/LCM).

Table 3
Data sets for experiments. Column "Training," "Development," and "Test" present the number of sentences/words/dependencies in the training, development, and test sets.

	Training	Development	Test
#Sentence	22,277	1,762	2,556
#Word	609,060	49,620	73,153
#Dependency	556,551	45,724	67,635

6.2 Main Results of the Graph-Based Parsing

6.2.1 Results of Graph Decomposition. Table 4 shows the results of graph decomposition on the training set. If we use simple decomposition, for example, directly extracting three trees from a graph, we get three subgraphs. On the training set, each of the subgraphs covers around 90% of edges and 30% of sentences. When merged together, they cover nearly 97% of edges and more than 70% of sentences. This indicates that the ability of a single tree is limited and that three trees can cover most of the edges. To achieve the best coverage, we need to tune the weights defined in Equations (1–4). This can be done on the development data. When we apply Lagrangian Relaxation to the decomposition process, both the edge coverage and the sentence coverage gain a great error reduction, indicating that Lagrangian Relaxation is very effective in the task of decomposition.

6.2.2 Main Results of Graph Merging. Table 5 shows the results of graph merging on the development set when tree parsers based on global linear models are applied. The three training sets of trees are from the decomposition with Lagrangian Relaxation and the models are trained from them. In both tables, simple merging (SM) refers to decoding the three trees for a sentence first, then combining them by putting all the edges together. As is shown, the merged graph achieves a higher f-score than other single models. With Lagrangian Relaxation, the performance scores of the merged graph and the three subgraphs are both improved, due to the capturing of the consistency information.

Table 4
Results of graph decomposition. *SD* is for Simple Decomposition and *LR* for Lagrangian Relaxation.

	Coverage	Edge	Sentence
SD	Subgraph1	85.52	28.73
	Subgraph2	88.42	28.36
	Subgraph3	90.40	34.37
	Merged	96.93	71.66
LR	Subgraph1	85.66	29.01
	Subgraph2	88.48	28.63
	Subgraph3	90.67	34.72
	Merged	**99.55**	**96.90**

Table 5
Accuracies of the graph merging–based parser on development set. *SM* is for Simple Merging, and *LR* for Lagrangian Relaxation. A second-order graph-based parser with a global linear disambiguation model is used for tree parsing.

		UP	UR	UF	UCM	LP	LR	LF	LCM
SM	Subgraph1	88.63	76.19	81.94	18.09	85.94	73.88	79.46	16.11
	Subgraph2	88.04	78.20	82.83	17.47	85.31	75.77	80.26	15.43
	Subgraph3	88.91	81.12	84.84	20.36	86.57	78.99	82.61	17.30
	Merged	83.23	88.45	85.76	22.97	80.59	85.64	83.04	19.29
LR	Subgraph1	89.76	77.48	83.17	18.60	87.17	75.25	80.77	16.39
	Subgraph2	89.30	79.18	83.93	18.66	86.68	76.85	81.47	16.56
	Subgraph3	89.42	81.55	85.31	20.53	87.09	79.43	83.08	17.81
	Merged	88.07	85.14	86.58	26.32	85.55	82.70	84.10	21.61

When we do simple merging, though the recall of each subgraph is much lower than its precision, it achieves the opposite effect of the merged graph. This is because the consistency between the three models is not required and the models tend to give diverse subgraph predictions. When we require high consistency between the three models, the precision and recall become comparable, and higher f-scores are achieved.

Table 6 shows the results based on the neural tree parsing model. Similar to Table 5, it is shown that the LR-based graph merging model is effective in improving individual tree parsers and thus obtains encouraging deep dependency parsing results.

6.3 Main Results of Transition-Based Parsing

Table 7 summarizes the parsing results obtained by the transition-based parser. We compare two strategies, namely, dynamic oracle and beam search, for improving training and decoding. Dynamic oracle is a very useful training strategy for the improvement of a transition-based parser, especially for neural models (Goldberg and Nivre 2012; Kiperwasser and Goldberg 2016). Beam search is a very effective search method that

Table 6
Accuracies of the graph merging–based parser on development set. A first-order graph-based parser with a neural disambiguation model is used for tree parsing.

