# A Data-Driven, Factorization Parser for CCG Dependency Structures

**Yantao Du, Weiwei Sun*** 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 CCG-grounded, semantics-oriented deep dependency structures with a data-driven, factorization model. Three types of factorization together with different higher-order features are designed to capture different syntacto-semantic properties of functor-argument dependencies. Integrating heterogeneous factorizations results in intractability in decoding. We propose a principled method to obtain optimal graphs based on dual decomposition. Our parser obtains an unlabeled f-score of 93.23 on the CCGBank data, resulting in an error reduction of 6.5% over the best published result. which yields a significant improvement over the best published result in the literature. Our implementation is available at http://www.icst.pku.edu.cn/lcwm/grass.

## 1 Introduction

Combinatory Categorial Grammar (CCG; Steedman, 2000) is a linguistically expressive grammar formalism which has a transparent yet elegant interface between syntax and semantics. By assigning each lexical category a dependency *interpretation*, we can derive typed dependency structures from CCG derivations (Clark et al., 2002), providing a useful approximation to the underlying meaning representations. To date, CCG parsers are among the most competitive systems for generating such deep bi-lexical dependencies that appropriately encode a wide range of local and non-local syntacto-semantic information (Clark and Curran, 2007a; Bender et al., 2011). Such semantic-oriented dependency structures have been shown very helpful for NLP applications e.g. Question Answering (Reddy et al., 2014).

Traditionally, CCG graphs are generated as a by-product by grammar-guided parsers (Clark and Curran, 2007b; Fowler and Penn, 2010). The main challenge is that a *deep-grammar-guided* model usually can only produce limited coverage and corresponding parsing algorithms is of relatively high complexity. Robustness and efficiency, thus, are two major problems for handling practical tasks. To increase the applicability of such parsers, lexical or syntactic pruning has been shown necessary (Clark and Curran, 2004; Matsuzaki et al., 2007; Sagae et al., 2007; Zhang and Clark, 2011).

In the past decade, the techniques for data-driven dependency parsing has made a great progress (McDonald et al., 2005a,b; Nivre et al., 2004; Torres Martins et al., 2009; Koo et al., 2010). The major advantage of the data-driven architecture is complementary to the grammar-driven one. On one hand, data-driven approaches make essential uses of machine learning from linguistic annotations and are flexible to produce analysis for arbitrary sentences. On the other hand, without hard constraints, parsing algorithms for spanning specific types of graphs, e.g. projective (Eisner, 1996) and 1-endpoint-crossing trees (Pitler et al., 2013), can be of low complexity.

This paper proposes a new data-driven dependency parser that efficiently produces globally optimal CCG dependency graphs according to a discriminative, factorization model. The design of the factorization is motivated by three essential properties of the CCG dependencies. First, all arguments associated with the same predicate are highly correlated due to the nature that they approximates type-logical semantics. Second, all predicates govern the same argument exhibit the hybrid syntactic/semantic, i.e. head-complement-adjunct, relationships. Finally, the CCG dependency graphs are not but look very much like

*Email correspondence.

trees, which have many good computational properties. Simultaneously modeling the three properties yields intrinsically heterogeneous factorizations over the same graph, and hence results in intractability in decoding. Inspired by (Koo et al., 2010; Rush et al., 2010), we employ dual decomposition to perform principled decoding. Though not always, we can obtain the optimal solution most of time. The time complexity of our parser is $O(n^3)$ when various 1st- and 2nd-order features are incorporated.

We conduct experiments on English CCGBank (Hockenmaier and Steedman, 2007). Though our parser does not use any grammar information, including both lexical categories and syntactic derivations, it produces very accurate CCG dependency graphs with respect to both token and complete matching. Our parser obtains an unlabeled f-score of 93.23, resulting in, perhaps surprisingly, an error reduction of up to 6.5% over the best published performance reported in (Auli and Lopez, 2011). Our work indicates that high-quality data-driven parsers can be built for producing more general dependency graphs, rather than trees. Nevertheless, empirical evaluation indicates that explicitly or implicitly using tree-structured information plays an essential role. The result also suggests that a wider range of complicated linguistic phenomena beyond surface syntax can be well modeled even without explicitly using grammars. Our algorithm is also applicable to other graph-structured representations, e.g. HPSG predicate-argument analysis (Miyao et al., 2004).

## 2 Related Work

Hockenmaier and Steedman (2007) developed linguistic resources, namely CCGBank, from the Penn Treebank (PTB; Marcus et al., 1993). In CCGBank, PTB phrase-structure trees have been transformed into normal-form CCG derivations, and deep bi-lexical dependency graphs that encode functor-argument strcutures have been extracted from these derivations using coindexation information. The typed dependency analysis provides a useful approximation to the underlying meaning representations, and has been shown very helpful for NLP applications e.g. Question Answering (Reddy et al., 2014).

Traditionally, CCG graphs are generated as a by-product by deep parsers with a core grammar (Clark et al., 2002; Clark and Curran, 2007b; Fowler and Penn, 2010). On the other hand, modeling these dependencies within a CCG parser has been shown very effective to improve the parsing accuracy (Clark and Curran, 2007b; Xu et al., 2014). Besides CCG, similar deep dependency structures can be also extracted from parsers under other deep grammar formalisms, e.g. LFG (King et al., 2003) and HPSG (Miyao et al., 2004).

