The Covert Helps Parse the Overt

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

Abstract

This paper is concerned with whether deep syntactic information can help surface parsing, with a particular focus on empty categories. We design new algorithms to produce dependency trees in which empty elements are allowed, and evaluate the impact of information about empty category on parsing overt elements. Such information is helpful to reduce the approximation error in a structured parsing model, but increases the search space for inference and accordingly the estimation error. To deal with structure-based overfitting, we propose to integrate disambiguation models with and without empty elements, and perform structure regularization via joint decoding. Experiments on English and Chinese TreeBanks with different parsing models indicate that incorporating empty elements consistently improves surface parsing.

1 Introduction

In the last two decades, there was an increasing interest in producing rich syntactic annotations that are not limited to surface analysis. See, among others, (Callmeier, 2000; Kaplan et al., 2004; Clark and Curran, 2007; Miyao and Tsujii, 2008; Zhang et al., 2016). Such analysis, e.g. deep dependency structures (King et al., 2003), is usually coupled with grammars under deep formalisms, e.g. Combinatory Categorial Grammar (CCG; Steedman, 2000), Head-driven Phrase-Structure Grammar (HPSG; Pollard and Sag, 1994) and Lexical-Functional Grammar (LFG; Bresnan and Kaplan, 1982). Although deep grammar formalisms allow information beyond local construction to be constructed, it is still not clear whether such additional information is helpful for surface syntactic analysis. This is partly because analysis grounded on different grammar formalisms, e.g. HPSG and CFG, are not directly comparable.

In the Government and Binding (GB; Chomsky, 1981) theory, empty category is a key concept bridging S-Structure and D-Structure, due to its possible contribution to trace *movements*. Following the linguistic insights underlying GB, a traditional dependency analysis can be augmented with empty elements, viz. covert elements (Xue and Yang, 2013). See Figure 1 for an example. The new representation provides a considerable amount of deep syntactic information, while keeping intact all dependencies of overt words. Integrating both overt and covert elements in one unified representation provides an effective yet lightweight way to achieve deeper language understanding beyond surface syntax[1]. Even more important, this modest way to modify tree analysis makes possible fair evaluation of the influence of deep syntactic elements on surface parsing.

We study graph-based parsing models for this new representation with a particular focus on the impact of information about the covert on parsing the overt. The major advantage of the graph-based approach to dependency parsing is that its constrained factorization enables the design of polynomial time algorithms for decoding, especially for projective structures. Following GB, an empty element can be only a dependent. Furthermore, the number and distribution of empty elements in one sentence is highly constrained. These properties makes polynomial time decoding for joint empty element detection and dependency parsing still plausible. To exemplify our idea, we design novel second- and third-order algorithms for the

[1] In this paper, we arguably call dependencies among overt words only surface analysis.

Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), pages 343–353,
Vancouver, Canada, August 3 - August 4, 2017.

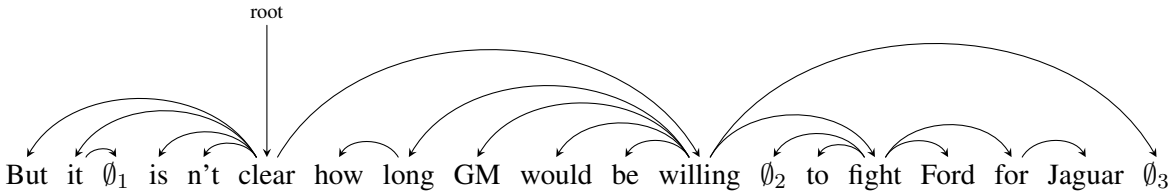

Figure 1: An example from PTB. The dependency structure is according to Stanford Dependency (de Marneffe et al., 2006). "$\emptyset$" denotes an empty element. "$\emptyset_1$" indicates an expletive construction; "$\emptyset_2$" indicates that the subject for *fight*, i.e. *GM*, is located in another place; "$\emptyset_3$" indicates a *wh*-movement.

new problem.

The influence of incorporating empty elements is twofold. On the one hand, the extra information enriches the structural information of the outputs, which is important to reduce the approximation error in a structured prediction problem. On the other hand, predicting empty elements increases the search space for decoding, and thus increases the difficulty of parameter estimation for disambiguation. Our experiments on English Penn TreeBank (PTB; Marcus et al., 1993) and Chinese TreeBank (CTB; Xue et al., 2005) shows that the second effect is prominant. The accuracy of predicting dependencies among overt words sometimes declines slightly.

To ensure that predicting the empty elements helps parse the overt, we need to reduce the new estimation error. To this end, we propose to integrate scores from parsing models with and without empty elements and perform *joint* decoding. The intuition is to leverage parameters estimated without empty elements as a backoff, which exhibit better generalization ability. We evaluate two joint decoders: One is based on chart merging and the other is based on dual decomposition. Experiments demonstrate that information about the covert improves surface analysis in this way. Accuracy evaluated using parsing models with different factorizations and on data sets from different languages is consistently improved. Especially, for those sentences in which there is no empty element, accuracy is improved too. This highlights the fact that empty category can help reduce the approximation error for surface analysis.