		UP	UR	UF	UCM	LP	LR	LF	LCM
SM	Subgraph1	89.66	78.81	83.88	19.13	87.40	76.82	81.77	17.25
	Subgraph2	89.04	81.31	85.00	19.52	86.82	79.29	82.88	16.97
	Subgraph3	90.50	81.24	85.62	20.20	88.38	79.33	83.61	17.88
	Merged	84.01	90.70	87.23	23.72	81.70	88.20	84.83	20.60
LR	Subgraph1	91.18	79.90	85.17	20.72	88.80	77.82	82.95	18.39
	Subgraph2	90.45	82.29	86.17	20.72	88.16	80.20	83.99	18.16
	Subgraph3	91.29	81.91	86.35	20.66	89.11	79.95	84.28	17.93
	Merged	88.75	87.96	88.36	29.34	86.43	85.66	86.05	25.09

Table 7
Accuracies of the transition-based parser on development set.

	UP	UR	UF	UCM	LP	LR	LF	LCM
Dynamic oracle	89.59	85.50	87.49	24.74	87.49	83.49	85.44	21.45
Beam search	84.58	87.38	85.96	20.89	82.17	84.89	83.51	17.99

achieves excellent results for various NLP tasks, for example, Machine Translation. When coupled with linear models, beam search has shown to be a useful technique to improve both training and decoding (Zhang and Clark 2011b). However, in the particular neural parsing architecture employed here, beam search performs significantly worse than the dynamic oracle strategy. However, do note that beam search and structured learning may be very helpful for neural parsing models of other architectures (Weiss et al. 2015; Andor et al. 2016). Here, the beam size is set to 16.

6.4 Analysis

6.4.1 Precision vs. Recall. A noteworthy fact about the overall performance of the neural transition-based system is that the precision is promising but the recall is low. This difference is consistent with the result obtained by transition-based parsers with linear scoring models (Zhang et al. 2016), and the result obtained by a shift-reduce CCG parser (Zhang and Clark 2011a). The functor-argument dependencies generated by the CCG parser also has a relatively high precision but considerably low recall. To build NLP application, for example, information extraction, and systems upon GR parsing, such property merits attention. A good trade-off between the precision and the recall may have a great impact on final results.

The graph merging system coupled with neural tree parser performs quite differently. The precision and recall are quite harmonious. Figure 17 and Table 8 present detailed analyses. With respect to the dependency length, it is clear that the transition-based parser obtains higher precision, while the graph merging system parser obtains higher recall. With respect to the dependency relation, for most types, the precision of the transition-based parser is higher than its recall. Again, we can see a difference for

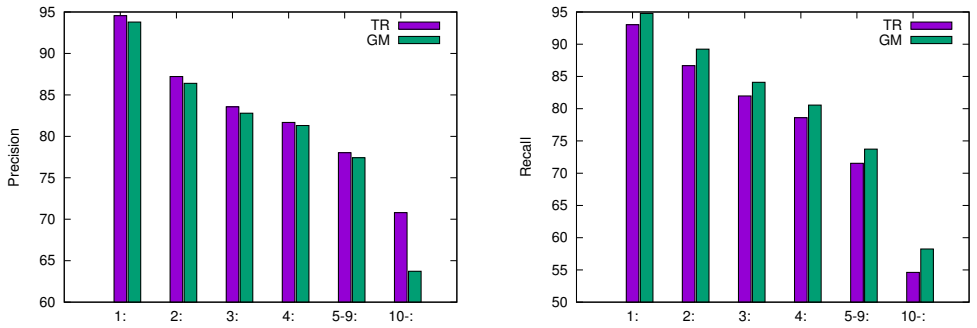


Figure 17
Labeled precision and recall relative to dependency length. "TR" denotes the transition-based model; "GM" denotes the graph merging model.

Table 8
Accuracies with respect to dependency relations. The results are evaluated on the development data. Only the relation types that appear more than 500 times on the development data are reported.

Relation	#	Transition-based			Graph merging		
		P	R	F	P	R	F
ADV	4,795	89.35	80.15	84.50	87.24	79.96	83.44
AMOD	1,309	88.45	88.31	88.38	89.86	89.38	89.62
AUX	926	88.63	86.72	87.66	88.26	86.07	87.15
COMP	7,930	89.73	86.96	88.33	87.27	90.68	88.94
DET	583	95.59	92.97	94.26	93.48	95.88	94.67
LOC	572	80.29	67.66	73.43	81.04	68.01	73.95
NMOD	8,563	89.45	87.43	88.43	88.09	91.00	89.52
OBJ	5,406	89.66	86.29	87.94	89.94	89.68	89.81
QUANT	897	92.02	92.53	92.27	92.38	94.65	93.50
RELATIVE	1,201	92.06	82.01	86.75	92.12	86.68	89.32
SBJ	6,313	82.15	80.41	81.27	81.50	79.41	80.44
TMP	1,273	78.76	71.64	75.03	79.49	73.68	76.48

the graph merging parser: for most types, the precision is lower than its recall. Note that the overall F-scores of these two systems are somehow equivalent.