In recent years, data-driven dependency parsing has been well studied and widely applied to many NLP tasks. Research on data-driven approach to producing dependency graphs that are not limited to tree or forest structures has also been initialized. Sagae and Tsujii (2008) introduced a transition-based parser that is able to handle projective directed dependency graphs for HPSG-style predicate-argument analysis. McDonald and Pereira (2006) presented a graph-based parser that can generate graphs in which a word may depend on multiple heads, and evaluated it on the Danish Treebank. Encouraged by their work, we study factorization models as well as principled decoding for CCG-grounded, graph-structured representations.

Dual decomposition, and more generally Lagrangian relaxation, is a classical method for solving combinatorial optimization problems. It has been successfully applied to several NLP tasks, including parsing (Koo et al., 2010; Rush et al., 2010) and machine translation (Rush and Collins, 2011). To provide principled decoding for our factorization parser, we employ the dual decomposition technique. Our work directly follows (Koo et al., 2010). The two basic factorizations are similar to the model introduced in (Martins and Almeida, 2014). Lluís et al. (2013) introduced a dual decomposition based joint model for joint syntactic and semantic parsing. They are concerned with shallow semantic representation, i.e. Semantic Role Labeling, whose graphs are sparse. Different from their concern on integrating syntactic parsing and semantic role labeling under 1st-order factorization, we are interested in designing higher-order factorization models for more dense and general linguistic graphs.

## 3 Graph Factorization

### 3.1 Background Notations

Consider a sentence $s = \langle \mathbf{w}, \mathbf{p} \rangle$ with words $\mathbf{w} = w_1 w_2 \cdots w_n$ and POS-tags $\mathbf{p} = p_1 p_2 \cdots p_n$. First we add one more virtual word $w_0 = \#W_{root}\#$

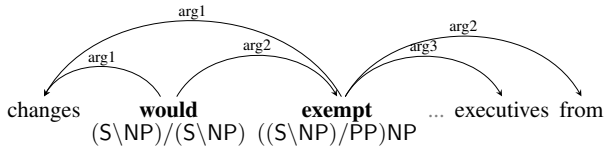


Figure 1: Examples to illustrate the predicate-centric view.

with POS-tag $p_0 = \#P_{root}\#$ which is conventionally considered as the root node of trees or graphs on the sentence. Then we denote the *index set* of all possible dependencies as $\mathcal{I} = \{(i,j)|i \in \{0,\cdots,n\}, j \in \{1,\cdots,n\}, i \neq j\}$. A dependency parse then can be represented as a vector

$$\boldsymbol{y} = \{y(i,j) : (i,j) \in \mathcal{I}\},$$

where $y(i,j) = 1$ if a dependency with predicate $i$ and argument $j$ is in the graph, 0 otherwise. Note that $\boldsymbol{y}$ is not a matrix but a long vector though we use two indexes to index it. In this paper, we only consider the unlabeled parsing task. Nevertheless, it is quite straightforward to extend our models to labeled parsing. Let $\mathcal{Y}$ denote the set of all possible $\boldsymbol{y}$. Given a function $f : \mathcal{Y} \to \mathbb{R}$ that assigns scores to parse graphs, the optimal parse is

$$\boldsymbol{y}^* = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} f(\boldsymbol{y}).$$

Following recent advances in discriminative dependency parsing, we build disambiguation models based on global linear models, as in (McDonald et al., 2005a). In this framework, we score a dependency graph using a linear model:

$$f_\theta(\boldsymbol{y}) = \theta^\top \Phi(s, \boldsymbol{y}),$$

where $\Phi(s, \boldsymbol{y})$ produces a $d$-dimensional vector representation of the event that a CCG graph $\boldsymbol{y}$ is assigned to sentence $s$. In order to perform the decoding efficiently, we assume that the dependency graphs can be *factored* into smaller pieces. The main goal of this paper is to design appropriate factorization models, namely different types of $f_\theta$'s, to reflect essential properties of the semantics-oriented CCG dependency graphs.

### 3.2 Predicate-Centric Factorization

The very fundamental view of the CCG dependency graphs is based on their lexicalized, predicate-centric nature. Every word is assigned a lexical category, which directly encodes its subcategorization information. Due to the type-transparency nature of the formalism, this lexical category provides sufficient information for not only syntactic derivation but also semantic composition. It is important to capture functor-argument relations by putting all arguments of one particular predicate together. Figure 1 gives an example. The predicate "exempt" is of type "((S\NP)/PP)/NP," indicating that it takes three semantic dependents. This part of information is very similar to Semantic Role Labeling (SRL), whose goal is to find semantic roles for verbal predicates as well as their normalization. However, functor-argument analysis grounded in CCG is approximation of underlying logic forms and thus provides bi-lexical relations for almost all words. For instance, the second word in focus—"would"—captures structural information to organize other predicates yet entities.

In order to perform maximization efficiently in this view, we treat each predicate separately. Given a vector $\boldsymbol{y}^p$, we define

$$\boldsymbol{y}^p_{i\curvearrowright} = \{y(i,j) : j \in \{1,\cdots,n\}, j \neq i\}$$

and assume that $f(\boldsymbol{y}^p)$ takes the form

$$f^p(\boldsymbol{y}^p) = \sum_{i=0}^{n} f^p_i(\boldsymbol{y}^p_{i\curvearrowright})$$

To capture the relationships of all arguments to one particular predicate as a whole, we employ a Markov model. Let $a_1, \cdots, a_m$ be the sequence of the arguments of the word $w_i$ under $\boldsymbol{y}^p_{i\curvearrowright}$. To keep the arguments in order, we constrain $1 \leq a_{j_1} < a_{j_2} \leq n$ if $j_1 < j_2$. In a *k-th order predicate-centric model*, we define

$$f^p_i(\boldsymbol{y}^p_{i\curvearrowright}) = \sum_{j=1}^{m+k-1} \theta_p^\top \Phi_p(a_{j-(k-1)}, ..., a_j, i, \mathbf{w}, \mathbf{p})$$

where $a_j$ ($j \leq 0$ or $j \geq m+1$) are treated as specific initial or end state.

