Reorder and then Parse, Fast and Accurate Discontinuous Constituency Parsing

Kailai Sun[1,2], **Zuchao Li**[3,*], **and Hai Zhao**[1,2,*]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University
[2]MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
[3]School of Computer Science, Wuhan University
kaishu2.0@sjtu.edu.cn, zcli.charlie@gmail.com, zhaohai@cs.sjtu.edu.cn

Abstract

Discontinuous constituency parsing is still kept developing for its efficiency and accuracy are far behind its continuous counterparts. Motivated by the observation that a discontinuous constituent tree can be simply transformed into a pseudo-continuous one by artificially reordering words in the sentence, we propose a novel reordering method, thereby construct fast and accurate discontinuous constituency parsing systems working in continuous way. Specifically, we model the relative position changes of words as a list of actions. By parsing and performing this actions, the corresponding pseudo-continuous sequence is derived. Discontinuous parse tree can be further inferred via integrating a high-performance pseudo-continuous constituency parser. Our systems are evaluated on three classical discontinuous constituency treebanks, achieving new state-of-the-art on two treebanks and showing a distinct advantage in speed.

1 Introduction

Constituency parses represent the syntactic structure by hierarchical constituent trees, which decompose the sentence into constituents and establish hierarchical relations between constituents and words (Corro, 2020). In continuous constituent trees, the sequence of words that compose the constituent must be contiguous. This type of structure is enough for representing most of the syntactic structure (Fernández-González et al., 2021a) and has been extensively studied.

Nevertheless, some syntactic phenomena, such as long-distance dependencies and extractions, require discontinuous words to form a constituent, which have been argued to be unavoidable (McCawley, 1982; Bunt et al., 1987; Müller, 2004). Thus, discontinuous constituent trees with crossing branches are introduced to describe all the syntactic phenomena present in human language.

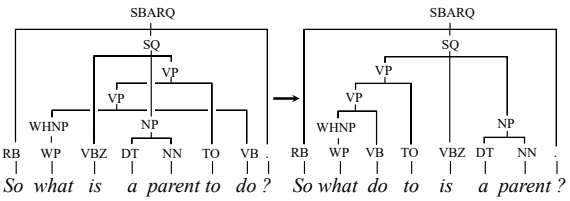

Figure 1: An example of converting a discontinuous tree into a pseudo-continuous one.

Compared to continuous constituency parsing, discontinuous parsing is a more complicated problem, so that the high computational cost is particularly prominent. Many researches (Ruprecht and Mörbitz, 2021; Coavoux and Crabbé, 2017; Coavoux et al., 2019) have focused on how to reduce the time complexity of discontinuous parsing algorithm to reduce or get rid of the constraint of sentence length.

Based on the observation that a discontinuous tree can be converted into a pseudo-continuous one by simply reordering the words in the sentence, as an example in Figure 1, and the advantage that parsing a continuous tree is much simpler than parsing a discontinuous one. Fernández-González et al. (2021b) develops a pointer network to generate reordered word sequence by predicting the position of each word in the pseudo-continuous sequence, and then applies off-the-shelf continuous constituency parser to get the constituent tree. This method is intuitive to achieve improvement in speed, and its F_1 score is also greatly improved.

However, the accuracy of existing reordering methods still lags behind. We find that the performances of the pseudo-continuous constituency parser outperforms all the existing discontinuous works by a wide margin if the input pseudo-continuous sequences are completely correct in word positions. In order to preserve the high performance of pseudo-continuous parser as completely

*Corresponding author. This work was supported by Key Projects of National Natural Science Foundation of China (U1836222 and 61733011).

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 10575–10588
December 7-11, 2022

as possible, it is an urgent problem to study how to improve the accuracy of reordering and the influence of reordering accuracy on the overall discontinuous constituency parsing result.

In this work, we construct fast and accurate discontinuous constituency parsing systems working in continuous way by designing a novel method of reordering the words in sentences. Specifically, we model the relative position changes of words when converting the original sentence to the pseudo-continuous sequence as a list of actions that moving words forward and backward in the sentence. We employ an action graph which indicates these actions by directed edges with action labels. Since the action graph and the semantic dependency graph have similar structure, we adopt the semantic dependency parsing model (Wang and Tu, 2020) as our basic model for parsing the action graph. We further establish a robust rearrangement algorithm with time complexity of $O(n^2)$, which derives the pseudo-continuous sequence by performing the actions in the predicted graph.

With our action schema and rearrangement algorithm, we achieve a considerable improvement in reordering accuracy than current state-of-the-art method and remain comparable efficiency. Meantime, we innovatively revise the architecture of the parser to better adapt to the structural characteristics of action graph and achieve better results. After deriving the pseudo-continuous sequence, any well-performed continuous constituency parsing model can be used as our pseudo-continuous parser to parse this sequence into a pseudo-continuous tree. Finally, we can restore the corresponding discontinuous tree by reducing the pseudo-continuous sequence to the original sentence.

We evaluate our systems on three classical discontinuous constituency treebanks: Discontinuous English Penn Treebank (DPTB) (Evang and Kallmeyer, 2011), Tiger (Brants et al., 2002) and Negra (Skut et al., 1997). Two continuous constituency parsing models: a span-based model (Kitaev et al., 2019) and a transition-based model (Yang and Deng, 2020), are used as the pseudo-continuous parser for experiments. Our method is faster than most of the existing discontinuous parsers. We achieve new state-of-the-art results on Tiger and Negra, and comparable to state-of-the-art results on DPTB.

2 Related Work

Syntax shows its effect in language understanding (He et al., 2018; Li et al., 2021a,b; Sun et al., 2021; Li et al., 2022; Zhang et al., 2022). Discontinuous constituency parsing, as a special constituency parsing form has drawn much attention.

