Parsing for Grammatical Relations via Graph Merging

Weiwei Sun, Yantao Du and **Xiaojun Wan**
Institute of Computer Science and Technology, Peking University
The MOE Key Laboratory of Computational Linguistics, Peking University
{ws,duyantao,wanxiaojun}@pku.edu.cn

Abstract

This paper is concerned with building deep grammatical relation (GR) analysis using data-driven approach. To deal with this problem, we propose graph merging, a new perspective, for building flexible dependency graphs: Constructing complex graphs via constructing simple subgraphs. We discuss two key problems in this perspective: (1) how to decompose a complex graph into simple subgraphs, and (2) how to combine subgraphs into a coherent complex graph. Experiments demonstrate the effectiveness of graph merging. Our parser reaches state-of-the-art performance and is significantly better than two transition-based parsers.

1 Introduction

Grammatical relations (GRs) represent functional relationships between language units in a sentence. Marking not only local but also a wide variety of long distance dependencies, GRs encode in-depth information of natural language sentences. Traditionally, GRs are generated as a by-product by grammar-guided parsers, e.g. RASP (Carroll and Briscoe, 2002), C&C (Clark and Curran, 2007b) and Enju (Miyao et al., 2007). Very recently, by representing GR analysis using general directed dependency graphs, Sun et al. (2014) and Zhang et al. (2016) showed that considerably good GR structures can be directly obtained using data-driven, transition-based parsing techniques. We follow their encouraging work and study the data-driven approach for producing GR analyses.

The key challenge of building GR graphs is due to their flexibility. Different from surface syntax, the GR graphs are not constrained to trees, which is a fundamental consideration in designing parsing algorithms. To deal with this problem, we propose graph merging, a new perspective, for building flexible representations. The basic idea is to decompose a GR graph into several subgraphs, each of which captures most but not 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 whole target GR structure to be produced.

There are two major problems in the graph merging perspective. First, how to decompose a complex graph into simple subgraphs in a principled way? To deal with this problem, we considered structure-specific properties of the syntactically-motivated GR graphs. One key property is their reachability: In a given GR graph, almost every node is reachable from a same and unique root. If a node is not reachable, it is disconnected from other nodes. This property ensures a GR graph to be successfully decomposed into limited number of forests, which in turn can be accurately and efficiently built via tree parsing. We model the graph decomposition problem as an optimization problem and employ Lagrangian Relaxation for solutions.

Second, how to 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 treat this problem as a combinatory optimization problem and also employ Lagrangian Relaxation to solve the problem. In particular, the parsing phase consists of two steps. First, graph-based models are applied to assign scores to individual arcs and various tuples of arcs. Then, a Lagrangian Relaxation-based joint decoder is applied to efficiently produces globally optimal GR graphs according to all graph-based models.

We conduct experiments on Chinese GRBank

Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), pages 26–35,
Vancouver, Canada, August 3 - August 4, 2017. ©2017 Association for Computational Linguistics

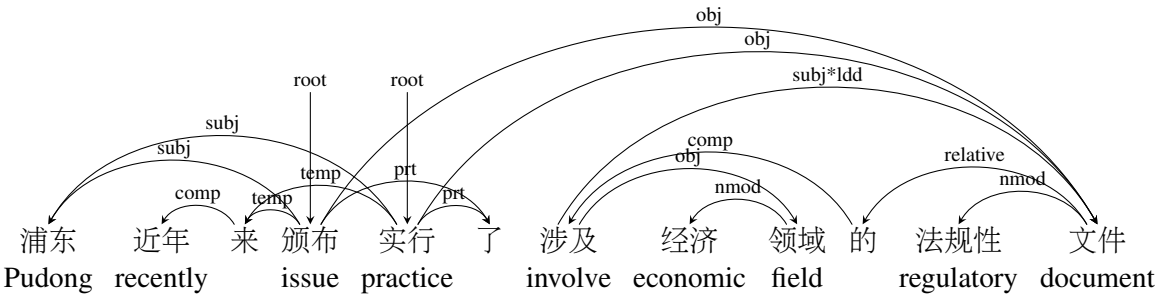

Figure 1: An example: *Pudong recently enacted regulatory documents involving the economic field.*

(Sun et al., 2014). Though our parser does not use any phrase-structure information, it produces high-quality GR analysis with respect to dependency matching. Our parsers obtain a labeled f-score of 84.57 on the test set, resulting in an error reduction of 15.13% over Sun et al. (2014)'s single system. and 10.86% over Zhang et al. (2016)'s system. The remarkable parsing result demonstrates the effectiveness of the graph merging framework. This framework can be adopted to other types of flexible representations, e.g. semantic dependency graphs (Oepen et al., 2014, 2015) and abstract meaning representations (Banarescu et al., 2013).