Higher-order rather than arc-factored features can be conveniently extracted from adjacent arguments. This is similar to the sibling factorization defined by a number of syntactic tree parsers, e.g. (McDonald and Pereira, 2006), (Koo and Collins, 2010) and (Ma and Zhao, 2012).

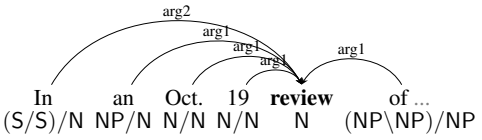


Figure 2: An example to illustrate the argument-centric view.

### 3.3 Argument-Centric Factorization

The syntactic principle for tree annotation treats the dependency relations between two words as syntactic projection. In another word, the head determines the syntactic category of the whole structure. The (type-logical) semantic principle determines a dependency according the types of the two words. The two kinds of dependency are coherent but not necessarily the same. In particular, an adjunct is a syntactic dependent but usually a semantic predicate of its syntactic head. Figure 2 gives an example to illustrate the idea. The argument in focus is "review" that is the complement of the preposition "in." The direction of this semantic dependency is the same to its corresponding syntactic dependency. Other predicates that semantically govern "review" are actually its modifiers, so the direction of these semantic dependencies are the opposite of their syntactic counterparts. It is important to capture head-complement-adjunct relations by putting all predicates of one particular argument together.

Similar to the predicate-centric model, we treat the graph fragment involved by each argument as independent, and capture the relationships among all predicates that governs the same argument using a Markov model. In the definition of predicate-centric model, if we exchange predicates and arguments, then we get our *argument-centric model*. Formally, we define

$$\boldsymbol{y}^a_{\curvearrowright j} = \{y(i,j) : i \in \{0, \cdots, n\}, j \neq i\}.$$

Let $p_1, \cdots, p_m$ be the sequence of the predicates (in linear word order) that semantically governs the word $j$ under $\boldsymbol{y}^a_{\curvearrowright j}$. A *$k$-th order argument-centric model* scores the dependency graph as

$$\begin{aligned} f^a(y) &= \sum_{j=1}^{n} f^a_j(\boldsymbol{y}^a_{\curvearrowright j}) \\ &= \sum_{j=1}^{n} \sum_{i=1}^{m+k-1} \theta_a^\top \Phi_a(p_{i-(k-1)}, ..., p_i, j, \mathbf{w}, \mathbf{p}) \end{aligned}$$

Similarly, we define the initial and end states for $p_i$ ($i < 0$ or $i \geq m+1$).

### 3.4 Tree Approximation Model

Tree structures exhibit many computationally-good properties, and have been widely applied to model linguistic, especially syntactic, structures. Tree-structured representation is an essential prerequisite for both the parsing algorithms and the machine learning methods in state-of-the-art syntactic dependency parsers. The CCG dependency graphs are not but look very much like trees. We thus argue that a tree-centric model can on one hand capture some topologically essential characteristics and on the other hand benefit from mature tree parsing techniques.

To this end, we propose tree approximation to obtain CCG sub-graphs under the factorization using tree parsing algorithms. In particular, we introduce an algorithm to associate every graph with a projective dependency tree, which we call *weighted conversion*. The tree reflects partial information about the corresponding graph. In this algorithm, we assign heuristic weights to all possible edges, and then find the tree with maximum weights. The key idea behind is to find a tree frame of a given graph. Given an arbitrary CCG graph, the conversion is perhaps imperfect in the sense that information about a small portion of edges is "lost." As a result, our tree approximation model can only generate partial graphs. Nevertheless, we will show (in Section 3.5 and 4.2) that such a model can be combined with predicate- and argument-centric factorization models in an elegant way.

#### 3.4.1 Weighted Conversion

We assign weights to all the possible edges, i.e. all pairs of words, and then determine which edges to be kept by finding the maximum spanning tree. More formally, given a graph $\boldsymbol{y} = \{y(i,j)\}$, each possible edge $(i,j)$ is assigned a heuristic weight $\omega(i,j)$. The maximum spanning tree $\boldsymbol{t} = \{t(i,j)\}$ contains the maximum sum of values of edges:

$$\boldsymbol{t}^{\max} = \arg\max_{\boldsymbol{t}} \sum_{(i,j)} t(i,j)\omega(i,j)$$

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

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

After all edges are weighted, we can use maximum spanning tree (MST) algorithms to get the converted tree. To get the projective tree, we choose Eisner's algorithm. However, the obtained tree must be labeled in order to encode the original graph. Here we introduce a label vector $\mathbf{l} = \{l(i,j)\}$. For each $(i,j) \in \mathcal{I}$, we assign a label $l(i,j)$ to edge $(i,j)$ as follows.