The remaining part of the paper is organized as follows. Section 2 is a brief introduction to the problem. Section 3 describes existing algorithms for parsing for overt words only, while Section 4 gives the details of our new algorithms for parsing with empty elements. Section 5 describes the details of the joint models as well as the decoding algorithms. Section 6 presents experimental results and empirical analyses. Section 7 concludes the paper.

2 Syntactic Analysis with Empty Category

In GB, empty categories are an important piece of machinery in representing the syntactic structure of a sentence. An empty category is a covert nominal element that is unpronounced, such as dropped pronouns and traces of dislocated elements. In treebanks, empty categories have been used to indicate long-distance dependencies, discontinuous constituents, and certain dropped elements (Marcus et al., 1993; Xue et al., 2005). Together with labeled brackets and function tags, they make up the full syntactic representation of a sentence.

Empty category is one key concept bridging S-Structure and D-Structure, given that they contain essential information to trace *movements*, i.e. the transformation procedure to convert a D-Structure to an S-Structure. When representing empty categories in dependency trees, we can use a null symbol to depict the idea that there is a mental category at the level being represented. See Figure 1 for an example.

Detecting empty elements is important to the interpretation of the syntactic structure of a sentence. For example, Chung and Gildea (2010) reported preliminary work that has shown a positive impact of automatic empty element detection on statistical machine translation. There are three strategies to find empty categories. Dienes and Dubey (2003) introduced a model that utilizes clues from word forms and POS tags to predict the existence of empty categories. In their method, syntactic parsing was treated as a next-step task and therefore had no influence on finding empty elements. Johnson (2002) and Xue and Yang

(2013) proposed to identify empty categories after syntactic parsing. Different from the above pre-processing strategy, their post-processing models can not use information about empty category to improve parsing. Cai et al. (2011) introduced an integrated model, where empty category detection and phrase-structure parsing are combined in a single model. They, however, did not report any improvement for parsing.[2]

Seeker et al. (2012) evaluted all above strategies to include empty nodes in dependency parsing for German and Hungarian. To predict both empty nodes and dependency relations, they enriched the information encoded in dependency labels. They showed that both pre-processing and integrated strategies failed to leverage empty categories to improve parsing. Especially, their pre-processing method significantly descreased parsing accuracy.

Although empty categories are very important in theory, it is still unclear that they can help parsing in practice.

3 The Existing Parsing Algorithms

Data-driven dependency parsing has received an increasing amount of attention in the past decade. Such approaches, e.g. transition-based (Yamada and Matsumoto, 2003; Nivre, 2008; Andor et al., 2016) and graph-based (McDonald, 2006; Torres Martins et al., 2009; Lei et al., 2014) models have attracted the most attention of dependency parsing in recent years. 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). A number of dynamic programming (DP) algorithms have been designed. Here we summarize the design of two widely used algorithms for second- and third-order factorization, since it is the basis of our new algorithms.

3.1 Algorithm 1: Sibling Factorization

Eisner (1996) introduced a widely-used DP algorithm for first-order parsing. Their algorithm includes two interrelated types of DP structures: (1) complete spans, which consist of a head-word and its descendents on one side, and (2) incomplete spans, which consist of a dependency and the region between the head and modifier. To include

[2] Comparing their numeric results with other papers', we find that their model does not result in improved parsing.

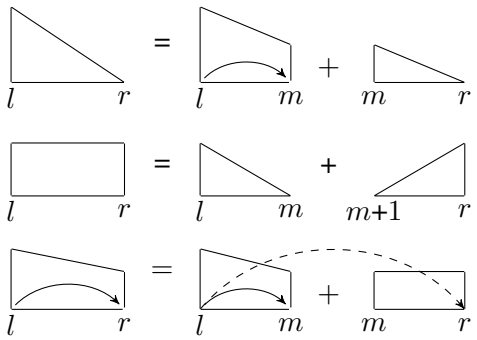

Figure 2: The DP structures and derivations of the standard sibling algorithm. Complete spans are depicted as triangles, incomplete spans as trapezoids, and sibling spans as rectangles. A new dependency is created by applying the last rule. Especially, the score associated with the last rule is determined by a sibling part. For brevity, we elide the right-headed versions.

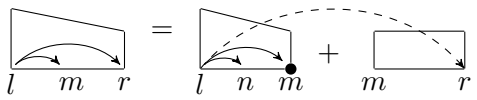

Figure 3: The modified construction rule for the tri-sibling algorithm.

second-order sibling parts, McDonald and Pereira (2006) extended Eisner's algorithm with a third structure, viz. (3) sibling spans, which represent the region between successive modifiers of same head. The second-order algorithm visits all the spans from bottom to top, finding the best combination of smaller structures to form a new one. Each type of span is created by recursively combining two smaller, adjacent spans. The DP structures and their constructions are specified graphically in Figure 2.