Grammar-based Method Inheriting and expanding the grammar-based methods of continuous constituency parsing, mildly context-sensitive grammar such as linear context-free rewriting systems (LCFRS) (Vijay-Shanker et al., 1987) and multiple context-free grammars (MCFGS) (Seki et al., 1991) has been applied for discontinuous parsing, which has higher parsing complexity than its continuous counterpart. For example, the complexity of accurate CKY-style LCFRS is $O(n^{3 \cdot \text{fan-out}})$, where grammar-specific fan-out is greater than 1 (Kallmeyer, 2010). To reduce the limit on the sentence length, Ruprecht and Mörbitz (2021) proposes a supertagging-based parser for LCFRS. However, there is still a considerable gap in speed and accuracy between discontinuous parsing and continuous parsing.

Span-based Method In continuous constituency parsing, span-based methods generally excel in both speed and accuracy (Stern et al., 2017; Kitaev and Klein, 2018; Kitaev et al., 2019). Span-based methods are also proposed for discontinuous constituency parsing (Corro, 2020; Stanojević and Steedman, 2020), but their discontinuous accuracy is not good enough. Through error analysis, Coavoux (2021) concludes that compared to transition-based parser, span-based parser has less fine-grained features which are particularly helpful for predicting discontinuous constituents. Thus, first converting the discontinuous tree into a pseudo-continuous one and then utilizing span-based parser may be a better choice.

Transition-based Method In continuous constituency parsing, transition-based methods based on shift-reduce strategy (Zhu et al., 2013; Sagae and Lavie, 2005) are widely used. Some works propose to add specific transition actions to change the order of words (Maier, 2015; Stanojević and Alhama, 2017) for discontinuous parsing. However, this type of extension may make transition sequences so long that affecting efficiency and accuracy. To alleviate this problem, methods of dealing directly with discontinuous sequences (Coavoux and Crabbé, 2017) and designing new data structures and corresponding actions (Coavoux et al.,

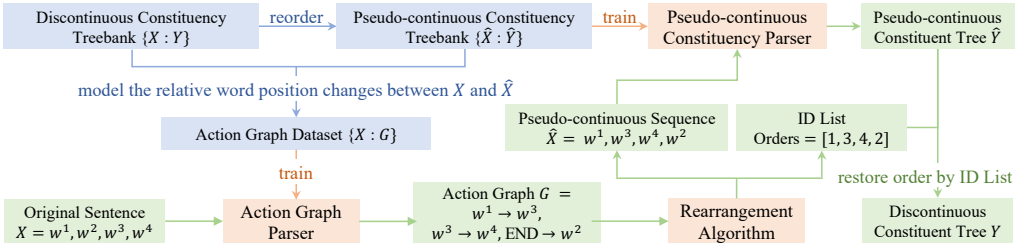

Figure 2: The flow of our system. w^i $(i = 1, 2, 3, 4)$ is the word with ID $= i$.

2019; Coavoux, 2021) are used to reduce the length of the transition sequence. These methods are indeed effective, but there is still room for improvement in terms of speed and accuracy.

Conversion-based Method Several works also aim to convert discontinuous constituency parsing to a simpler task. For example, some works reduce discontinuous constituency parsing to non-projective dependency parsing by encoding the constituents to arc labels (Fernández-González and Martins, 2015; Fernández-González and Gómez-Rodríguez, 2020) , while Vilares and Gómez-Rodríguez (2020) converts the task to sequence tagging by representing the discontinuous tree as a sequence of tag. In this work, we convert the discontinuous parsing task into continuous parsing task by modeling, parsing and presenting the words relocating actions, and thus reduce the complexity of the problem.

3 Method

The flow of our system[1] is shown in Figure 2. In data pre-processing stage, we first build a pseudo-continuous constituency treebank based on discontinuous treebank by reordering the words in the sentences. The reordering strategy is consistent with Fernández-González et al. (2021b) and is described in detail in Appendix A. Then, we construct an action graph dataset based on our action schema by modeling the relative position changes of words as action edges. In training stage, we train an action graph parser on the action graph dataset and a pseudo-continuous constituency parser on the pseudo-continuous treebank. In prediction stage, we first parse the original sentence: X into action graph: G. Then, we propose an algorithm which derives the reordered sequence: $\hat{X}$ and the ID list: `Orders` from X by performing actions in G. $\hat{X}$ is then used as input to the pseudo-continuous constituency parser for pseudo-continuous tree $\hat{Y}$ parsing. Finally, `Orders` which indicates the original positions of the words in X, is used for discontinuous tree Y restoration.

3.1 Action Graph

Denote the original sentence as $X = w_1, w_2, ..., w_N$, and the reordered sequence as $\hat{X}$. For each word w_i $(i = 1, 2, ..., N)$, its index in X is i and in $\hat{X}$ is r_i. That means w_i and w_{r_i} represent the same word in different positions in X and $\hat{X}$ respectively. In addition, we use the index of each word in X as its ID, which remains the same in X and $\hat{X}$.

According to our observation, reordering words in a sentence is identical to moving some of the words forward or backward. Therefore, we design an action schema that indicates the specific relative positions to which some words need to be moved forward or backward when converting X to $\hat{X}$. The criteria for judging whether a word needs to be moved are as follows:

- ***Move Forward***: For w_i (w_{r_i}), if there exist a word which is to the left of w_i in X and to the right of w_{r_i} in $\hat{X}$, w_i is considered to be moved forward.

$$\exists\, w_j : 1 \leq j < i \;\&\; N \geq r_j > r_i$$

- ***Move Backward***: For w_i (w_{r_i}), if there exist a word which is to the right of w_i in X and to the left of w_{r_i} in $\hat{X}$, w_i is considered to be moved backward.

$$\exists\, w_j : N \geq j > i \;\&\; 1 \leq r_j < r_i$$

On this basis, we indicate movements by annotating edges pointing from head to child and thus forming an action graph. The criteria for assigning edges and labels are as follows:

- For w_i (w_{r_i}), if it needs to be moved forward and $w_{r_i-1} \neq w_{i-1}$. We use w_{r_i-1} as the positioning for w_i and assign an edge with label **`MOVE-L`**: $(w_{r_i-1} \rightarrow w_i)$, which indicates the action that moving w_i after w_{r_i-1}.
- For a continuous segment $w_i, w_{i+1}, ..., w_{i+s}$ $(s \geq 1)$ in X, if it is also a continuous segment in

[1]Our code is published in https://github.com/KAI-SHU/Reorder-and-then-Parse.