**Case** $y(i,j) = 1$**:** label "X";

**Case** $y(i,j) = 0 \wedge y(j,i) = 1$**:** label "X∼R";

**Case** $y(i,j) = 0 \wedge y(j,i) = 0$**:** label "None".

We can convert the *labeled* tree back to graph and obtain $\boldsymbol{y}^t$. Tough some edges are lost during the conversion, a lot more are kept. In fact, according to our evaluation, $92.74\%$ of edges in the training set are retained after conversion.

#### 3.4.2 Factorizing Trees

We use the tree parsing model proposed in (Bohnet, 2010) to score the converted trees. The model factorizes a tree into 1st-order and 2nd-order factors. When decoding, the model searches for a tree with the best score. The score defined for graphs as well as trees is

$$\begin{aligned} f^t(\boldsymbol{y}^t) &= g^t(\boldsymbol{t}, \mathbf{l}) = \theta_t^\top \Phi_t(s, \boldsymbol{t}, \mathbf{l}) \\ &= \theta_t^{1\top} \Phi_t^1(s, \boldsymbol{t}, \mathbf{l}) + \theta_t^{2\top} \Phi_t^2(s, \boldsymbol{t}, \mathbf{l}) \\ &= \Big( \sum_{(i,j)\in\mathcal{I}} t(i,j) \theta_t^{1\top} \Phi_t^1(l(i,j), \mathbf{w}, \mathbf{p}) \Big) \\ &\quad + \theta_t^{2\top} \Phi_t^2(s, \boldsymbol{t}, \mathbf{l}), \end{aligned}$$

where $\Phi_t^1$ is the 1st-order features and $\Phi_t^2$ is the 2nd-order features.

### 3.5 Parsing as Optimization

Motivated by linguistic properties of the semantics-oriented CCG dependencies, we have designed three single factorization models from heterogeneous views. Our single models exhibit different predictive strengths considering that they are designed to capture different properties separately. Integrating them can generate better graphs, but is provably hard. To this end, we formulate the parsing problem as the following constrained optimization problem.

$$\begin{aligned} \text{maximize} \quad & f^p(\boldsymbol{y}^p) + f^a(\boldsymbol{y}^a) + f^t(\boldsymbol{y}^t) \\ \text{subject to} \quad & y^p(i,j) = y^a(i,j), \\ & y^p(i,j) \geq y^t(i,j), \\ & y^a(i,j) \geq y^t(i,j) \text{ for all } (i,j) \end{aligned}$$

The equality constraint says that the graph given by the predicate- and the argument-centric model must be identical, while the inequality constraints say that the frame of graph given by the tree approximation model must be a subgraph of what is given by the first two models.

## 4 Decoding

### 4.1 Easiness and Hardness of Decoding

The three factorization models are all solvable in polynomial time. The predicate-centric model and the argument-centric model can be decoded using dynamic programming. We provide the detailed description of such an algorithm in our supplementary note. The decoding method for $k$-th ($k \geq 2$) order model costs time of $O(n^{k+1})$ where $n$ is the length of the sentence. The tree approximation model can re-use existing dependency tree parsing algorithms.

Unfortunately, the exact joint decoding of 2nd-order predicate- and argument-centric models is already NP-hard, not to mention other model combinations. The following gives a brief proof for the problem of combining the 2nd-order predicate- and argument-centric models.

*Proof.* Formally, we want to find a graph $\boldsymbol{y}$ which maximizes $F(\boldsymbol{y}) = f^p(\boldsymbol{y}) + f^a(\boldsymbol{y})$. We can design the feature function $\Phi_a$ and the parameter $\theta_a$, such

that for all $1 \leq i_1 \leq i_2 \leq n$,

$$\begin{cases} \theta_a^\top \Phi_a(0, i_1, j, \mathbf{w}, \mathbf{p}) = 0 \\ \theta_a^\top \Phi_a(i_1, n+1, j, \mathbf{w}, \mathbf{p}) = 0 \\ \theta_a^\top \Phi_a(i_1, i_2, j, \mathbf{w}, \mathbf{p}) = -\infty \\ \theta_a^\top \Phi_a(0, n+1, j, \mathbf{w}, \mathbf{p}) = -\infty \end{cases}$$

where $n$ is the length of the sentence. Note that those 4 equations make the nodes except the root node in the optimal graph each have exactly one incoming edge. So the problem of finding a tree $\boldsymbol{t}$ maximizing $f^p(\boldsymbol{t})$ is reduced to this problem. Moreover, the NP-hard problem 3DM can be reduced to the problem of finding a tree $\boldsymbol{t}$ maximizing $f^p(\boldsymbol{t})$ (see McDonald and Pereira (2006)), leading to the NP-hardness of both of the problems. □

## 4.2 Decoding via Dual Decomposition

To solve the joint decoding problem, optimization techniques based on decomposition with coupling variables are applicable. In this paper, we propose to solve it via dual decomposition. The experiment results show that though not always, we can obtain the optimal solution most of time. To simplify the description, we only consider the 2nd-order case for all three models.