6.4.2 Deep vs. Deep. CCG and HPSG parsers also favor the dependency-based metrics for evaluation (Clark and Curran 2007b; Miyao and Tsujii 2008). Previous work on Chinese CCG and HPSG parsing unanimously agree that obtaining the deep analysis of Chinese is more challenging (Yu et al. 2011; Tse and Curran 2012). The successful C&C and Enju parsers provide very inaccurate results for Chinese texts. Though the numbers profiling the qualities of deep dependency structures under different formalisms are not directly comparable, all empirical evaluation indicates that the state of the art of deep linguistic processing for Chinese lags very much behind.

6.4.3 Impact of POS Tagging. According to our previous study (Sun and Wan 2016), the use of different POS taggers has a great impact on syntactic analysis. This is highly related to a prominent language-specific property of Mandarin Chinese: as an analytic language, Mandarin Chinese lacks morphosyntactic features to explicitly indicate lexical categories. To evaluate the parsing performance in a more realistic setup, we report parsing results based on two different POS taggers introduced in Sun and Wan (2016). Table 9 presents the results. We can see that automatic POS tagging has a great impact on deep dependency parsing. Even when assisted with a state-of-the-art tagger that is highly engineered, the parser still performs rather poorly.

6.5 Improved Results with ELMo

Table 9 indicates the importance of high-quality POS tagging. We take it as a reflection of the importance of lexical information. Chinese lacks explicit morphosyntactic features, which makes it hard to infer the grammatical functions from word content only. That means contextual clues are essential for predicting a dependency relation.

Table 9
Parsing accuracies with different POS taggers. The results are evaluated on the development data. "LGLM" is short for the POS tagger based on a linear-chain global linear model, while "SC" denotes the same POS tagger enhanced with structure compilation.

		UP	UR	UF	UCM	LP	LR	LF	LCM
GM	Gold	88.75	87.96	88.36	29.34	86.43	85.66	86.05	25.09
	LGLM	81.99	81.62	81.80	23.50	76.94	76.59	76.77	19.58
	SC	83.65	83.03	83.34	23.78	79.03	78.43	78.73	20.09
TR	Gold	89.59	85.50	87.49	24.74	87.49	83.49	85.44	21.45
	LGLM	83.69	79.22	81.39	19.92	78.92	74.71	76.76	17.03
	SC	85.15	80.73	82.88	20.89	80.72	76.54	78.57	17.76

6.5.1 ELMo: Contextualized Word Embedding. Quite recently, Peters et al. (2018) proposed ELMo, an unsupervised approach to model words, especially for modeling their syntactic and semantic property in different linguistic contexts. The key design of ELMo is to train an LSTM-based language model and utilize the hidden vector as contextualizd word embeddings. In particular, every word is assigned a vector according to information on the whole sentence rather than several neighboring words in a limited context. ELMo has been proved very effective in inducing both syntagmatic and paragmatic knowledges and thus extremely useful for improving various English NLP tasks. Incorporating the word embeddings obtained by ELMo into a parser is very simple: We can take such embeddings as additional input word vectors and keep other things unchanged.

6.5.2 Training an ELMo Model for Chinese. In this work, we trained an ELMo model for Chinese and applied it to enhance our GR parsers. The model is trained using Chinese Gigawords, a comprehensive archive of newswire text data that has been acquired over several years by the Linguistic Data Consortium (LDC), choosing the Mandarin news text, that is, Xinhua newswire. This data covers all news published by Xinhua News Agency (the largest news agency in China) from 1991 to 2004, which contains about 12 million sentences. Different from English and other Western languages, Chinese is written without explicit word delimiters such as space characters. In our experiments, we employ a supervised segmenter introduced in Sun and Xu (2011) to process the raw texts.

6.5.3 Main Results. Table 10 presents the results. We concatenate the 1024-dimensional ELMo with word and sometimes POS tag embeddings on our neural parsers. The ELMo vectors significantly improve the performance of our parser, and this gain is more obvious when gold standard POS tags are removed.