$\hat{X}$ and it needs to be moved forward. We assign an edge with label **SPAN**: $(w_i \rightarrow w_{i+s})$, which indicates that words from w_i to w_{i+s} should be moved forward as a whole.

- For w_i (w_{r_i}), if it needs to be moved backward and $w_{r_i+1} \neq w_{i+1}$. We use w_{r_i+1} as the positioning for w_i and assign an edge with label **MOVE-R**: $(w_{r_i+1} \rightarrow w_i)$, which indicates the action that moving w_i before w_{r_i+1}.
- For a continuous segment $w_{i-s}, ..., w_{i-1}, w_i$ $(s \geq 1)$ in X, if it is also a continuous segment in $\hat{X}$ and it needs to be moved backward. We assign an edge with label **SPAN**: $(w_i \rightarrow w_{i-s})$, which indicates that words from w_{i-s} to w_i should be moved backward as a whole.

We introduce an example from DPTB in the first block of Figure 3 to show the gold bi-directional action graph where the green edges indicate forward movements, and the purple edges indicate backward movements. An "END" virtual node is appended because several words may be moved backward to the end of the sequence in some instances, which requiring a node to be the head of such edges. It is worth noting that only using the edges of a single direction is sufficient to derive an accurate pseudo-continuous sequence. However, there is no guarantee that the action graph parser can make decisions exactly, so it is necessary to combine the forward and backward actions together and let them complement each other to achieve higher reordering accuracy.

3.2 Action Graph Parsing Model

Since there are action edges of two directions, we introduce two parsing configurations. The first one is to merge the forward and backward edges into one graph and employ a single parser for bi-directional action prediction directly. The other one is to train two separate parsers to predict the forward and backward edges respectively.

Bi-directional Parsing Model In semantic dependency parsing, a dependency is represented by an annotated edge pointing from head to dependent. A word can be regarded as a dependent or head multiple times in different dependencies, or it may not belong to any dependency. Therefore, our bi-directional action graph has a similar structure to semantic dependency graph. In this case, we employ a second-order graph-based semantic dependency model (Wang et al., 2019) for parsing our bi-directional action graph. We make a simple clarification for this model as follows.

First, the model uses four single-layer feedforward layers (FNNs) to differentiate the output of the encoder: $H = [h_1, h_2, ..., h_N]$ into four hidden states with different functions: $H^{(o\text{-}m)}$, where $o \in [\text{edge}, \text{label}]$ and $m \in [\text{head}, \text{dep}]$. Then, the first-order scores of edge: $S^{(edge)} \in \mathcal{R}^{N \times N}$ and the first-order scores of label: $S^{(label)} \in \mathcal{R}^{N \times N \times C}$ are calculated by biaffine attention, where C is the number of labels:

$$\text{Biaff}(v_1, v_2) := v_1^T U v_2 + b,$$

$$s_{ij}^{(o)} = \text{Biaff}^{(o)}(h_i^{(o\text{-dep})}, h_j^{(o\text{-head})}),\ o \in [\text{edge}, \text{label}].$$

Meantime, H is also differentiated into other three hidden states for second-order scoring: $H^{(\text{pair-}q)}$, where $q \in [\text{grand}, \text{head}, \text{dep}]$. Three kinds of second-order scores of edge, siblings: $S^{(\text{sib})}$, co-parents: $S^{(\text{cop})}$ and grandparents: $S^{(\text{gp})}$ are calculated by trilinear functions:

$$\text{Trilin}(v_1, v_2, v_3) := v_3^T v_1^T U v_2 + b.$$

For label prediction, softmax function is performed on $S^{(\text{label})}$ to calculate the probability of each label given the edge. For edge prediction, a conditional random field (CRF) inference is defined to determine the existence of edges. The unary potential is defined by $S^{(\text{edge})}$ and the binary potential is defined by $S^{(\text{sib})}$, $S^{(\text{cop})}$ and $S^{(\text{gp})}$. For efficiency, mean field variational inference (MFVI) is used for approximate inference on this CRF.

During training, cross entropy losses are measured for both edge and label, and the optimization objective is the weighted average of the two losses.

$$\mathcal{L} = \lambda \mathcal{L}^{(\text{label})} + (1 - \lambda)\mathcal{L}^{(\text{edge})}.$$

Uni-directional Parsing Model Semantic dependency parsing model suited well for parsing bi-directional action graph. However, for the uni-directional action graph parsing, because of its relatively simple chain structure (compared to full graph), it is not suitable to use such model with graph constraints. Instead, we adopt and revise the syntactic dependency parsing model with tree constraints for better adaptation.

In uni-directional action parsing, due to the tree constraint characteristic that there is at most one head per word, we revise the basic biaffine model with a single-layer MLP as a binary classifier that receives the output of the encoder and predicts whether each word has a head or not.