### 4.2.1 Lagrangian Relaxation

Notice that $y^p(i,j) \geq y^t(i,j)$ can be written as

$$y^p(i,j) = \begin{cases} 1, \text{ if } y^t(i,j) = 1; \\ \text{arbitrary, if } y^t(i,j) = 0. \end{cases}$$

So the constraint can be written as $A^p \boldsymbol{y}^p + A^a \boldsymbol{y}^a + A^t \boldsymbol{y}^t = 0$, where

$$A^p = \begin{bmatrix} I \\ D_{\boldsymbol{y}^t} \\ \mathbf{0} \end{bmatrix} A^a = \begin{bmatrix} -I \\ \mathbf{0} \\ D_{\boldsymbol{y}^t} \end{bmatrix} A^t = \begin{bmatrix} \mathbf{0} \\ -D_{\boldsymbol{y}^t} \\ -D_{\boldsymbol{y}^t} \end{bmatrix}$$

$I$ is the identity matrix and $D_{\boldsymbol{y}^t}$ is a diagonal matrix whose main diagonal is the vector $\boldsymbol{y}^t$.

The Lagrangian of the optimization problem is

$$\begin{aligned} \mathcal{L}(\boldsymbol{y}^p, \boldsymbol{y}^a, \boldsymbol{y}^t; u) &= f^p(\boldsymbol{y}^p) + f^a(\boldsymbol{y}^a) + f^t(\boldsymbol{y}^t) \\ &\quad + u^\top (A^p \boldsymbol{y}^p + A^a \boldsymbol{y}^a + A^t \boldsymbol{y}^t), \end{aligned}$$

where $u$ is the Lagrangian multiplier.

Omitting the constraints, the dual objective is

$$\begin{aligned} \mathcal{L}(u) &= \max_{\boldsymbol{y}^p, \boldsymbol{y}^a, \boldsymbol{y}^t} \mathcal{L}(\boldsymbol{y}^p, \boldsymbol{y}^a, \boldsymbol{y}^t; u) \\ &= \max_{\boldsymbol{y}^p} (f^p(\boldsymbol{y}^p) + u^\top A^p \boldsymbol{y}^p) \\ &\quad + \max_{\boldsymbol{y}^a} (f^a(\boldsymbol{y}^a) + u^\top A^a \boldsymbol{y}^a) \\ &\quad + \max_{\boldsymbol{y}^t} (f^t(\boldsymbol{y}^t) + u^\top A^t \boldsymbol{y}^t) \end{aligned}$$

Let $\mathcal{L}^*$ be the maximized value of $\mathcal{L}(\boldsymbol{y}^p, \boldsymbol{y}^a, \boldsymbol{y}^t; u)$ subjected to the constraints, then $\mathcal{L}^* = \min_u \mathcal{L}(u)$, according to the duality principle.

### 4.2.2 Decoding Algorithm

There are two challenges in solving the dual problem. One challenge is to find the minimum value of the dual objective. For this, we can use subgradient method, as is demonstrated in Algorithm 1. The other is the evaluation of $\mathcal{L}(u)$. For this, we decompose the dual objective into three optimization problems. Let $B^p = u^\top A^p$, $B^a = u^\top A^a$, $B^t = u^\top A^t$, and

$$C_{\mathbf{l}}^t(i,j) = \begin{cases} B^t(i,j), \text{ if } l(i,j) = \mathrm{X}; \\ B^t(j,i), \text{ if } l(i,j) = \mathrm{X} \sim \mathrm{R}; \end{cases}$$

we can just redefine

$$\begin{aligned} f_i^p(\boldsymbol{y}_{i\curvearrowright}) &= \sum_{j=1}^{m+1} \big(\theta_p^\top \Phi_p(a_{j-1}, a_j, i, \mathbf{w}, \mathbf{p}) \\ &\quad + B^p(i,j)\big) \\ f_j^a(\boldsymbol{y}_{\curvearrowright j}) &= \sum_{i=1}^{m+1} \big(\theta_a^\top \Phi_a(p_{i-1}, p_i, j, \mathbf{w}, \mathbf{p}) \\ &\quad + B^a(i,j)\big) \\ f^t(\boldsymbol{y}) &= \Big( \sum_{(i,j)\in\mathcal{I}} t(i,j) \big(\theta_t^{1\top} \Phi_t^1(l(i,j), \mathbf{w}, \mathbf{p}) \\ &\quad + C_{\mathbf{l}}^t(i,j)\big)\Big) + \theta_t^{2\top} \Phi_t^2(s, \boldsymbol{t}, \mathbf{l}), \end{aligned}$$

and decode according to the new scores. In fact, this equals to attach some new weights to 1st-order factors, without changing the decoding algorithms for the subproblems. This nice property also allows using higher-order models for subproblems.

**Algorithm 1:** Joint decoding algorithm

**Initialization**: set $u^{(0)}$ to 0
**for** $k = 1$ **to** $K$ **do**
  $\boldsymbol{y}_{(k)}^p \leftarrow \arg\max_{\boldsymbol{y}} f^p(\boldsymbol{y}) + u_{(k)} A^p \boldsymbol{y}$
  $\boldsymbol{y}_{(k)}^a \leftarrow \arg\max_{\boldsymbol{y}} f^a(\boldsymbol{y}) + u_{(k)} A^a \boldsymbol{y}$
  $\boldsymbol{y}_{(k)}^t \leftarrow \arg\max_{\boldsymbol{y}} f^t(\boldsymbol{y}) + u_{(k)} A^t \boldsymbol{y}$
  **if** $A^p \boldsymbol{y}_{(k)}^p + A^a \boldsymbol{y}_{(k)}^a + A^t \boldsymbol{y}_{(k)}^t = 0$ **then**
    **return** $\boldsymbol{y}_a$
  $u_{(k)} \leftarrow u_{(k-1)}$
    $-\alpha^k (A^p \boldsymbol{y}_{(k)}^p + A^a \boldsymbol{y}_{(k)}^a + A^t \boldsymbol{y}_{(k)}^t)$

Algorithm 1 is our decoding algorithm. In every iteration, we first compute the optimal $\boldsymbol{y}$'s of

the three subproblems. If the $\boldsymbol{y}$'s satisfies the constraints, then we've find the optimal solution for the original problem. If not, we update the Lagrangian multiplier $u$, towards the negative subgradient. We initialized $u$ to be a zero vector, and use $\alpha^k$ to be the step length of each iteration. When we decode the subproblems, the $D_{\boldsymbol{y}^c}$ in $A^p$ and $A^a$ is derived from the $\boldsymbol{y}^c$ obtained in the current iteration.