Table 11 reports the graph merging results. When ELMo vectors are added, both the individual and the joint decoders are improved. Furthermore, Table 12 shows the performance for representative grammatical functions. Note that only the word content and their corresponding ELMo embeddings are utilized in this experiment. This is a realistic setup to evaluate how accurate the GR parsers could be for processing real-world Chinese texts.

Table 10
Parsing accuracies with ELMo. The results are evaluated on the development data. "NoPOS" means training and parsing without POS tags.

		UP	UR	UF	UCM	LP	LR	LF	LCM
GM	Gold+ELMo	91.01	91.05	91.03	31.93	88.76	88.81	88.79	26.21
	LGLM+ELMo	86.16	86.69	86.42	25.77	80.77	81.27	81.02	20.09
	NoPOS+ELMo	88.86	88.97	88.92	28.93	84.98	85.08	85.03	22.12
TR	Gold+ELMo	86.89	92.92	89.80	26.38	84.65	90.53	87.49	22.35
	LGLM+ELMo	82.01	88.92	85.32	21.40	76.67	83.13	79.77	17.88
	NoPOS+ELMo	84.40	91.24	87.69	22.86	80.59	87.12	83.73	18.15

Table 11
Accuracies of the graph merging–based parser on development set without POSTag information. A first-order graph-based parser with a neural scoring model is used for tree parsing. ELMo is used.

		UP	UR	UF	UCM	LP	LR	LF	LCM
SM	Subgraph1	89.74	79.79	84.47	19.80	85.69	76.19	80.66	17.07
	Subgraph2	89.47	81.63	85.37	19.06	85.43	77.94	81.51	15.94
	Subgraph3	90.62	81.50	85.82	20.76	86.60	77.88	82.01	16.28
	Merged	84.40	91.24	87.69	22.86	80.59	87.12	83.73	18.15
LR	Subgraph1	91.13	80.82	85.67	21.10	86.95	77.11	81.73	17.30
	Subgraph2	90.86	82.78	86.63	20.31	86.67	78.97	82.64	16.73
	Subgraph3	91.56	82.38	86.73	21.21	87.49	78.72	82.88	16.79
	Merged	88.86	88.97	88.92	28.93	84.98	85.08	85.03	22.12

Table 12
Accuracies with respect to representative grammatical functions. The results are obtained by the graph merging parser on the development data. The ELMo vectors are utilized but not POS tags.

Relation	P	R	F
ADV	83.08	79.81	81.41
AMOD	82.92	80.76	81.83
COMP	87.17	90.13	88.63
LOC	82.62	70.63	76.15
NMOD	86.41	89.19	87.78
OBJ	88.08	91.00	89.52
SBJ	82.29	80.59	81.43
TMP	78.39	76.36	77.36

6.5.4 Comparing the POS Tags and the ELMo Vectors. Both gold POS tags and ELMo word embeddings have improved the GR parser with comparable effects, as shown by equivalent results. If we consider only the unlabeled parsing quality, POS tags and ELMo help the GR parsers in a different way. To evaluate the differences between two

Table 13
Recalls with respect to dependency types, that is, local or non-local dependencies. The results are evaluated on the development data.

Model	UR_L	LR_L	UR_{NL}	LR_{NL}
GM (−ELMo)	88.66	86.68	72.60	63.21
GM (+ELMo)	91.65	89.64	77.84	70.48

models trained with gold POS tags and ELMo, respectively, we define the following metric:

$$\frac{2 * |\mathcal{D}_{\text{POS}} \cap \mathcal{D}_{\text{ELMo}}|}{|\mathcal{D}_{\text{POS}}| + |\mathcal{D}_{\text{ELMo}}|}$$

where $\mathcal{D}_{\text{POS}}/\mathcal{D}_{\text{ELMo}}$ denotes the set of dependencies related to the held-out sentences returned by the POS/ELMo-enhanced model. When evaluated on the development data, the values of the unlabeled and labeled metric are 84.46 and 81.05, respectively. These relatively low values highlight the complementary strengths of the POS tags and ELMo vectors. The complementary strengths are also confirmed by the experiment that uses POS and ELMo together. Comparing results in Tables 9 and 10, we can see clearly a significant improvement.