3.2 Algorithm 2: Tri-sibling Factorization

It is easy to extend the second-order sibling factorization to parts containing multiple siblings. For example, Koo and Collins (2010) introduced tri-sibling factorization in which a triple of three successive edges on the same side. Here, we consider parsing for tri-sibling factorization only. To this end, we augment the incomplete span structure with an internal index. The modified construction rule is specified graphically in Figure 3. Note that the presentation is slightly different from Koo and Collins's.

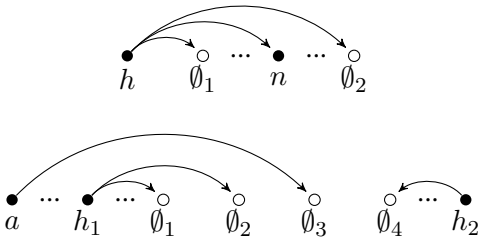

Figure 4: Prototypes of structures related to empty categories.

4 The New Algorithms

We propose three novel algorithms for the new parsing problem. We only consider projective structures. For sake of concision, we call an edge with an empty child empty edge, and call other edges normal ones. Not only normal edges but also empty edges do not cross with each other. We illustrate several properties of empty elements that are fundamental requirements of our algorithms, and then give details of our new algorithms.

4.1 Properties of empty elements

Two theoretical and empirical properties of empty elements results in the design of exact parsing algorithms for simultaneously predicting dependencies as well detecting empty elements.

1. According to GB, an empty element cannot be regarded as a head word.

2. There are very limited number of successive empty elements in between two successive overt words. Therefore, we can treat all successive empty elements governed by same head as one word. We can use the label associated to the corresponding empty edge to distinguish how many empty nodes are there.

According to the first property, we have two prototype structures of empty nodes and their associated edges. If there are two empty elements that are governed by the same head, the overt words in between them must be their siblings or dominated by their siblings. We graphically show this case as the upper figure in Figure 4. If there is a sequence of successive empty elements in between two overt words, the prototype structure is shown as the bottom figure in Figure 4.

Now we consider the emprical coverage of the second property on PTB and CTB. The coverage is evaluated using sentences in the training sets (as defined in Section 6). We show the statistics in Table 1. The length indicates the number of successive empty elements that are governed by the same overt word. At most three empty elements are next to each other.

Length	English	Chinese
1	54800	13115
2	2534	385
3	8	18
Total	57342	13518

Table 1: Coverage relative to the number of successive empty elements that have the same head.

4.2 Algorithm 3: Partial Sibling Model

4.2.1 DP Structures

Now we consider parsing with empty category detection by extending Algorithm 1. We consider six DP structures when we construct a tree with empty elements on a given span $[i, k]$ of vertices. See Figure 5 for graphical visualization. The first two are adapted in concord with Algorithm 1, and we introduce four new DP structures, transformed from incomplete constituent, which can manipulate the empty nodes. These new DP structures are explained below. Without loss of generality, we only illustrate the left-headed versions.

Overt-outside Incomplete Span. The rightmost word must be an overt word.

Overt-both Incomplete Span. Both the rightmost mode and the inner sibling of incomplete spans are overt words. An incomplete span structure is associated with an edge that crosses the whole span. We also care about its left sibling, and thus record it using an extra index. We call this sibling the inner sibling.

Covert-inside Incomplete Span. The rightmost word must be an overt word, while the inner sibling of incomplete span is a covert word.

Covert-ouside Incomplete Span. The rightmost word must be a covert word, while the inner sibling of incomplete span is an overt word.

Note that we already combine all successive empty nodes as one, so there is no *covert-both incomplete span*.

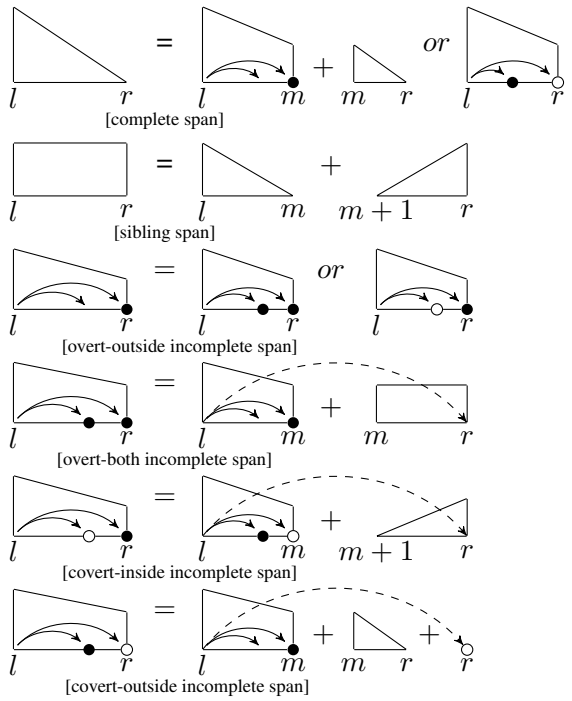

Figure 5: Graphic representations of new DP structures and their derivations of Algorithm 3. For brevity, we elide the right-headed versions.