We can also assign weights to different factorization models. If we choose to do so, the Lagrangian becomes

$$\mathcal{L}(\boldsymbol{y}^p, \boldsymbol{y}^a, \boldsymbol{y}^c; u) = w^p f^p(\boldsymbol{y}^p) + w^a f^a(\boldsymbol{y}^a) + w^c f^t(\boldsymbol{y}^t) + u^\top (A^p \boldsymbol{y}^p + A^a \boldsymbol{y}^a + A^t \boldsymbol{y}^t).$$

And the decoding of subproblems in our algorithm becomes

$$\begin{aligned} \boldsymbol{y}^p_{(k)} &\leftarrow \arg\max_{\boldsymbol{y}} f^p(\boldsymbol{y}) + \frac{1}{w^p} u_{(k)} A^p \boldsymbol{y}; \\ \boldsymbol{y}^a_{(k)} &\leftarrow \arg\max_{\boldsymbol{y}} f^a(\boldsymbol{y}) + \frac{1}{w^a} u_{(k)} A^a \boldsymbol{y}; \\ \boldsymbol{y}^t_{(k)} &\leftarrow \arg\max_{\boldsymbol{y}} f^t(\boldsymbol{y}) + \frac{1}{w^c} u_{(k)} A^t \boldsymbol{y}. \end{aligned}$$

The algorithm we give here is the joint decoding for all the three models. We can also decode using any two of them, and it is trivial to adapt the algorithm to the decoding.

### 4.3 Pruning

In order to improve the efficiency of the algorithm, we also do some pruning. One idea is that, in predicate-centric model, different type of predicates has different number of arguments. For example, the predicates POS-tagged "DT" each has only one argument in most cases. Therefore, we assign each POS-tag a max number of arguments. When decoding, we search at most those number of arguments instead of all the words in the sentence. This pruning method can also be applied to the argument-centric model. The other idea is that, some pairs of types never form a predicate-argument relation. So we can skip extracting feature of those POS-tag pairs, just take $-\infty$ to be their scores.

## 5 Evaluation and Analysis

### 5.1 Experimental Setup

CCGbank is a translation of the Penn Treebank into a corpus of CCG derivations (Hockenmaier and Steedman, 2007). CCGbank pairs syntactic derivations with sets of word-word dependencies which approximate the underlying functor-argument structure. Our experiments were performed using CCGBank which was split into three subsets for training (Sections 02-21), development testing (Section 00) and the final test (Section 23). We also use the syntactic dependency trees provided by the CCGBank to obtain necessary information for graph parsing. However, different from experiments in the CCG parsing literature, we use no grammar information. Neither lexical categories nor CCG derivations are utilized.

| | Devel. | Test |
|---|---|---|
| HMM Tagger | 96.74% | 97.23% |
| Transition-based Parser | 93.48% | 93.09% |
| Graph-based Parser | 93.47% | 93.19% |

Table 1: The accuracy of the POS tagger and the UAS of the syntax tree parsers.

All experiments were performed using automatically assigned POS-tags that are generated by a symbol-refined generative HMM tagger[1] (Huang et al., 2010), and automatically parsed dependency trees that are generated by our in-house implementation of the transition-based model presented in (Zhang and Nivre, 2011) as well as a 2nd-order graph-based parser[2] (Bohnet, 2010). The accuracy of these preprocessors is shown in Table 1. We ran 5-fold jack-knifing on the gold-standard training data to obtain imperfect dependency trees, splitting off 4 of 5 sentences for training and the other 1/5 for testing, 5 times. For each split, we re-trained the tree parsers on the training portion and applied the resulting model to the test portion.

Previous research on dependency parsing shows that structured perceptron (Collins, 2002) is one of the strongest discriminative learning algorithms. To estimate $\theta$'s of different models, we utilize the averaged perceptron algorithm. We implement our own the predicate- and argument-centric models. To perform tree parsing, we re-use the open-source implementation provided by the mate-tool. See the source code attached for details. We set iteration 5 to train predicate- and argument-centric models and 10 for the tree approximation model. To perform dual decomposition, we set the maximum iteration 200.