6.5.5 Local vs. Non-Local. Although the micro accuracy of all dependencies are considerably good, the ability of current state-of-the-art statistical parsers to find difficult non-local materials is far from satisfactory, even for English (Rimell, Clark, and Steedman 2009; Bender et al. 2011). We report the accuracy in terms of local and non-local dependencies, respectively, to show the difficulty of the recovery of non-local dependencies. Table 13 demonstrates the labeled/unlabeled recall of local (UR_L/LR_L) and non-local dependencies (UR_{NL}/LR_{NL}). We can clearly see that non-local dependency recovery is extremely difficult for Chinese parsing. Another noteworthy thing is that the ELMo word embeddings improves the prediction for non-local dependencies with a very significant margin.

6.6 Results on the Test Data

Table 14 gives the parsing accuracy on the test set. We evaluate different architectures and different scoring models, as compared to the results reported in Zhang et al. (2016). The beam size of our transition-based parser is set to 16, which is the same as Zhang et al. (2016). Comparing the global linear and LSTM-based models, we can see the advantage of applying neural models for syntactic parsing as well as the effectiveness of our graph merging model: It is significantly better than the transition-based parser. Table 15 shows the results obtained using a realistic setup for real-world applications: The POS tags are removed while the ELMo vectors are incorporated.

Table 14
Accuracies of different models on test set. "GM" is short for graph merging; "TR (DO)" is short for transition-based parsing with dynamic oracle; "TR (BS)" is short for transition-based parsing with beam search. The performance of the *TR (BS)* parser with linear scoring model is from Zhang et al.'s paper. Gold-standard POS tags are utilized.

		UP	UR	UF	UCM	LP	LR	LF	LCM
GM	Neural	88.84	88.23	88.53	29.14	86.47	85.87	86.17	24.68
GM	Linear	88.06	85.11	86.56	26.24	86.03	83.16	84.57	22.84
TR (DO)	Neural	89.56	85.75	87.61	25.11	87.41	83.69	85.51	21.86
TR (BS)	Neural	84.77	87.70	86.21	20.61	82.21	85.05	83.61	18.07
TR (BS)	Linear	- -	- -	- -	- -	82.28	83.11	82.69	- -

Table 15
Accuracies of the GM model on test set. No POS tags are utilized, whereas the ELMo vectors are used.

POS	ELMo	UP	UR	UF	UCM	LP	LR	LF	LCM
NO	YES	89.18	88.86	89.02	29.25	85.05	84.75	84.90	22.68

7. Conclusion

The availability of large-scale treebanks has contributed to the blossoming of statistical approaches to build accurate *surface* constituency and dependency parsers. Recent years have witnessed rapid progress on deep linguistic processing of English texts, and initial attempts have been made for Chinese. Our work attempts to strike a balance between traditional CFG, dependency tree parsing, and deep linguistic processing. By integrating the linguistically-deep grammatical function information and the key bilexical relation underlying the dependency analysis, we propose a new type of deep dependency analysis for parsing Mandarin Chinese sentences. Based on LFG-like dependency annotations, we developed statistical models for deep dependency parsing in both constituency and dependency formalisms. To construct complex linguistic graphs beyond trees, we propose a new approach, namely graph merging, by constructing GR graphs from subgraphs. There are two key issues involved in the approach, that is, graph decomposition and merging. To solve these two problems in a principled way, we treat both problems as optimization problems and employ combinatorial optimization techniques. Experiments demonstrate the effectiveness of the graph merging framework, which can be adopted to other types of flexible representations, for example, semantic dependency graphs (Oepen et al. 2014, 2015) and abstract meaning representations (Banarescu et al. 2013). The study is significant in demonstrating a linguistically motivated and computationally effective way of parsing Chinese texts.

Appendix

In annotations of dependency structures, typical grammatical relations are subject, object, etc., which imply the role the dependent plays with regard to its head. In addition to phrasal categories, CTB also has functional labels to represent relations. The CTB

Table 16
Illustration of major dependency relations in our corpus.

ADV	adverbial adjunct
AMOD	adjectival modifier
AUX	auxiliary
COMP	complement
DET	determiner
LOC	location
NMOD	nominal modifier
OBJ	object
PRT	particle
QUANT	quantification
RELATIVE	relative clause
SUBJ	subject
TMP	temporal modifier

utilizes syntactic distribution as the main criterion for distinguishing lexical categories. By exploring CTB's original annotations, we define a rich set of dependency relations. Table 16 lists the major relations. For more details, please refer to the source codes.