$$S^{(\text{head})} = \text{MLP}(H) \in \mathcal{R}^{N \times 2}.$$

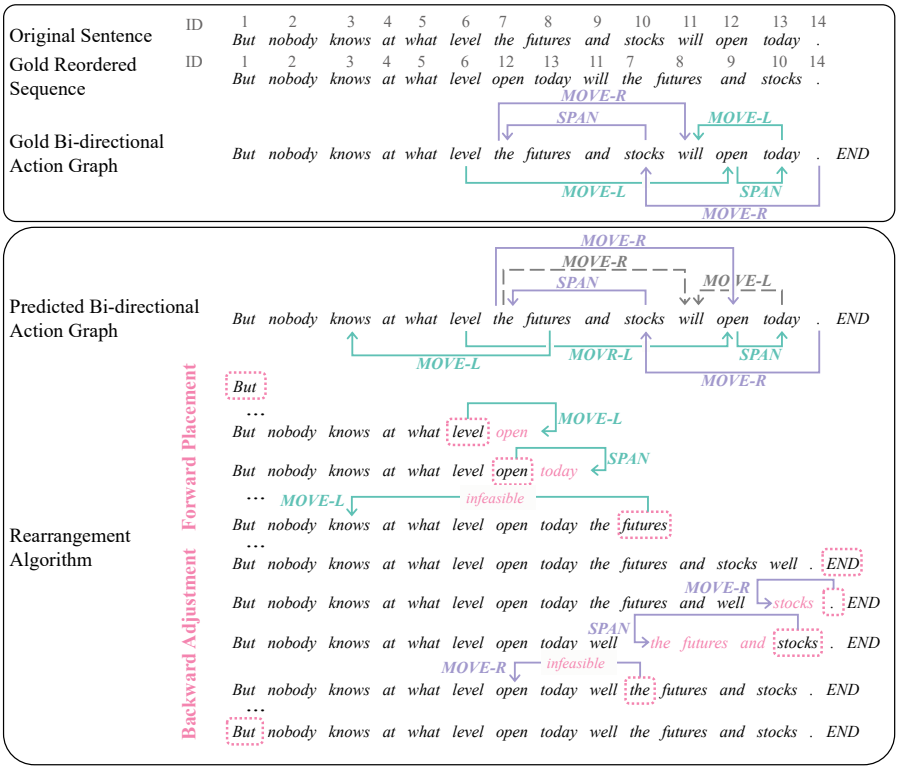

Figure 3: An example of the gold bi-directional action graph and the process of rearrangement algorithm.

The process of calculating $S^{(\text{edge})}$ and $S^{(\text{label})}$ is consistent with the bi-directional action graph parser using biaffine attention. Then, we perform softmax on $S^{(\text{head})}$, $S^{(\text{edge})}$ and $S^{(\text{label})}$ to calculate the probability of the presence and absence of the head of each word, the probability of each edge given the child, and the probability of each label given the edge. While in the inference, we first decide the nodes with heads according to $S^{(\text{head})}$, and performs greedy search on $S^{(\text{edge})}$ and $S^{(\text{label})}$ for action prediction.

Denote $G^{\star}$ as the gold action graph for sentence X, we define the following cross entropy losses:

$$\mathcal{L}^{(\text{head})} = -\sum log(P_\theta(g_i^{\star(\text{head})}|X),$$

$$\mathcal{L}^{(\text{edge})} = -\sum_{i}^{i} \mathbb{1}(g_i^{\star(\text{head})}) log(P_\theta(g_i^{\star(\text{edge})}|X),$$

$$\mathcal{L}^{(\text{label})} = -\sum_{i,j}^{i} \mathbb{1}(g_i^{\star(\text{head})}) \mathbb{1}(g_i^{\star(\text{edge})} = w_j) log(P_\theta(g_{ij}^{\star(\text{label})}|X),$$

where θ is the parameters of our model and $\mathbb{1}(\cdot)$ denotes the indicator function. $\mathbb{1}(g_i^{\star(\text{head})})$ equals 1 when w_i has head in $G^{\star}$ and 0 otherwise. The final optimization objective is the weighted average of the three losses.

$$\mathcal{L} = \lambda \mathcal{L}^{(\text{label})} + (1-\lambda)(\mathcal{L}^{(\text{edge})} + \mathcal{L}^{(\text{head})}).$$

With the removal of complicate second-order scoring of edges, the uni-directional action graph parser has fewer parameters and less time complexity compared with the bi-directional version, so it is more efficient at parsing a single graph.

3.3 Rearrangement Algorithm

After predicting the action graph, we design an algorithm performing the actions in the graph to derive the reordered sequence. The algorithm is shown in Algorithm 1 which consists of two phases: forward placement and backward adjustment. For convenience, we perform actions directly on the IDs of the words and map the IDs to the words at the end of the algorithm. Since the implementation of backward adjustment are complicated, we define several functions to simplify the writing of the pseudo code:

- **`MoveRAble`**(`curID`,`chID`): If `chID` is placed by `Nature` and it is to the left of `curID`, return true.
- **`SpanRAble`**(`curID`,`span`): If the entire `span` is placed by `Nature` and it is a contiguous segment in `Orders` and it is to the left of `curID`, return true.
- **`Adjust`**(`curID`,`segment`): Move the `segment`, which can be a span or a single ID, before `curID`.

In the forward placement, we place the words from left to right according to the forward edges. Since the first word never moves, we first add ID $=$ 1 to the ID list: `Orders`. In each loop, we take the last ID in `Orders` as the currently observed object:

Algorithm 1 Rearrangement (Perform Action Graph)

Input: X, G

```
/* Forward Placement */
Orders ← [1] ;
while len(Orders) < len(X) do
    curID ← Orders[-1] ;
    /* Place by MOVE-L */
    for (curID, chID) in G[MOVE-L] do
        if chID ∉ Orders then
            Orders ← Orders + [chID] ;
            back to while
    /* Place by SPAN */
    if curID is placed by MOVE-L then
        for (curID, chID) in G[SPAN] do
            if curID < chID then
                span ← [curID+1 to chID] ;
                if ∀ ID ∈ span: ID ∉ Orders then
                    Orders ← Orders + span ;
                    back to while
    /* Place by Nature */
    Orders ← Orders + [min(ID |ID ∉ Orders)] ;
/* Backward Adjustment */
curID ← Orders[-1] ;
while curID ≠ 1 do
    if curID is placed by Nature then
        /* Adjust by MOVE-R */
        for (curID, chID) in G[MOVE-R] do
            if MoveRAble(curID, chID) then
                Adjust(curID, chID);
                curID ← chID ;
                back to while
        /* Adjust by SPAN */
        if curID is adjusted by MOVE-R then
            for (curID, chID) in G[SPAN] do
                if curID > chID then
                    span ← [chID to curID−1] ;
                    if SpanRAble(curID, span) then
                        Adjust(curID, span) ;
                        curID ← chID ;
                        back to while
    curID ← curID << 1 ;
X̂ ← X[Orders] ;
```

Output: $\hat{X}$, Orders

curID, and iterate through all edges with it as the head until we find a feasible action. The MOVE-L action works if its child has not been placed in Orders. The SPAN action works if the entire span has not been placed in Orders and curID is placed in Orders by MOVE-L. If no feasible action, the one which is the smallest ID that has not been placed in Orders, is placed by Nature. The first phase ends when all IDs have been placed in Orders.