[1] www.code.google.com/p/umd-featured-parser/

[2] www.code.google.com/p/mate-tools/

| Tree | Model | UP | UR | UF | UEM |
|---|---|---|---|---|---|
| No | PC | 91.85 | 87.26 | 89.50 | 18.77 |
| | AC | 91.94 | 87.06 | 89.43 | 16.47 |
| | TA | 92.85 | 86.39 | 89.51 | 14.48 |
| | PC+AC | 93.84 | 88.18 | 90.93 | 23.05 |
| | PC+TA | 91.80 | 91.69 | 91.74 | 27.29 |
| | AC+TA | 90.19 | 92.88 | 91.52 | 25.51 |
| | PC+AC+TA | 93.01 | 92.08 | 92.54 | 32.83 |
| Gr | PC | 94.01 | 90.76 | 92.36 | 30.16 |
| | AC | 94.14 | 90.44 | 92.25 | 27.71 |
| | TA | 93.07 | 86.59 | 89.71 | 15.16 |
| | PC+AC | 94.66 | 91.09 | 92.84 | 33.19 |
| | PC+TA | 92.98 | 92.68 | 92.83 | 35.02 |
| | AC+TA | 92.47 | 93.13 | 92.80 | 33.93 |
| | PC+AC+TA | 93.66 | 92.73 | 93.19 | 37.64 |
| Tr | PC | 93.93 | 90.85 | 92.37 | 30.21 |
| | AC | 93.94 | 90.66 | 92.27 | 28.23 |
| | TA | 93.19 | 86.68 | 89.82 | 14.79 |
| | PC+AC | 94.58 | 91.14 | 92.83 | 31.94 |
| | PC+TA | 92.93 | 92.71 | 92.82 | 35.34 |
| | AC+TA | 92.33 | 93.16 | 92.74 | 33.61 |
| | PC+AC+TA | 93.50 | 92.73 | 93.11 | 37.69 |

Table 2: Parsing performance on the development data. The column "Tree" denotes the parsers that give dependency tree of development set: no tree (No), transition-based (Tr) or graph-based (Gr). "PC," "AC" and "TA" in the second column denotes the predicate-centric, the argument-centric and the tree approximation models, respectively.

### 5.2 Effectiveness of Data-driven Models

Table 2 summarizes parsing performance on development set with different configurations. We report unlabeled precision (UP), recall (UR), f-score (UF) as well as complete match (UEM). It can be clearly seen that the data-driven models obtains high-quality graphs with respect to token match. Even without any syntactic information (see the top block associated with "No Tree"), our parser with all three factorization models obtains an f-score of 92.5. when assisted by a syntactic parser, this figure goes up to over 93.1. If the predicate- or argument-centric model is applied by itself, either one can achieve a competitive accuracy, especially when syntactic features are utilized.

### 5.3 Effectiveness of Multiple Factorization

We use dual decomposition to perform joint decoding. First we combine the predicate-centric model and the argument-centric model. Compared to each single model, an error reduction of about 7% on f-score (UF) on average is achieved. Furthermore, we ensemble all the three models. If no syntactic features are extracted, the "TA" model brings in a remarkable further absolute gain of 1.02 with respect to token match. If syntactic features are used, the "PC" and "AC" models already achieves relatively good performance, and the "TA" model does not contribute much considering token match. The join of the tree approximation model lowers the precision, it increases the recall further, resulting in a modest improvement of the f-score. Nonetheless, the "TA" model still significantly improve the complete match metric. It is noticeable that in all setting, the "TA" model result in very significant boost in complete match.

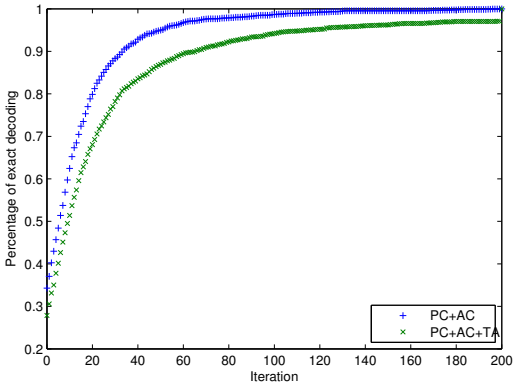


Figure 3: The exact decoding rate.

The dual decomposition does not guarantee to find an exact solution (in a limited number of iterations) in theory but usually works very well in practice. We calculate the percentage of finding exact decoding below $k$ iterations, and the result is show in Figure 3. The transition-based tree parser is utilized here. We can see that for most sentences, dual decomposition practically gives the exact solutions.

### 5.4 Importance of Tree Structures

Our factorization parser (with best experimental setting) does not utilize a grammar but do use syntactic information in the dependency formalism. In particular, the parser extracts the so-called path features from dependency trees. The syntactic trees is very importance to our parser, which provide a critical set of features for the predicate-centric and the argument-centric models. Without the syntactic trees, their performances decrease significantly. We try two different parsers to obtain the syntax tree parses. One is of the transition-based architecture, and the other graph-

| Tree | Model | UP | UR | UF | UEM |
|---|---|---|---|---|---|
| No | PC+AC+TA | 93.03 | 92.03 | 92.53 | 32.61 |
| Gr | PC+AC+TA | **93.71** | 92.72 | 93.21 | **38.14** |
| Tr | PC+AC+TA | 93.63 | **92.83** | **93.23** | 37.47 |
| Auli and Lopez | | 93.08 | 92.44 | 92.76 | - |
| Xu et al. | | 93.15 | 91.06 | 92.09 | 37.56 |

Table 3: Comparing the state-of-art with our models on test set.

based. The architecture of the syntactic tree parser does not affect the results much. The two tree parsers give identical attachment scores, and lead to similar graph parsing accuracy. This result is somehow non-obvious given that the combination of a graph-based and transition-based parser usually gives significantly better parsing performance (Nivre and McDonald, 2008; Torres Martins et al., 2008).