4.2.2 Construction Rules

Figure 5 provides a graphical specification of the construction of all six DP structures. The following is the explaination for each construction rule.

1. The rightmost child of the head in a complete span may be an empty node. If so, the empty node must located at the boundary, because no empty node can be a head. Therefore a complete span itself is a covert-outside incomplete span. Otherwise, the rightmost child separates the complete span into an overt-outside incomplete span and another smaller complete span.

2. A sibling span is decomposed in the same way to Algorithm 1.

3. An overt-outside incomplete span is the extension of the incomplete span from the old algorithm. We consider two cases according to the type of the inner sibling. So it is either an overt-both or a covert-inside incomplete span.

4. Like a standard incomplete span in Algorithm 1, an overt-both incomplete span is made of an overt-outside incomplete span and a sibling span. A normal edge is created during the construction.

5. A covert-inside incomplete span is built from a covert-outside incomplete span and a complete span in the opposite direction rather than a sibling span. This is because the empty node does not have any child. A normal edge is also created here.

6. A covert-outside incomplete span is made up of an overt-outside incomplete span, an adjecent complete span and a new covert word. In this step, we add a new empty element as well as an empty edge.

4.2.3 Complexity

The set of complete or sibling spans has $O(n^2)$ elements, while the set of each type of incomplete spans has $O(n^2)$ elements. Therefore, the space requirement is of $O(n^2)$. To build a new DP structure in any type, we either change the type of an existing DP structure or search for the best position to separate the whole structure into two parts. The second case is worse and needs time of $O(n^3)$. As a result, Algorithm 3 runs in time of $O(n^3)$.

4.3 Algorithm 4: Full Sibling Model

Now consider the difference between Algorithm 1 and 3. It is easy to figure out that not all sibling factors summed by Algorithm 1 are included by Algorithm 3. Specifically, if two normal edges that are adjacent to each other (say e_1 and e_2) are inserted with an empty edge, e_1 and e_2 are taken as a sibling part by Algorithm 1 but not 3. Now we are going to modify Algorithm 3 to include all such sibling parts. To this end, we modify the covert-inside span structure as well as its the construction rule. In particular, we explictly utilize the index of the inner child provided by a covert-inside span. Figure 6 gives a specification.

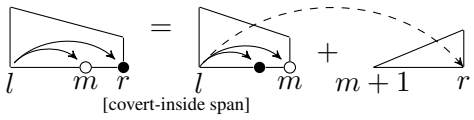

Figure 6: The modified construction rule for overt-both incomplete span in Algorithm 4. For brevity, we elide the right-headed versions.

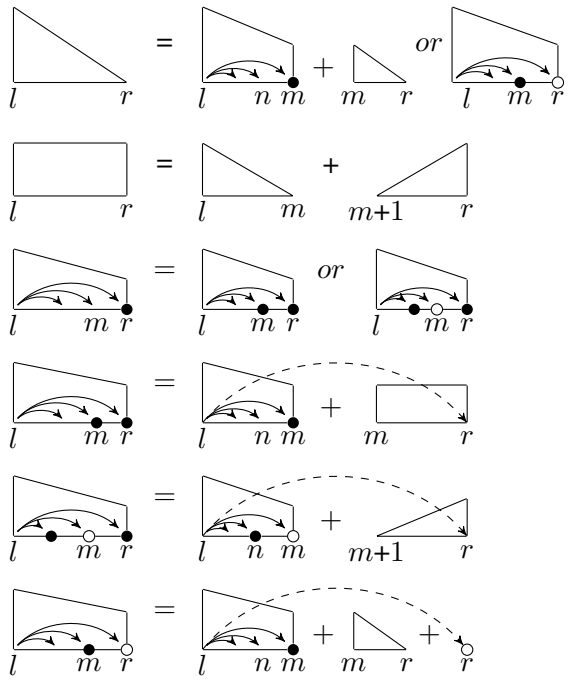

Figure 7: Graphic representations of new DP structures and their derivations of Algorithm 5. For brevity, we elide the right-headed versions.

The modification of the construction rule for the covert-inside incomplete span increases the complexity. The time and space requirements of Algorithm 4 are $O(n^4)$ and $O(n^3)$ respectively, because we must consider the position of the inner sibling, viz. m, in Figure 6.