References

Andor, Daniel, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, August.

Banarescu, Laura, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Association for Computational Linguistics, Sofia, Bulgaria, August.

Bender, Emily M., Dan Flickinger, Stephan Oepen, and Yi Zhang. 2011. Parser evaluation over local and non-local deep dependencies in a large corpus. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 397–408, Edinburgh, July. Association for Computational Linguistics.

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, August, COLING 2010 Organizing Comittee.

Bresnan, Joan, and Ronald 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.

Bresnan, Joan. 2001. *Lexical-Functional Syntax*. Number 16. Blackwell, Malden, MA.

Briscoe, Ted, and John Carroll. 2006. Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 41–48, Sydney, July. Association for Computational Linguistics.

Cahill, Aoife, Mairead Mccarthy, Josef Van Genabith, and Andy Way. 2002. Automatic annotation of the Penn-treebank with LFG f-structure information. In *Proceedings of the LREC Workshop on Linguistic Knowledge Acquisition and Representation: Bootstrapping Annotated Language Data*, Las Palmas, Canary Islands, pages 8–15.

Carnie, Andrew. 2007. *Syntax: A Generative Introduction*, 2nd edition, Blackwell Publishing, Malden, MA.

Chen, Danqi, and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, Association for Computational Linguistics.

Chomsky, Noam. 1981. *Lectures on Government and Binding*. Foris Publications, Dordecht.

Chu, Yoeng-Jin, and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

Clark, Stephen, and James Curran. 2007a. Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 248–255, Prague, June.

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

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

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

Dalrymple, M. 2001. *Lexical-Functional Grammar*. Volume 34 of *Syntax and Semantics*, 34, Academic Press, New York.

Dozat, Timothy, and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734.

Edmonds, Jack. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

Eisner, Jason M. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, pages 340–345, Stroudsburg, PA.

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.

Goldberg, Yoav, and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. *COLING-2012*, 2(December): 959–976.

Guo, Yuqing, Josef van Genabith, and Haifeng Wang. 2007. Treebank-based acquisition of LFG resources for Chinese. In *Proceedings of the LFG07 Conference*, CSLI Publications, Stanford, CA, pages 214–232.

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.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November.

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, Sweden, July.

Hudson, Richard. 1990. *English Word Grammar*. Blackwell, Malden, MA.

Joshi, Aravind K., and Yves Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, Volume 3, Springer, Berlin, New York, pages 69–124.

Kaplan, Ron, Stefan Riezler, Tracy H. King, John T. Maxwell III, Alex Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In Daniel Marcu Susan Dumais and Salim Roukos, editors. *HLT-NAACL 2004: Main Proceedings*, pages 97–104, Boston, May 2–May 7. Association for Computational Linguistics.

King, Tracy Holloway, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 dependency bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora*, pages 1–8.

Kiperwasser, Eliyahu, and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

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, Sweden, July.

Koo, Terry, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA, October. Association for Computational Linguistics.

Kübler, Sandra, Ryan T. McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool, London.

Kuhlmann, Marco. 2010. *Dependency Structures and Lexicalized Grammars: An Algebraic Approach*. Ph.D. thesis, Saarland University.

Li, Zhenghua, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Li. 2011. Joint models for Chinese POS tagging and dependency parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1180–1191, Edinburgh, July. Association for Computational Linguistics.

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

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*, Volume 6, pages 81–88.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, October. Association for Computational Linguistics.

Mel'čuk, I. A. 2001. *Communicative Organization in Natural Language: The Semantic-Communicative Structure of Sentences*. *Studies in Language Companion Series*. J. Benjamins, Amsterdam.

Miyao, Yusuke, Takashi Ninomiya, and Jun'ichi Tsujii. 2005. Corpus-oriented grammar development for acquiring a head-driven phrase structure grammar from the Penn Treebank. In *The Second International Joint Conference on Natural Language Processing*, pages 684–693.

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, Ohio, June. Association for Computational Linguistics.

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

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

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, June. Association for Computational Linguistics.

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In Hwee Tou Ng and Ellen Riloff, editors. *HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 49–56, Boston, May 6–May 7. Association for Computational Linguistics.

Oepen, Stephan, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, and Zdenka Uresova. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 915–926, Denver, Colorado, June. Association for Computational Linguistics.

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*, pages 63–72, Dublin, August. Association for Computational Linguistics and Dublin City University.