2 Background

In this paper, we focus on building GR analysis for Mandarin Chinese. Mandarin is an analytic language that lacks inflectional morphology (almost) entirely and utilizes highly configurational ways to convey syntactic and semantic information. This analytic nature allows to represent all GRs as bilexical dependencies. Sun et al. (2014) showed that analysis for a variety of complicated linguistic phenomena, e.g. coordination, raising/control constructions, extraction, topicalization, can be conveniently encoded with directed graphs. Moreover, such deep syntactic dependency graphs can be effectively derived from Chinese TreeBank (Xue et al., 2005) with very high quality. Figure 1 is an example. In this graph, "subj*ldd" between the word "涉及/involve" and the word "文件/documents" represents a long-distance subject-predicate relation. The arguments and adjuncts of the coordinated verbs, namely "颁布/issue" and "实行/practice," are separately yet distributively linked to the two heads.

By encoding GRs as directed graphs over words, Sun et al. (2014) and Zhang et al. (2016) showed that the data-driven, transition-based approach can be applied to build Chinese GR structures with very promising results. This architecture is complementary to the traditional approach to English GR analysis, which leverages grammar-guided parsing under deep formalisms, such as LFG (Kaplan et al., 2004), CCG (Clark and Curran, 2007a) and HPSG (Miyao et al., 2007). We follow Sun et al.'s and Zhang et al.'s encouraging work and study the discriminative, factorization models for obtaining GR analysis.

3 The Idea

The key idea of this work is constructing a complex structure via constructing simple partial structures. Each partial structure is *simple* in the sense that it allows efficient construction. For instance, projective trees, 1-endpoint-corssing trees, non-crossing dependency graphs and 1-endpoint-crossing, pagenumber-2 graphs can be taken as simple structures, given that low-degree polynomial time parsing algorithms exist (Eisner, 1996; Pitler et al., 2013; Kuhlmann and Jonsson, 2015; Cao et al., 2017; Sun et al., 2017). To construct each partial structure, we can employ mature parsing techniques. To get the final target output, we also require the total of all partial structures enables whole target structure to be produced. In this paper, we 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, shown in Figure 2. If we can parse the sentence into subgraphs and combine them in a principled way, we get the original GR graph.

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

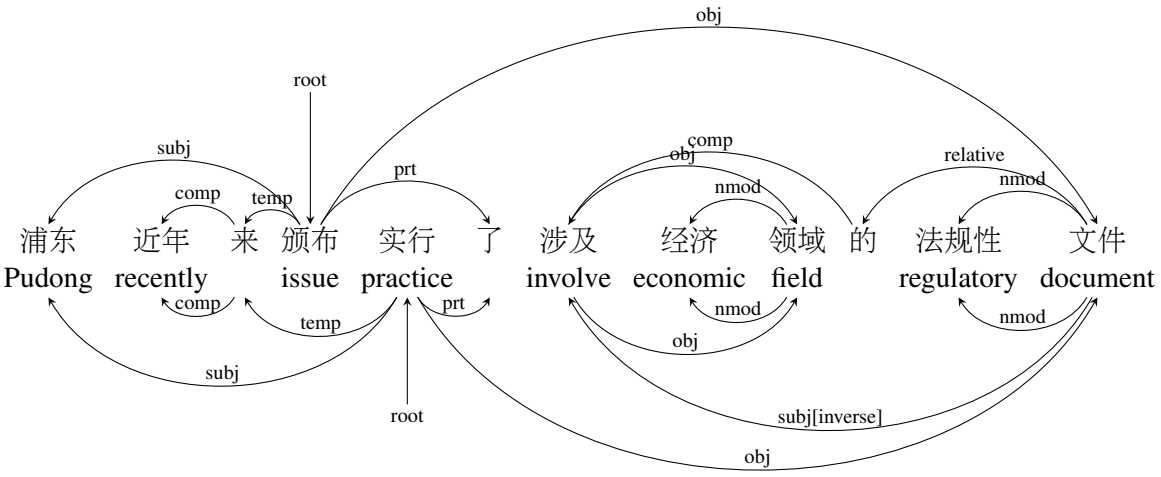

Figure 2: A graph decomposition for the GR graph in Figure 1. The two subgraphs are shown on two sides of the sentence respectively. The subgraph on the upper side of the sentence is exactly a tree, while the one on the lower side is slightly different. The edge from the word "文件/document" to "涉及/involve" is tagged "[inverse]" to indicate that the direction of the edge in the subgraph is in fact opposite to that in the original graph.

structures as outputs. We are going to demonstrate the techniques we use to solve these two problems.