4.4 Algorithm 5: Partial Tri-sibling Model

Previous work shows that it is relatively easy to extend the second-order sibling factorization to parts containing multiple siblings for standard parsing. It is similar when empty elements are taken into account. Adding the index of one more inner modifier to all incomplete span structures allows tri-sibling features to be calculated. We sketch the idea in Figure 7.

We add one more index to all the four incomplete DP structures in Algorithm 3. The time and space complexity are increased by a factor of $O(n)$. The analysis of the complexity of Algorithm 5 is similar to Algorithm 3. In short, Algorithm 5 runs in time $O(n^4)$ with a space requirement of $O(n^3)$. We can also extend Algorithm 5 to a full version, like what we have done for sibling models. The time complexity will go up to $O(n^5)$, which makes the algorithm somehow impractical.

5 Structure Regularization via Joint Decoding

We can see from the definition of the extended algorithms that the search space for decoding is significantly increased. This results in a side effect for practical parsing. Given the limit of available annotations for training, searching for more complicated structures in a larger space is harmful to the generalization ability in structured prediction (Sun, 2014). Incorporating empty elements significantly increases the difficulty for parameter estimation, and therefore it is harder to find a good disambiguation model. To control structure-based overfitting, we propose a new way to perform structure regularization: combining the two score functions learned from models with and without empty elements.

We formalize the idea as follows. Consider a sentence $s = w_1 w_2 \cdots w_n$. We denote the *index set* of all possible dependencies as $\mathcal{I} = \{(i,j)|i,j \in \{1,\cdots,n\}, i \neq j\}$. 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 there is an arc $i \rightarrow j$ in the graph, 0 otherwise. Let $\mathcal{Y}$ denote the set of all possible $\boldsymbol{y}$. We use another index set $\mathcal{I}' = \{(i,j)|i,j \in \{1,\cdots,n+1^2\}\}$, where $i > n$ indicates an empty node. Then a dependency parse with empty nodes can be represented as a vector similar to $\boldsymbol{y}$:

$$\boldsymbol{z} = \{z(i,j) : (i,j) \in \mathcal{I}'\}.$$

Let $\mathcal{Z}$ denote the set of all possible $\boldsymbol{z}$. Assume that $f : \mathcal{Y} \rightarrow \mathbb{R}$ and $g : \mathcal{Z} \rightarrow \mathbb{R}$ assign scores to parse trees without and with empty elements. A reasonable model to integrate f and g is to find the optimal parse by solving the following optimization problem:

$$\begin{array}{ll} \text{max.} & \lambda f(\boldsymbol{y}) + (1-\lambda) g(\boldsymbol{z}) \\ \text{s.t.} & \boldsymbol{y} \in \mathcal{Y}, \boldsymbol{z} \in \mathcal{Z} \\ & y(i,j) = z(i,j),\ \forall (i,j) \in \mathcal{I} \end{array} \quad (1)$$

λ is a weight for combining scores. We use the validation data to get an appropriate value for λ[3].

[3]Given the similarity of the parsing models with and without empty elements, $\lambda = 0.5$ usually achieves optimal performance.

$\mathbf{u}^{(0)} \leftarrow 0$
for $k \leftarrow 0..T$ **do**
  $\boldsymbol{y} \leftarrow \arg\max_{\boldsymbol{y} \in \mathcal{Y}} (f(\boldsymbol{y}) + \sum_{i,j} u(i,j) y(i,j))$
  $\boldsymbol{z} \leftarrow \arg\max_{\boldsymbol{z} \in \mathcal{Z}} (g(\boldsymbol{z}) - \sum_{i,j} u(i,j) z(i,j))$
  if $\forall (i,j) \in \mathcal{I}, y(i,j) = z(i,j)$ **then**
    return $\boldsymbol{z}$
  else
    $\mathbf{u}^{(k+1)} \leftarrow \mathbf{u}_{(k)} - \alpha^{(k)}(\boldsymbol{y} - \boldsymbol{z})$
return $\boldsymbol{z}$

Figure 8: Joint decoding based on dual decomposition.

5.1 Chart Merging

The optimization problem (1) can be solved using Algorithm 3 to 5. For example, if we try to combine models coupled with Algorithm 1 and 3, or Algorithm 1 and 4, we can merge the local scores of all sibling parts and then apply Algorithm 4 for solutions. Note that Algorithm 3 here cannot produce the exact solution. If we try to combine models coupled with Algorithm 2 and 5, we can use an algorithm of which the time complexity is $O(n^5)$. We have mentioned such an algorithm at the end of Section 4.4. Similar to sibling factorization, Algorithm 5 can only produce approximate solutions.

5.2 Dual Decomposition

The chart merging method can be applied to algorithms that have highly coherent DP structures. Dual decomposition is an alternative yet more flexible method for solving the optimization problem (1). Heterogeneous models can be combined, and for the majority of input sentences exact solutions can be found in a few iterations. We sketch the solution as follows.