Oepen, Stephan, Erik Velldal, Jan Tore Lnning, Paul Meurer, Victoria Rosén, and Dan Flickinger. 2007. Towards hybrid quality-oriented machine translation. On linguistics and probabilities in MT. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation*, pages 144–153.

Peters, Matthew, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for*

Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 2227–2237.

Pitler, Emily, Sampath Kannan, and Mitchell Marcus. 2013. Finding optimal 1-endpoint-crossing trees. *TACL*, 1:13–24.

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

Rimell, Laura, Stephen Clark, and Mark Steedman. 2009. Unbounded dependency recovery for parser evaluation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 813–821, Singapore, Association for Computational Linguistics.

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, UK, August. Coling 2008 Organizing Committee.

Steedman, M. 1996. *Surface Structure and Interpretation*. Linguistic Inquiry Monographs. MIT Press, Cambridge, MA.

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, June. Association for Computational Linguistics.

Sun, Weiwei, Yantao Du, and Xiaojun Wan. 2017. Parsing for grammatical relations via graph merging. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 26–35, Vancouver, August. Association for Computational Linguistics.

Sun, Weiwei, and Hans Uszkoreit. 2012. Capturing paradigmatic and syntagmatic lexical relations: Towards accurate Chinese part-of-speech tagging. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 242–252, Jeju Island, Korea, July.

Sun, Weiwei, and Xiaojun Wan. 2016. Towards accurate and efficient Chinese part-of-speech tagging. *Computational Linguistics*, 42(3):391–419.

Sun, Weiwei, and Jia Xu. 2011. Enhancing Chinese word segmentation using unlabeled data. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 970–979, Edinburgh, July. Association for Computational Linguistics.

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 Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 342–350, Suntec, Singapore, August.

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, August. Coling 2010 Organizing Committee.

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, June. Association for Computational Linguistics.

Wang, Mengqiu, Kenji Sagae, and Teruko Mitamura. 2006. A fast, accurate deterministic parser for Chinese. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 425–432, Sydney, July.

Watanabe, Taro, and Eiichiro Sumita. 2015. Transition-based neural constituent 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 1169–1179.

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, July.

Wu, Xianchao, Takuya Matsuzaki, and Jun'ichi Tsujii. 2010. Fine-grained tree-to-string translation rule extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 325–334, Uppsala, Sweden, July.

Xia, Fei. 2001. *Automatic Grammar Generation from Two Different Perspectives*. Ph.D. thesis, University of Pennsylvania.

Xue, Naiwen, Fei Xia, Fu-dong Chiou, and Marta Palmer. 2005. The Penn Chinese Treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11:207–238, June.

Xue, Nianwen. 2007. Tapping the implicit information for the PS to DS conversion of the Chinese Treebank. In *Proceedings of the Sixth International Workshop on Treebanks and Linguistics Theories*, pages 431–438.

Xue, Nianwen, and Martha Palmer. 2009. Adding semantic roles to the Chinese treebank. *Natural Language Engineering*, 15:143–172, January.

Yakushiji, Akane, Yusuke Miyao, Yuka Tateisi, and Jun'ichi Tsujii. 2005. Biomedical information extraction with predicate-argument structure patterns. In *Proceedings of the Eleventh Annual Meeting of the Association for Natural Language Processing*, pages 60–69.

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

Yu, Kun, Yusuke Miyao, Takuya Matsuzaki, Xiangli Wang, and Junichi Tsujii. 2011. Analysis of the difficulties in Chinese deep parsing. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 48–57, Dublin, October. Association for Computational Linguistics.

Yu, Kun, Miyao Yusuke, Xiangli Wang, Takuya Matsuzaki, and Junichi Tsujii. 2010. Semi-automatically developing Chinese HPSG grammar from the Penn Chinese Treebank for deep parsing. In *Coling 2010: Posters*, pages 1417–1425, Beijing, August. Coling 2010 Organizing Committee.

Zhang, Xun, Yantao Du, Weiwei Sun, and Xiaojun Wan. 2016. Transition-based parsing for deep dependency structures. *Computational Linguistics*, 42(3):353–389.

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, October. Association for Computational Linguistics.

Zhang, Yue, and Stephen Clark. 2009. Transition-based parsing of the Chinese Treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 162–171, Paris, October. Association for Computational Linguistics.

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, Oregon, June.

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

Zhou, Hao, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency 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 1213–1222.