4 Decomposing GR Graphs

4.1 Graph Decomposition as Optimization

Given a sentence $s = \mathbf{w}_1\mathbf{w}_2\cdots\mathbf{w}_n$ of length n, we use a vector $\boldsymbol{y}$ of length n^2 to denote a graph on it. We use indices i and j to index the elements in the vector, $y(i,j) \in \{0,1\}$, denoting whether there is an arc from w_i to w_j ($1 \le i,j \le n$).

Given a graph $\boldsymbol{y}$, we hope to find m subgraphs $\boldsymbol{y}_1$, ..., $\boldsymbol{y}_m$, 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 sub-models. Therefore, we hope each subgraph $\boldsymbol{y}_k$ is informative enough to train a good disambiguation 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,

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

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 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 by itself an independent connected component. This property allows a GR graph to be decomposed into limited number of *tree-like* subgraphs. By tree-like we mean if we treat a graph on a sentence as undirected, it is a tree, or it is 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 link the virtual root and this word. Furthermore, we can decompose the graph into two tree-like subgraphs, as shown in Figure 2. In this decomposition, one subgraph is exactly a tree, and the other is very close to a tree.

We restrict the number of subgraphs to 3. The intuition is that we use one tree to capture long distance information and the other two to capture

coordination information.[1] In other words, we decompose each given graph $\boldsymbol{y}$ into three tree-like subgraphs $\boldsymbol{g}_1$, $\boldsymbol{g}_2$ and $\boldsymbol{g}_3$. The goal is to let $\boldsymbol{g}_1, \boldsymbol{g}_2$ and $\boldsymbol{g}_3$ carry important information of the graph as well as cover all edges in $\boldsymbol{y}$. The optimization problem can be written as

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

4.2.1 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 we define the score function, extracting a subgraph from a GR graph works like this: 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:

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

$\boldsymbol{g}_k$ is not exactly a subgraph of $\boldsymbol{y}$, because there may be some edges in the tree but not in the graph. To guarantee we get a 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 necessary the same as the original graph $\boldsymbol{y}$, but must be a subgraph of it.

4.2.2 Three Variations of Scoring

With different weight assignments, we can extract different trees from a graph, obtaining different subgraphs. We devise three variations of weight assignment: ω_1, ω_2, and ω_3. Each ω_k (k is 1,2 or 3) 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

$$\begin{aligned} S_1(i,j) &= \begin{cases} c_1 & \text{if } \boldsymbol{y}(i,j) = 1 \text{ or } \boldsymbol{y}(j,i) = 1 \\ 0 & \text{else} \end{cases} \\ S_2(i,j) &= \begin{cases} c_2 & \text{if } \boldsymbol{y}(i,j) = 1 \\ 0 & \text{else} \end{cases} \\ S_3(i,j) &= c_3(n - |i-j|) \\ S_4(i,j) &= c_4(n - l_p(i,j)) \end{aligned}$$

In the definitions above, c_1, c_2, c_3 and c_4 are coefficients, satisfying $c_1 \gg c_2 \gg c_3$, 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 exits, then $l_p(i,j) = n$. The intuition behind the design is illustrated below.

S_1 indicates whether there is an edge between i and j, and we want it to matter mostly;

S_2 indicates whether the edge is from i to j, and we want the edge with correct direction to be selected more likely;

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 type of path between i and j that reflects c-commanding relationships, and the coefficient remains to be tuned.

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

[1] In this paper, 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, while 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.

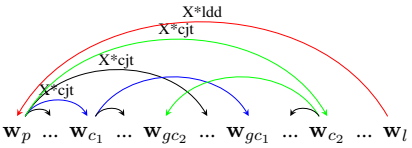

Figure 3: Examples to illustrate the additional weights.

V_1. First for edges $\boldsymbol{y}(i,j)$ whose label is tagged with *ldd, we assign $V_1(i,j) = d$. d is a coefficient to be tuned on validation data.. Whenever we come across a parent p with a set of conjunction children $cjt_1, cjt_2, \cdots, cjt_n$, we find 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 that are added additional d's to are shown in blue in Figure 3.