The Lagrangian of (1), i.e. $\mathcal{L}(\boldsymbol{y}, \boldsymbol{z}; \mathbf{u})$, is

$$f(\boldsymbol{y}) + g(\boldsymbol{z}) + \sum_{(i,j) \in \mathcal{I}} u(i,j)(y(i,j) - z(i,j))$$

where $\mathbf{u}$ is the Lagrangian multiplier. Then the dual is

$$\begin{aligned}
\mathcal{L}(\mathbf{u}) &= \max_{\boldsymbol{y} \in \mathcal{Y}, \boldsymbol{z} \in \mathcal{Z}} \mathcal{L}(\boldsymbol{y}, \boldsymbol{z}; \mathbf{u}) \\
&= \max_{\boldsymbol{y} \in \mathcal{Y}} (f(\boldsymbol{y}) + \sum_{(i,j) \in \mathcal{I}} u(i,j) y(i,j)) \\
&\quad + \max_{\boldsymbol{z} \in \mathcal{Z}} (g(\boldsymbol{z}) - \sum_{(i,j) \in \mathcal{I}} u(i,j) z(i,j))
\end{aligned}$$

We instead try to find the solution for $\min_{\mathbf{u}} \mathcal{L}(\mathbf{u})$. By using a subgradient method for this optimization problem, we have another joint decoding algorithm, as shown in Figure 8.

		#{Sent}	#{Overt}	#{Covert}
En	train	38667	909114	57342
	test	2336	54242	3447
Ch	train	8605	193417	13518
	test	941	21797	1520

Table 2: Numbers of sentences, overt and covert elements in training and test sets.

6 Experiments

6.1 Data Sets

We conduct experiments on both English and Chinese treebanks. In particular, PTB and CTB are used. Because PTB and CTB are phrase-structure treebanks, we need to convert them into dependency annotations. To do so, we use the tool provided by Stanford CoreNLP to process PTB, and the tool provided by Xue and Yang (2013) to process CTB 5.0. We use gold-standard POS to derive features for disambiguation.

To simpify our experiments, we preprocess the obtained dependency tree in the following way.

1. We combine successive empty elements with identical head into one new empty node which is still linked to the common head word.

2. Because the high-order algorithm spends is very expensive, we only use relatively short sentence. Here we only keep sentences less than 64 tokens.

3. We focus on unlabeled parsing.

The statistics of the data after cleaning is shown in Table 2

We use standard training, validation, and test splits to facilitate comparisons. Accuracy is measured with unlabeled attachment score for all overt words (UAS_o): the percentage of overt words with the correct head. We are also concerned with the prediction accuracy for empty elements. To evaluate performance on empty nodes, we consider the correctness of empty edges. We report the percentage of empty words in right slot with correct head. The i-th slot in the sentence means that the position immediately after the i-th concrete word. So if we have a sentence with length n, we get $n + 1$ slots.

$\mathrm{f}_{\mathrm{uni}}(X)$: $X.w$, $Y.p$, $X.w \circ X.p$
$\mathrm{f}_{\mathrm{bi}}(X,Y)$: $X.wp \circ Y.w$, $X.wp \circ Y.p$, $X.w \circ Y.wp$, $X.p \circ Y.wp$, $X.wp \circ Y.wp$
$\mathrm{f}_{\mathrm{context}}(X,Y)$: $X.p \circ Y.p \circ X_1.p \circ Y_{-1}.p$, $X.p \circ Y.p \circ X_{-1}.p \circ Y_{-1}.p$, $X.p \circ Y.p \circ X_1.p \circ Y_1.p$, $X.p \circ Y.p \circ X_{-1}.p \circ Y_1.p$ $X.p \circ Y.p \circ Z.p$, Z is token between X and Y
$\mathrm{f}_{\mathrm{sib}}(X,Y)$: $X.w \circ Y.w$, $X.w \circ Y.p$, $X.p \circ Y.w$, $X.p \circ Y.p$
$\mathrm{f}_{\mathrm{sib}}(X,Y,Z)$: $X.p \circ Y.p \circ Z.p$
$\mathrm{f}_{\mathrm{tsib}}(X,Y,Z,W)$: $X.w \circ Y.w \circ Z.p \circ W.p$, $X.w \circ Y.p \circ Z.w \circ W.p$, $X.w \circ Y.p \circ Z.p \circ W.w$, $X.p \circ Y.w \circ Z.w \circ W.p$, $X.p \circ Y.w \circ Z.p \circ W.w$, $X.p \circ Y.p \circ Z.w \circ W.w$, $X.w \circ Y.p \circ Z.p \circ W.p$, $X.p \circ Y.w \circ Z.p \circ W.p$, $X.p \circ Y.p \circ Z.w \circ W.p$, $X.p \circ Y.p \circ Z.p \circ W.w$

Table 3: Feature template functions.