In the backward adjustment, we scan Orders from right to left, looking for possible missing actions based on the backward edges. We assume that the forward actions we have used are correct. Therefore, we only focus on the IDs that are placed by Nature and ignore those that have already been moved forward. In each loop, we iterate through all edges with curID as the head until we find a feasible action. The MOVE-R action works if MoveRAble(curID, chID) is true. The SPAN action works if SpanRAble(curID, span) is true and curID is adjusted by MOVE-R. If no feasible action, move curID one to the left. The second phase ends when we move to the first ID.

To make our algorithm robust to errors in the action graph parse, we filter out some unreasonable actions through the feasibility judgment, and use the backward edges to supplement the result derived by the forward edges. This is important for high-accuracy reordering. As an example shown in the second block of Figure 3, the predicted action graph is not completely correct: two edges are missing and two edges are redundant compared to the gold action graph in the first block. However, we can still derive the correct reordered sequence with our algorithm.

3.4 Time Complexity

The bi-directional action graph parser has a time complexity of $O(d_b^2 + d_t^2 + n^3)$ and the uni-directional action graph parser has a time complexity of $O(d_b^2 + n^2)$, where d_b and d_t are the hidden sizes of the biaffine and trilinear and n is the sentence length. The rearrangement algorithm has a time complexity of $O(n^2)$. The continuous constituency parsing models (Kitaev et al., 2019; Yang and Deng, 2020) used as our pseudo-continuous constituency parser in our experiments both have a time complexity of $O(d_b^2 + n^3)$. Therefore, the whole time complexity of our system is $O(d_b^2 + d_t^2 + n^3)$ which is comparable to that of the most efficient methods currently, including Fernández-González et al. (2021b) and Corro (2020) both with a time complexity of $O(d_b^2 + n^3)$.

4 Experiments

Data Setup We conduct our experiments on English treebank DPTB with standard split and German treebanks: Tiger and Negra with common split (Seddah et al., 2013; Dubey and Keller, 2003). We transform these three treebanks into corresponding action graph datasets and pseudo-continuous constituency treebanks for training and evaluating

Parser(no tags or predict PoS tags)		NEGRA			TIGER			DPTB		
		F_1	DF_1	sent/s	F_1	DF_1	sent/s	F_1	DF_1	sent/s
Fully supervised or Semi-supervised (Pre-trained embeddings)										
Vilares and Gómez-Rodríguez (2020)		77.1	36.5	**715**	79.2	40.1	**568**	89.1	41.2	**611**
Corro (2020)		86.3	56.1	478	85.2	51.2	474	92.9	64.9	355
Ruprecht and Mörbitz (2021)		86.54	61.89	104	85.12	61.00	80	91.77	76.14	86
Semi-supervised (Pre-trained language models)										
Vilares and Gómez-Rodríguez (2020) + $BERT_{base}$		84.2	46.9	81	84.7	51.6	80	91.7	49.1	80
Vilares and Gómez-Rodríguez (2020) + $BERT_{large}$		-	-	-	-	-	-	92.8	53.9	34
Corro (2020) + $BERT_{base}$		91.6	66.1	-	90.0	62.1	-	94.8	68.9	-
Ruprecht and Mörbitz (2021) + $BERT_{base}$		90.94	72.58	68	88.34	69.02	60	93.32	80.53	57
Coavoux (2021) + $BERT_{base}$		91.7	73.3	-	90.2	72.9	-	95.0	82.5	-
Fernández-González et al. (2021a)+ $GottBERT_{base}^{\dagger}$		89.08	67.06	-	88.53	67.76	-	-	-	-
Fernández-González et al. (2021a) + $RoBERTa_{large}^{\dagger}$		-	-	-	-	-	-	95.47	**83.80**	-
Fernández-González et al. (2022) + $BERT_{base}^{\dagger\diamond}$		91.0	76.6	-	89.8	71.0	-	-	-	-
Fernández-González et al. (2021b) + $BERT_{base}^{\ddagger}$		90.0	65.9	275	88.5	63.0	238	94.0	68.9	231
Fernández-González et al. (2021b) + $BERT_{large}^{\ddagger}$		92.0	67.9	216	90.5	68.1	207	94.7	72.9	193
Fernández-González et al. (2021b) + $XLNet_{large}^{\ddagger}$		-	-	-	-	-	-	95.1	74.1	179
Bi + Kitaev et al. (2019) + $BERT_{base}^{\ddagger}$	F & B	91.03	70.40	306	89.30	68.37	267	94.60	74.82	243
Bi + Kitaev et al. (2019) + $RoBERTa_{large}$	F & B	92.88	75.93	177	91.18	72.78	166	95.70	77.99	152
Bi + Kitaev et al. (2019) + $BERT_{large}$	F & B	**93.61**	**76.97**	185	**91.89**	**73.93**	177	94.97	75.78	161
Bi + Kitaev et al. (2019) + $XLNet_{large}$	F & B	-	-	-	-	-	-	**95.73**	78.46	137
Bi + Yang and Deng (2020) + $BERT_{large}$	F & B	92.93	76.02	149	91.52	73.44	142	94.59	75.31	136
Bi + Yang and Deng (2020) + $XLNet_{large}$	F & B	-	-	-	-	-	-	95.68	77.81	124
Uni + Kitaev et al. (2019) + $BERT_{large}$ (+ $XLNet_{large}$)	F	93.38	75.12	193	91.79	73.02	186	**95.89**	82.52	153
	B	93.67	77.18	193	91.86	74.24	185	95.76	79.83	153
	F & B	**93.69**	**77.83**	131	**91.92**	**74.65**	127	95.82	81.71	101
Uni + Yang and Deng (2020) + $BERT_{large}$ (+ $XLNet_{large}$)	F	92.75	74.93	158	91.34	72.83	151	95.87	81.89	139
	B	92.94	76.67	157	91.47	73.52	151	95.77	80.35	139
	F & B	93.01	77.24	102	91.54	73.89	98	95.81	81.58	82

Table 1: Comparison of our systems against discontinuous constituent methods on the test splits. †: not fine-tuning the pre-trained model. ⋄: using extra dependency information. ‡: the network for reordering does not fine-tune the pre-trained model. The results of Fernández-González et al. (2021b) we refer use Kitaev et al. (2019) as the continuous constituency parsing model. Bi: bi-directional action graph parser. Uni: uni-directional action graph parser. For DPTB, the Uni systems use $XLNet_{large}$. *F*: forward placement, *B*: backward placement, *F & B*: forward placement and backward adjustment.

action graph parsers and pseudo-continuous constituency parsers. The data statistics of action graph datasets are shown in Appendix B.