V_2. Different from V_1, for edges $\boldsymbol{y}(i,j)$ whose label is tagged with *ldd, we assign an $V_2(j,i) = d$. 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 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 3.

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 an d to $V_3(p, cjt_1)$, $V_3(p, cjt_2)$, $\cdots$, and $V_3(p, cjt_n)$.

4.3 Lagrangian Relaxation with Approximation

As soon as we get three trees $\boldsymbol{g}_1$, $\boldsymbol{g}_2$ and $\boldsymbol{g}_3$, we get three subgraphs $t2g(\boldsymbol{g}_1)$, $t2g(\boldsymbol{g}_2)$ and $t2g(\boldsymbol{g}_3)$. As is stated above, we want every edge in a graph $\boldsymbol{y}$ to be covered by at least one subgraph, and we want 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

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

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

Algorithm 1: The Tree Extraction Algorithm

Initialization: set $u^{(0)}$ to 0
for $k = 0$ **to** K **do**
  $\boldsymbol{g}_1 \leftarrow \arg\max_{\boldsymbol{g}_1} s_1(\boldsymbol{g}_1) + u^{(k)\top}\boldsymbol{g}_1$
  $\boldsymbol{g}_2 \leftarrow \arg\max_{\boldsymbol{g}_2} s_2(\boldsymbol{g}_2) + u^{(k)\top}\boldsymbol{g}_2$
  $\boldsymbol{g}_3 \leftarrow \arg\max_{\boldsymbol{g}_3} s_3(\boldsymbol{g}_3) + u^{(k)\top}\boldsymbol{g}_3$
  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 $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3$

of the problem is

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

where u is the Lagrangian multiplier.

Then the dual is

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

According to the duality principle, $\max_{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3; u} \min_u \mathcal{L}(\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3) = \min_u \mathcal{L}(u)$, so 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.

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

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

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

4.4 The Algorithm

Algorithm 1 is our tree decomposition algorithm. In the algorithm, we use subgradient method to find $\min_u \mathcal{L}'(u)$ iteratively. 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, 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 that it will be more likely for them to be selected in the next iteration.

5 Graph Merging

The extraction algorithm gives three classes of trees for each graph. We apply the algorithm to the graph training set, and get three training tree sets. After that, we can train three parsing models with the three tree sets. In this work, the parser we use to train models and parse trees is Mate (Bohnet, 2010), a second-order graph-based dependency parser.

Let the scores the three models use be f_1, f_2, f_3 respectively. Then the parsers can find trees with highest scores for a sentence. That is solving 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, and 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, we obtain the graph $\boldsymbol{y} = \max\{t2g(\boldsymbol{g}_1), t2g(\boldsymbol{g}_2), t2g(\boldsymbol{g}_3)\}$. We call this process **simple merging**.

However, the simple merging process omits some consistency that the three trees extracted from the same graph achieve, thus losing some important information. The information is that when we decompose a graph into three subgraphs, some edges tend to appear in certain classes of subgraphs at the same time. We want to retain the co-occurrence relationship of the edges when doing parsing and merging. To retain the hidden consistency, we must do *joint* decoding instead of decode the three models separately.

5.1 Capturing the Hidden Consistency

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 tag 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. Specially, since we do not need to consider 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 them. 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, we also get labels with consistency information. Our goal is to guarantee the tags in edges of the parse trees for a same sentence are consistent while graph merging. Since the consistency tags emerge, for convenience we index the graph and tree vector representation using three indices. $\boldsymbol{g}(i, j, t)$ denotes whether there is an edge from word $\mathbf{w}_i$ to word $\mathbf{w}_j$ with tag t in graph g.

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

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

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 $\boldsymbol{g}_2$ exist. If all of these constraints are satisfied, the subgraphs achieve the consistency.

5.2 Lagrangian Relaxation with Approximation

To solve the constrained optimization problem above, we do 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 pointwisely. So can the other inequality constraints. If we take $\boldsymbol{a}_{12}, \boldsymbol{a}_{13}, \cdots, \boldsymbol{a}_{32}$ as constants, then all the constraints are linear. The constraints thus 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

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

Again, we use the subgradient method to minimize $\mathcal{L}(u)$. During the deduction, we take $\boldsymbol{a}_{12}, \boldsymbol{a}_{13}, \cdots, \boldsymbol{a}_{32}$ as constants, but unfortunately they are not. We propose an approximation for the $\boldsymbol{a}$'s in each iteration: Using the $\boldsymbol{a}$'s we got 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.