6.2 Statistical Disambiguation

In the context of data-driven parsing, we still need an extra disambiguation model for building a practical parser. As with many other parsers, we employ a global linear model. To estimate parameters, we utilize the averaged perceptron algorithm (Collins, 2002). Developing features has been shown crucial to advancing the state-of-the-art in dependency parsing. We adopt features from previous work.

We refer to the head/child of the arc as h/c, the k-th inner split point as m_k, and the grand point as g. We list features selected by different algorithm as follows, and all following features should be concatenated with direction and distance of the arc.

- arc features: $\mathrm{f}_{\mathrm{uni}}(h)$, $\mathrm{f}_{\mathrm{uni}}(c)$, $\mathrm{f}_{\mathrm{bi}}(h,c)$, $\mathrm{f}_{\mathrm{context}}(h,c)$.
- sibling features: $\mathrm{f}_{\mathrm{sib}}(c,m_0)$, $\mathrm{f}_{\mathrm{sib}}(h,c,m_k)$.
- tri-sibling features: $\mathrm{f}_{\mathrm{sib}}(h,c,m_1)$, $\mathrm{f}_{\mathrm{tsib}}(h,c,m,m_1)$.

6.3 Results of Individual Models

Table 4 lists the accuracy of individual models coupled with different decoding algorithms on the test sets. We focus on the prediction for overt words only. Models coupled with Algorithm 1, 3 and 4 are second-order models, while with 2 and 5 third-order ones. When we take into account empty categories, more information is available. The empirical results suggest that deep linguistic information does not necessarily help surface analysis.

Algo	English	Chinese
1	91.73	89.16
3	91.70 (−0.03)	89.20 (+0.04)
4	91.72 (−0.01)	89.28 (+0.12)
2	92.23	90.00
5	92.41 (+0.18)	89.82 (−0.18)

Table 4: UAS_o of different individual models on test data. The upper and bottom blocks present results obtained by sibling and tri-sibling models respectively.

	Algo	English	Chinese
CM	1+3	91.94 (+0.21)	89.53 (+0.37)
	1+4	91.88 (+0.15)	89.44 (+0.28)
DD	1+3	91.96 (+0.23)	89.53 (+0.37)
	1+4	91.94 (+0.21)	89.53 (+0.37)
CM	2+5	92.60 (+0.37)	90.35 (+0.35)
DD	2+5	92.71 (+0.48)	90.38 (+0.38)

Table 5: UAS_o of different joint decoding models on test data. "CM" and "DD" are short for joint decoders based on chart merging and dual decomposition respectively. The upper and bottom blocks present results obtained by sibling and tri-sibling models respectively. All improvements are statistically significant.

6.4 Results of Joint Decoding

Table 5 lists the accuracy of different joint decoding models on the test sets. We can see that the joint decoding framework is effective to deal with structure-based overfitting. This time, the accuracy of analysis for overt words is consistently improved across a wide range of conditions. Especially, the third-order model is improved more. We use the Hypothesis Tests method (Berg-Kirkpatrick et al., 2012) to evaluate the improvements. When the *p-value* is set to 0.05, all improvements in Figure 5 is statistically significant.

We separate all sentences in test data set into two subsets: One contains sentences that have no

Algo	English		Chinese	
	−EC	+EC	−EC	+EC
3	92.50	91.53	90.92	88.60
1+3	92.83	91.77	91.12	88.97
4	92.82	91.48	91.29	88.58
1+4	92.84	91.74	91.10	88.98
5	93.68	92.13	92.00	89.06
2+5	93.99	92.43	92.10	89.77

Table 6: UAS_o evaluated using different types of sentences. Dual decomposition is used for joint decoding.

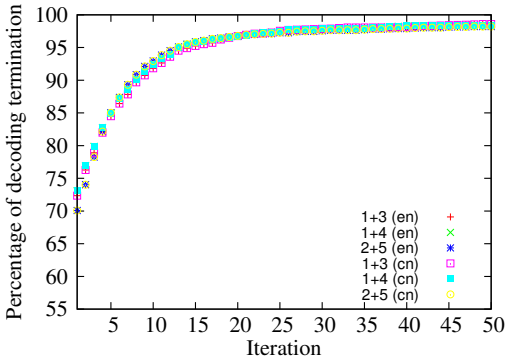

Figure 9: The exact decoding rate on development data.

empty elements and the other contains other sentences. The accuracy evaluated on the two sets are summarized in Table 6. For those sentences in which there is no empty element, accuracy is improved as well. This indicates that empty category can help reduce the approximation error for surface analysis.

6.5 Efficiency of Joint Decoding

One thing worth noting is that the chart merging method with an approximate decoder is comparable to other more complex solutions that produce exact or nearly exact results. Chart merging with approximate decoders only modifies scores assigned to secord- or third-order parts, while keeping intact the decoding procedure. As a result, the efficiency of approximate chart merging is comparable to individual models.