Evaluation Metric For action graph parsing, we calculate the labeled edge F_1 score (LF_1) as the metric to select the best model on the development set. For pseudo-continuous constituency parsing, we use the continuous constituency F_1 score as the metric to select the best model on the development set. For overall results, we use official `discodop` to calculate continuous F_1 score and discontinuous F_1 score (DF_1). Parsing speed is measured by the average number of sentences our systems process per second (sent/s).

Implementation The hyper-parameters of our action graph parsing models are shown in Appendix C and the hyper-parameters of the pseudo-continuous constituency parser are consistent with Kitaev et al. (2019) and Yang and Deng (2020). For DPTB, word embedding is initialized with Glove (Pennington et al., 2014) while it is randomly initialize for Tiger and Negra. We introduce $BERT_{base}$, $BERT_{large}$ (Devlin et al., 2019), $RoBERTa_{large}$ (Liu et al., 2019) and $XLNet_{large}$ (Yang et al., 2019) as the pre-trained models. We use $BERT_{base}$ for fixed pre-trained embedding and fine-tune other pre-trained models as the encoder. Our models are trained and tested on Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz and a single NVIDIA RTX TITAN GPU. All the results we report are averages of the results of five seeds.

Discontinuous Constituency Parsing Results We report F_1 scores, DF_1 scores and speeds of our systems on test splits in Table 1 compared with current discontinuous constituency parsers. When using bi-directional action graph parser, we achieved state-of-the-art F_1 for all the three treebanks and state-of-the-art DF_1 for German treebanks. When using uni-directional action graph parser, our results are even better. Merging the forward and backward action edges parsed by two uni-directional action parsers together achieves the best results

	Reordered Sequences		LSD	LCS	R-Rec	R-Prec	RF_1	EM	%pred	%gold
NEGRA	Bi + $BERT_{base}$ †	*F & B*	3.18	93.81	74.41	83.31	78.61	86.60	22.41	22.52
	Bi + $RoBERTa_{large}$	*F & B*	2.96	94.91	80.27	84.84	82.49	88.70	23.58	22.52
	Bi + $BERT_{large}$	*F & B*	2.70	95.60	80.38	88.21	84.11	90.40	22.82	22.52
		F	2.68	95.55	76.19	**90.60**	82.77	90.00	21.19	22.52
	Uni + $BERT_{large}$	*B*	2.58	**95.72**	77.99	90.30	83.69	**90.50**	21.90	22.52
		F & B	**2.55**	95.64	**83.37**	87.55	**85.41**	90.40	23.87	22.52
TIGER	Bi + $BERT_{base}$ †	*F & B*	3.16	94.56	70.00	86.68	77.45	87.88	20.16	21.19
	Bi + $RoBERTa_{large}$	*F & B*	2.65	95.46	76.37	87.73	81.66	89.74	21.09	21.19
	Bi + $BERT_{large}$	*F & B*	**2.21**	95.68	77.44	90.55	**83.49**	90.26	20.63	21.19
		F	**2.21**	**95.69**	75.59	**91.93**	82.96	**90.34**	19.77	21.19
	Uni + $BERT_{large}$	*B*	2.66	95.56	73.82	90.41	81.28	90.04	19.94	21.19
		F & B	2.27	95.52	**77.96**	89.57	83.36	89.98	21.23	21.19
DPTB	Bi + $BERT_{base}$ †	*F & B*	1.11	97.53	77.06	86.64	81.57	94.21	8.47	8.63
	Bi + $BERT_{large}$	*F & B*	1.07	97.56	80.04	86.40	83.10	94.37	8.61	8.63
	Bi + $RoBERTa_{large}$	*F & B*	0.88	98.04	81.11	**91.55**	86.01	95.41	8.17	8.63
	Bi + $XLNet_{large}$	*F & B*	**0.83**	97.97	82.95	90.32	86.48	95.36	8.46	8.63
		F	0.86	**98.12**	83.30	90.59	**86.79**	**95.57**	8.34	8.63
	Uni + $XLNet_{large}$	*B*	1.09	97.68	75.47	90.09	82.13	94.66	8.15	8.63
		F & B	0.88	98.08	**84.26**	89.03	86.58	95.53	8.64	8.63

Table 2: The accuracy of the reordered pseudo-continuous sequences in the test splits. LSD (location square deviation), LCS (% of the longest common string). R-Rec (relocated recall), R-Prec (relocated precision) and RF_1 (relocated F_1) are calculated by taking the relocated words in the gold set as positive examples and the other words as negative examples. EM (% of reordered sequences that are completely correct). %gold (% of gold relocated words), %pred (% of predicted relocated words).

on two German treebanks, and the forward unidirectional action parser achieves the best results on DPTB. Our results using span-based continuous constituency parsing model (Kitaev et al., 2019) are generally better than those of using transition-based one (Yang and Deng, 2020). Therefore, we infer that compared with the left-to-right parsing of transition-based method, the bottom-up parsing of span-based method is more suitable for sequences with rearranged word order. It is worth noting that in the case of using the same pseudo-continuous constituency parser, our DF_1 scores are greatly improved compared with Fernández-González et al. (2021b), which is due to our higher reordering accuracy. This shows that our reordering method is effective.