5.3 The Algorithm

The pseudo code of our algorithm is shown in Algorithm 2. We know that the score functions f_1, f_2, and f_3 each consist of first-order scores and higher order scores. So they can be written as

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

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 $\omega'_k(i,j,t) = \omega_k(i,j,t) + \mathbf{w}_k(i,j,t) + \mathbf{w}_k(j,i,t)$.

Algorithm 2: The Joint Decoding Algorithm

Initialization: set $u^{(0)}, A_1, A_2, A_3$ to 0,
for $k = 0$ **to** K **do**
  $\boldsymbol{g}_1 \leftarrow \arg\max_{\boldsymbol{g}_1} f_1(\boldsymbol{g}_1) + u^{(k)\top} A_1\boldsymbol{g}_1$
  $\boldsymbol{g}_2 \leftarrow \arg\max_{\boldsymbol{g}_2} f_2(\boldsymbol{g}_2) + u^{(k)\top} A_2\boldsymbol{g}_2$
  $\boldsymbol{g}_3 \leftarrow \arg\max_{\boldsymbol{g}_3} f_3(\boldsymbol{g}_3) + u^{(k)\top} A_3\boldsymbol{g}_3$
  update A_1, A_2, A_3
  if $A_1\boldsymbol{g}_1 + A_2\boldsymbol{g}_2 + A_3\boldsymbol{g}_3 = \mathbf{0}$ **then**
    return $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3$
  $u^{(k+1)} \leftarrow u_{(k)} - \alpha^{(k)}(A_1\boldsymbol{g}_1 + A_2\boldsymbol{g}_2 + A_3\boldsymbol{g}_3)$
return $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3$

The update of $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$ can be understood in an intuitive way. When one of the constraints is not satisfied, without loss of generality, say, the first one for edge $\boldsymbol{y}(i,j)$. 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 tag representing $\boldsymbol{g}_2(i,j) = 1$ in $\boldsymbol{g}_1$. After the update of the weights, the consistency is more likely to be achieved.

5.4 Labeled Parsing

For 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.

6 Evaluation and Analysis

6.1 Experimental Setup

We conduct experiments on Chinese GRBank (Sun et al., 2014), an LFG-style GR corpus for Mandarin Chinese. Linguistically speaking, this deep dependency annotation directly encodes information such as coordination, extraction, raising, control as well as many other long-range dependencies. The selection for training, development, test data is also according to Sun et al. (2014)'s experiments. Gold standard POS-tags are used for deriving features for disambiguation.

		UP	UR	UF	UCompl	LP	LR	LF	LCompl
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**

Table 1: Results on development set. *SM* is for Simple Merging, and *LR* for Lagrangian Relaxation.

	UP	UR	UF	UCompl	LP	LR	LF	LCompl
subgraph1	**89.80**	76.74	82.76	18.69	**87.81**	75.04	80.93	17.13
subgraph2	89.34	78.66	83.66	18.46	87.26	76.84	81.72	16.97
subgraph3	89.57	81.23	85.19	20.18	87.78	79.61	83.49	18.22
Merged	88.06	**85.11**	**86.56**	**26.24**	86.03	**83.16**	**84.57**	**22.84**
Sun et al.	-	-	-	-	83.93	79.82	81.82	-
Zhang et al.[Single]	-	-	-	-	82.28	83.11	82.69	-
Zhang et al.[Ensemble]	-	-	-	-	84.92	85.28	85.10	-

Table 2: Lagrangian Relaxation Results on test set.

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.

	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 3: Results of graph decomposition. *SD* is for Simple Decomposition and *LR* for Lagrangian Relaxation

6.2 Results of Graph Decomposition

Table 3 shows the results of graph decomposition on the training set. If we use simple decomposition, say, directly extracting three trees from a graph, we get three subgraphs. On the training set, each kind of the subgraphs cover around 90% edges and 30% sentences. When we merge them together, they cover nearly 97% edges and over 70% sentences. This indicates that the ability of a single tree is limited and three trees can cover most of the edges.

When we apply Lagrangian Relaxation to the decomposition process, both the edge coverage and the sentence coverage gain great error reduction, indicating that Lagrangian Relaxation is very effective on the task of decomposition.