Dual decomposition-based joint decoder iteratively calls individual decoders. Due to the similarity of models with and without empty elements, this iteration procedure usually terminates very fast. We calculate the percentage of finding exact decoding below k iterations, and the result is show in Figure 9. For most sentences, dual decomposition practically gives the exact solutions in a few iterations. One advantage relevant is that such a decoder can integrate parsing models that are somehow heterogeneous. Refer to (Koo et al., 2010) for example.

7 Discussion and Conclusion

Can deep syntactic information help surface parsing, which is the mainstream focus of NLP research. In this paper, we investigate this topic under the umbrella of Transformational Grammar, GB in particular. We focused on empty category augmented dependency analysis. We demonstrate that on the one hand deep information helps reduce the approximation error for traditional (surface) parsing, while on the other hand traditional parsing helps reduce the estimation error for deep parsing. Coupling surface and deep information in an appropriate way is able to produce better syntactic analysis. A natural avenue for further research would be the integrating parsing models under deep and shallow grammar formalisms.

In addition to empty category detection, empty categories should be linked to an overt element if possible. Take the second empty element in Figure 1 for example. The information about its existance is valuable, but knowing it acturally refers to *GM* is more helpful. However, adding such coreference information makes the syntactic representation no long trees and thus brings along new challenges for designing algorithms. How to deal with empty category detection and resolution in one unified model? It would be an interesting topic for future investigation.

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). Weiwei Sun is the corresponding author.

References

Daniel Andor, 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)*. Association for Computational

Linguistics, Berlin, Germany, pages 2442–2452. http://www.aclweb.org/anthology/P16-1231.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, Jeju Island, Korea, pages 995–1005. http://www.aclweb.org/anthology/D12-1091.

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

Shu Cai, David Chiang, and Yoav Goldberg. 2011. Language-independent parsing with empty elements. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, pages 212–216. http://www.aclweb.org/anthology/P11-2037.

Ulrich Callmeier. 2000. Pet. a platform for experimentation with efficient hpsg processing techniques. *Journal of Natural Language Engineering* 6(1):99–108.

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

Tagyoung Chung and Daniel Gildea. 2010. Effects of empty categories on machine translation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Cambridge, MA, pages 636–645. http://www.aclweb.org/anthology/D10-1062.

Stephen Clark and James R. Curran. 2007. 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.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1–8. https://doi.org/10.3115/1118693.1118694.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *IN PROC. INT' L CONF. ON LANGUAGE RESOURCES AND EVALUATION (LREC*. pages 449–454.

Pétr Dienes and Amit Dubey. 2003. Deep syntactic processing by combining shallow methods. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Sapporo, Japan, pages 431–438. https://doi.org/10.3115/1075096.1075151.

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.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, pages 136–143. https://doi.org/10.3115/1073083.1073107.

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.

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 the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Uppsala, Sweden, pages 1–11. http://www.aclweb.org/anthology/P10-1001.

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 the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Cambridge, MA, pages 1288–1298. http://www.aclweb.org/anthology/D10-1125.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Baltimore, Maryland, pages 1381–1391. http://www.aclweb.org/anthology/P14-1130.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large

annotated corpus of english: the penn treebank. *Computational Linguistics* 19(2):313–330. http://dl.acm.org/citation.cfm?id=972470.972475.

Ryan McDonald. 2006. *Discriminative learning and spanning tree algorithms for dependency parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)*. volume 6, pages 81–88.

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

Yusuke Miyao and Jun'ichi Tsujii. 2008. Feature forest models for probabilistic hpsg parsing. *Computational Linguistics* 34(1):35–80. https://doi.org/10.1162/coli.2008.34.1.35.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34:513–553. https://doi.org/http://dx.doi.org/10.1162/coli.07-056-R1-07-027.

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

Wolfgang Seeker, Richárd Farkas, Bernd Bohnet, Helmut Schmid, and Jonas Kuhn. 2012. Data-driven dependency parsing with empty heads. In *Proceedings of COLING 2012: Posters*. The COLING 2012 Organizing Committee, Mumbai, India, pages 1081–1090. http://www.aclweb.org/anthology/C12-2105.

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

Xu Sun. 2014. Structure regularization for structured prediction. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, Curran Associates, Inc., pages 2402–2410. http://papers.nips.cc/paper/5563-structure-regularization-for-structured-prediction.pdf.

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*. Association for Computational Linguistics, Suntec, Singapore, pages 342–350. http://www.aclweb.org/anthology/P/P09/P09-1039.

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.

Nianwen Xue and Yaqin Yang. 2013. Dependency-based empty category detection via phrase structure trees. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Atlanta, Georgia, pages 1051–1060. http://www.aclweb.org/anthology/N13-1125.

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

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.