The speeds of our systems are slightly slower than those of Fernández-González et al. (2021b) in which the reordering is achieved by a pointer network, and faster than other methods with pre-trained models. Since the time complexity of our rearrangement algorithm is $O(n^2)$ and it has no network part, its execution time is almost negligible compared to the total time. Therefore, the reason why our systems are slower than Fernández-González et al. (2021b) is that our action graph parser is more complex than the pointer network. However, the more complex reordering component gives better performance and this tradeoff of sacrificing speed for accuracy is valuable, because it is the reordering performance that limits the accuracy of discontinuous parsing in the 'reorder and then parse' method.

Obviously, large pre-trained models usually imply higher accuracy but lower speed. Thus, speed limits some relatively complex methods to further use large pre-trained models for performance improvement. However, the high efficiency of our systems allows us to maintain their practicality when including large pre-trained models. In other word, we can have cake and eat it too.

Reordering Accuracy Because the LF_1 score can not directly reflect the accuracy of the reordered sequences derived by the rearrangement algorithm, we only show the results of action graph parsing on the dev splits in Appendix D. In order to better demonstrate the performance of our whole reordering method, we include a series of metrics for evaluating the reordered sequences in Table 2. All of these metrics illustrate the reordering accuracy, but they may also lose their comprehensiveness in some cases. For example, LCS only counts the longest common substring and ignore the rest part of the sequence. LSD pays too much attention to the absolute location and ignores the relative position between words that might form a single constituent.

Referring to the final discontinuous constituency results, we find that R-Rec and RF_1 have a more stable measurement for reordering accuracy than other metrics. In addition, the R-Prec of a single

uni-directional action graph parser is high, but the R-Rec is relatively low. When merging the forward and backward action graphs parsed by two uni-directional action parsers together, although the R-Prec is slightly reduced, the R-Rec is significantly higher, thus improving the reordering accuracy.

	NEGRA		TIGER		DPTB	
	F_1	DF_1	F_1	DF_1	F_1	DF_1
S + $BERT_{base}$	92.94	88.13	91.35	87.05	94.97	84.92
S + $RoBERTa_{large}$	94.65	90.21	93.03	89.96	96.16	91.49
S + $BERT_{large}$	95.23	91.43	93.61	91.48	95.52	86.47
S + $XLNet_{large}$	-	-	-	-	96.20	91.61
T + $BERT_{large}$	94.87	90.66	93.24	90.82	95.51	85.76
T + $XLNet_{large}$	-	-	-	-	96.20	89.81

Table 3: Results of pseudo-continuous parsers using gold pseudo-continuous sequences on the test splits. S: Kitaev et al. (2019), T: Yang and Deng (2020).

Pseudo-continuous Parsing Performance We also include the performance of the pseudo-continuous constituency parsers when using gold reordered pseudo-continuous sequences in Table 3. We restore the predicted pseudo-continuous trees to discontinuous trees and score using discontinuous constituency metrics. The F_1 scores of the pseudo-continuous constituency parsers are much higher than current state-of-the-art discontinuous parsers, and the DF_1 scores show a huge increase of over 10% in all three treebanks. This shows that the reordering is the bottleneck, the overall discontinuous performance still has much room of improving if better reordering is achieved. In addition, span-based models perform better than transition-based models, especially DF1. This further verifies that the span-based continuous constituency parsing method is more suitable than the transition-based method for parsing the rearranged word sequences.

Ablation Study In Table 4, we report results of our bi-directional action graph parser under different configurations on test splits. Training the bi-directional action parser with both the forward and the backward edges is better than training with only one direction. This verifies that information from both directions can reinforce each other and help the learning process of bi-directional parser. By comparing *F* with *B* and *F (& B)* with *(F &) B*, we find that the models learn backward actions better than forward actions. Besides, the results of *B & F* and *F & B* are better than those of *F (& B)* and *(F &) B*, which suggests that our rearrangement algorithm using one direction for placement and the other for adjustment is effective.

		NEGRA		TIGER		DPTB	
Bi + Kitaev et al. (2019)		F_1	DF_1	F_1	DF_1	F_1	DF_1
+ $BERT_{base}$‡	*F*	90.64	68.06	88.94	66.11	94.55	71.97
	B	90.67	68.92	88.85	66.78	94.52	73.16
	F (& B)	90.83	68.47	89.17	66.99	94.51	72.09
	(F &) B	90.98	69.36	**89.37**	**68.76**	94.57	74.68
	B & F	90.96	69.40	89.29	68.44	94.54	74.73
	F & B	**91.03**	**70.40**	89.30	68.37	**94.60**	**74.82**
+ $BERT_{large}$ (+ $XLNet_{large}$)	*F*	92.98	70.00	91.59	71.05	95.38	73.81
	B	92.92	70.46	91.60	70.98	95.61	76.64
	F (& B)	93.14	71.65	91.82	73.05	95.68	76.73
	(F &) B	**93.61**	**77.59**	**91.95**	74.21	95.74	78.62
	B & F	93.58	76.38	91.93	**74.35**	**95.77**	**78.96**
	F & B	**93.61**	76.97	91.89	73.93	95.73	78.46

Table 4: Ablation results of bi-directional action parsers on the test splits. *F* trains on forward graph and performs forward placement, *B* trains on backward graph and performs backward placement. *F (& B)*, *(F &) B*, *B & F* and *F & B* train on bi-directional graph where *F (& B)* performs forward placement, *(F &) B* performs backward placement, *B & F* performs backward placement and forward adjustment, and *F & B* performs forward placement and backward adjustment.

5 Conclusion

In this work, we build fast and accurate discontinuous constituency parsing systems working in continuous way by innovatively propose a novel reordering method. Results show that our method is able to preserve the high performance of the pseudo-continuous constituency parser with high efficiency. We not only achieve state-of-the-art results on two treebanks, but also demonstrate that the bottleneck of discontinuous constituency parsing is reordering.

Limitations

The method has only been validated in two languages: English and German. Therefore, the universality of the method needs to be further verified. The metrics for evaluating action graph parser, which further determine the performance of reordering, are not effective enough that limit the selection of the best action graph parsing model. We have tried using the RF_1 score of reordered pseudo-continuous sequences as the metric for action graph parsing model selection, but have gotten similar results to using the LF_1 score. The performance of our systems are still behind that of the pseudo-continuous constituency parser using gold reordered sequences. Therefore, the research on improving the reordering still needs to be carried on.