6.3 Results of Graph Merging

Table 1 shows the results of graph merging on the development set, and Table 2 on test set. 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 first decode the three trees for a sentence then combine them by putting all the edges together. As is shown, the merged graph achieves higher f-score than other single models. With Lagrangian Relaxation, the performance of not only

the merged graph but also the three subgraphs are improved, due to capturing the consistency information.

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

The best scores reported by previous work, i.e. (Sun et al., 2014) and (Zhang et al., 2016) are also listed in Table 2. We can see that our subgraphs already achieve competitive scores, and the merged graph with Lagrangian Relaxation improves both unlabeled and labeled f-scores substantially, with an error reduction of 15.13% and 10.86%. We also include Zhang et al.'s parsing result obtained by an ensemble model that integrate six different transition-based models. We can see that parser ensemble is very helpful for deep dependency parsing and the accuracy of our graph merging parser is sightly lower than this ensemble model. Given that the architecture of graph merging is quite different from transition-based parsing, we think system combination of our parser and the transition-based parser is promising.

7 Conclusion

To construct complex linguistic graphs beyond trees, we propose a new perspective, namely graph merging. We take GR parsing as a case study and exemplify the idea. There are two key problems in this perspective, namely 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. This framework can be adopted to other types of flexible representations, e.g. semantic dependency graphs (Oepen et al., 2014, 2015) and abstract meaning representations (Banarescu et al., 2013).

Acknowledgments

This work was supported by 863 Program of China (2015AA015403), NSFC (61331011), and Key Laboratory of Science, Technology and Standard in Press Industry (Key Laboratory of Intelligent Press Media Technology). We thank anonymous reviewers for their valuable comments.

References

Laura Banarescu, 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*. Association for Computational Linguistics, Sofia, Bulgaria, pages 178–186. http://www.aclweb.org/anthology/W13-2322.

Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. Coling 2010 Organizing Committee, Beijing, China, pages 89–97. http://www.aclweb.org/anthology/C10-1011.

Junjie Cao, Sheng Huang, Weiwei Sun, and Xiaojun Wan. 2017. Parsing to 1-endpoint-crossing, pagenumber-2 graphs. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

John Carroll and Ted Briscoe. 2002. High precision extraction of grammatical relations. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*. Association for Computational Linguistics, Stroudsburg, PA, USA, COLING '02, pages 1–7. https://doi.org/10.3115/1072228.1072241.

Y.J. Chu and T.H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica* pages 14:1396–1400.

Stephen Clark and James Curran. 2007a. Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Association for Computational Linguistics, Prague, Czech Republic, pages 248–255. http://www.aclweb.org/anthology/P07-1032.

Stephen Clark and James R. Curran. 2007b. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics* 33(4):493–552. https://doi.org/10.1162/coli.2007.33.4.493.

J. Edmonds. 1967. Optimum branchings. *Journal of Research of the NationalBureau of Standards* pages 71B:233–240.

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

Ron Kaplan, 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*. Association for Computational Linguistics, Boston, Massachusetts, USA, pages 97–104.

Marco Kuhlmann and Peter Jonsson. 2015. Parsing to noncrossing dependency graphs. *Transactions of the Association for Computational Linguistics* 3:559–570.

Yusuke Miyao, Kenji Sagae, and Jun'ichi Tsujii. 2007. Towards framework-independent evaluation of deep linguistic parsers. In Ann Copestake, editor, *Proceedings of the GEAF 2007 Workshop*. CSLI Publications, CSLI Studies in Computational Linguistics Online, pages 238–258. http://www.cs.cmu.edu/ sagae/docs/geaf07miyaoetal.pdf.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic, and Zdenka Uresová. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. Association for Computational Linguistics and Dublin City University, Dublin, Ireland, pages 63–72. http://www.aclweb.org/anthology/S14-2008.

Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2013. Finding optimal 1-endpoint-crossing trees. *TACL* 1:13–24. http://www.transacl.org/wp-content/uploads/2013/03/paper13.pdf.

Weiwei Sun, Junjie Cao, and Xiaojun Wan. 2017. Semantic dependency parsing via book embedding. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Weiwei Sun, 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)*. Association for Computational Linguistics, Baltimore, Maryland, pages 446–456. http://www.aclweb.org/anthology/P14-1042.

Naiwen Xue, 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. https://doi.org/10.1017/S135132490400364X.

Xun Zhang, Yantao Du, Weiwei Sun, and Xiaojun Wan. 2016. Transition-based parsing for deep dependency structures. *Computational Linguistics* 42(3):353–389. http://aclweb.org/anthology/J16-3001.