Although the target representation of our parser is general graphs rather trees, implicitly or explicitly using tree-structured information plays an essential role. Syntactic features are able to improve the f-score achieved by the "PC+AC" model from 90.9 to 92.8, while the "TA" model can bring in an absolute gain of 1.6. Note that the "TA" model does not utilize any syntactic tree information. The converted trees are automatically induced from the CCG graphs. Even when syntactic trees are available, the automatically induced trees can still significantly improve the complete match with respect to the whole sentence.

### 5.5 Comparison to the State-of-the-art

We compare our results with the best published CCG parsing performance obtained by the models presented in (Auli and Lopez, 2011) and (Xu et al., 2014)[3]. Auli and Lopez (2011) reported best numeric performance. The performance is evaluated on sentences that can be parsed by their model. Xu et al. (2014) reported the best published results for sentences with full coverage. All results on the test set is shown in Table 3. Even without any syntactic features, our parser achieves accuracies that are superior to Xu et al.'s parser and comparable to Auli and Lopez's system. When unlabeled syntactic trees are provided, our parser outperform the state-of-the-art.

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

## 6 Conclusion

In this paper, we have presented a factorization parser for building CCG-grounded dependency graphs. It achieves substantial improvement over the state-of-the-art. Perhaps surprisingly, our data-driven, grammar-free parser yields a superior accuracy to all CCG parsers in the literature. Our work indicates that high-quality data-driven parsers can be built for producing more general dependency graphs, rather than trees. Our method is also applicable to other deep dependency structures, e.g. HPSG predicate-argument analysis (Miyao et al., 2004), as well as other graph-structured semantic representations, e.g. Abstract Meaning Representations (Banarescu et al., 2013).

## Acknowledgement

The work was supported by NSFC (61300064), National High-Tech R&D Program (2015AA015403), NSFC (61331011) and NSFC (61170166).

## References

Michael Auli and Adam Lopez. 2011. Training a log-linear parser with loss functions via softmax-margin. In *Proceedings of EMNLP*, pages 333–343. Association for Computational Linguistics, Edinburgh, Scotland, UK.

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*, pages 178–186. Association for Computational Linguistics, Sofia, Bulgaria.

Emily M. Bender, Dan Flickinger, Stephan Oepen, and Yi Zhang. 2011. Parser evaluation over local and non-local deep dependencies in a large corpus. In *Proceedings of EMNLP*, pages 397–408. Association for Computational Linguistics, Edinburgh, Scotland, UK.

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)*, pages 89–97. Coling 2010 Organizing Committee, Beijing, China.

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*, pages 248–255. Association for Computational Linguistics, Prague, Czech Republic.

Stephen Clark and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of Coling 2004*, pages 282–288. COLING, Geneva, Switzerland.

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

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures using a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA.*, pages 327–334.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pages 1–8. Association for Computational Linguistics.

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

Timothy A. D. Fowler and Gerald Penn. 2010. Accurate context-free parsing with combinatory categorial grammar. In *Proceedings of ACL*, pages 335–344. Association for Computational Linguistics, Uppsala, Sweden.

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

Zhongqiang Huang, Mary Harper, and Slav Petrov. 2010. Self-training with products of latent variable grammars. In *Proceedings of EMNLP*, pages 12–22. Association for Computational Linguistics, Cambridge, MA.

Tracy Holloway King, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 dependency bank. In *In Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*, pages 1–8.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of ACL*, pages 1–11. Association for Computational Linguistics, Uppsala, Sweden.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of EMNLP*, pages 1288–1298. Association for Computational Linguistics, Cambridge, MA.

Xavier Lluís, Xavier Carreras, and Lluís Màrquez. 2013. Joint arc-factored parsing of syntactic and semantic dependencies. *TACL*, 1:219–230.

Xuezhe Ma and Hai Zhao. 2012. Fourth-order dependency parsing. In *Proceedings of COLING 2012: Posters*, pages 785–796. The COLING 2012 Organizing Committee, Mumbai, India.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, 19(2):313–330.

André F. T. Martins and Mariana S. C. Almeida. 2014. Priberam: A turbo semantic parser with second order features. In *Proceedings of SemEval 2014*, pages 471–476.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2007. Efficient hpsg parsing with supertagging and cfg-filtering. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1671–1676. Morgan Kaufmann publishers Inc., San Francisco, CA, USA.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. 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. Association for Computational Linguistics, Ann Arbor, Michigan.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of 11th Conference of the European Chapter of the Asso-*

*ciation for Computational Linguistics (EACL-2006)*, volume 6, pages 81–88.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of EMNLP*, pages 523–530. Association for Computational Linguistics, Vancouver, British Columbia, Canada.

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

Joakim Nivre, 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. Association for Computational Linguistics, Boston, Massachusetts, USA.

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

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

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

Alexander M. Rush and Michael Collins. 2011. Exact decoding of syntactic translation models through lagrangian relaxation. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 72–82.

Alexander M Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of EMNLP*, pages 1–11. Association for Computational Linguistics, Cambridge, MA.

Kenji Sagae, Yusuke Miyao, and Jun'ichi Tsujii. 2007. Hpsg parsing with shallow dependency constraints. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 624–631. Association for Computational Linguistics, Prague, Czech Republic.

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

Mark Steedman. 2000. *The syntactic process*. MIT Press, Cambridge, MA, USA.

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

André Filipe Torres Martins, Dipanjan Das, Noah A. Smith, and Eric P. Xing. 2008. Stacking dependency parsers. In *Proceedings of EMNLP*, pages 157–166. Association for Computational Linguistics, Honolulu, Hawaii.

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

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

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