References

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The tiger treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, volume 168, pages 24–41.

Harry Bunt, Jan Thesingh, and Ko van der Sloot. 1987. Discontinuous constituents in trees, rules, and parsing. In *Third Conference of the European Chapter of the Association for Computational Linguistics*.

Maximin Coavoux. 2021. BERT-proof syntactic structures: Investigating errors in discontinuous constituency parsing. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3259–3272, Online. Association for Computational Linguistics.

Maximin Coavoux and Benoît Crabbé. 2017. Incremental discontinuous phrase structure parsing with the GAP transition. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1259–1270, Valencia, Spain. Association for Computational Linguistics.

Maximin Coavoux, Benoît Crabbé, and Shay B. Cohen. 2019. Unlexicalized transition-based discontinuous constituency parsing. *Trans. Assoc. Comput. Linguistics*, 7:73–89.

Caio Corro. 2020. Span-based discontinuous constituency parsing: a family of exact chart-based algorithms with time complexities from O(nˆ6) down to O(nˆ3). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2753–2764, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Amit Dubey and Frank Keller. 2003. Probabilistic parsing for German using sister-head dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 96–103, Sapporo, Japan. Association for Computational Linguistics.

Kilian Evang and Laura Kallmeyer. 2011. PLCFRS parsing of English discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 104–116, Dublin, Ireland. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Discontinuous constituent parsing with pointer networks. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7724–7731. AAAI Press.

Daniel Fernández-González and André F. T. Martins. 2015. Parsing as reduction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1523–1533, Beijing, China. Association for Computational Linguistics.

Daniel Fernández-González et al. 2021a. Discontinuous grammar as a foreign language. *arXiv preprint arXiv:2110.10431*.

Daniel Fernández-González et al. 2021b. Reducing discontinuous to continuous parsing with pointer network reordering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10570–10578, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Daniel Fernández-González et al. 2022. Multitask pointer network for multi-representational parsing. *Knowl. Based Syst.*, 236:107760.

Shexia He, Zuchao Li, Hai Zhao, and Hongxiao Bai. 2018. Syntax for semantic role labeling, to be, or not to be. In *Proceedings of the 56th annual meeting of the association for computational linguistics (Volume 1: Long papers)*, pages 2061–2071.

Laura Kallmeyer. 2010. *Parsing Beyond Context-Free Grammars*. Cognitive Technologies. Springer.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

Zuchao Li, Kevin Parnow, and Hai Zhao. 2022. Incorporating rich syntax information in grammatical error correction. *Information Processing & Management*, 59(3):102891.

Zuchao Li, Masao Utiyama, Eiichiro Sumita, and Hai Zhao. 2021a. Unsupervised neural machine translation with universal grammar. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3249–3264, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zuchao Li, Hai Zhao, Shexia He, and Jiaxun Cai. 2021b. Syntax role for neural semantic role labeling. *Computational Linguistics*, 47(3):529–574.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Wolfgang Maier. 2015. Discontinuous incremental shift-reduce parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1202–1212, Beijing, China. Association for Computational Linguistics.

James D McCawley. 1982. Parentheticals and discontinuous constituent structure. *Linguistic Inquiry*, 13(1):91–106.

Stefan Müller. 2004. Continuous or discontinuous constituents? a comparison between syntactic analyses for constituent order and their processing systems. *Research on Language and Computation*, 2(2):209–257.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Thomas Ruprecht and Richard Mörbitz. 2021. Supertagging-based parsing with linear context-free rewriting systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2923–2935, Online. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology, IWPT 2005, Vancouver, Canada, October 9-10, 2005*, pages 125–132. Association for Computational Linguistics.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. 2013. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, Washington, USA. Association for Computational Linguistics.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Fifth Conference on Applied Natural Language Processing*, pages 88–95, Washington, DC, USA. Association for Computational Linguistics.

Miloš Stanojević and Raquel G. Alhama. 2017. Neural discontinuous constituency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1666–1676, Copenhagen, Denmark. Association for Computational Linguistics.

Miloš Stanojević and Mark Steedman. 2020. Span-based LCFRS-2 parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 111–121, Online. Association for Computational Linguistics.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada. Association for Computational Linguistics.

Kailai Sun, Zuchao Li, and Hai Zhao. 2021. Multilingual pre-training with universal dependency learning. *Advances in Neural Information Processing Systems*, 34:8444–8456.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting on Association for Computational Linguistics*, ACL '87, page 104–111, USA. Association for Computational Linguistics.

David Vilares and Carlos Gómez-Rodríguez. 2020. Discontinuous constituent parsing as sequence labeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2771–2785, Online. Association for Computational Linguistics.

Xinyu Wang, Jingxian Huang, and Kewei Tu. 2019. Second-order semantic dependency parsing with end-to-end neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4609–4618, Florence, Italy. Association for Computational Linguistics.

Xinyu Wang and Kewei Tu. 2020. Second-order neural dependency parsing with message passing and end-to-end training. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 93–99, Suzhou, China. Association for Computational Linguistics.

Kaiyu Yang and Jia Deng. 2020. Strongly incremental constituency parsing with graph neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 21687–21698. Curran Associates, Inc.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria. Association for Computational Linguistics.

A Reordering Strategy

Set the priority of each node in the discontinuous tree to the smallest one among the priorities of its children, and the priority of a leaf node is its word index in the original sentence. Assuming that all sibling nodes are sorted by priority from small to large, the word order given by a pre-order or post-order traversal of the discontinuous tree is the target pseudo-continuous sequence. The pseudo-continuous tree can be obtained by adjusting the original sentence into the pseudo-continuous sequence.

During the experiment, we convert the discontinuous treebank into a pseudo-continuous treebank by the following process: The training data for discontinuous constituency parsing can be represented in parentheses notation, where the integer to the left of '=' indicates the index of the word, which is to the right of '=', in the original sentence. From left to right, the parenthesis notation shows a pre-order traversal of the discontinuous tree, so the words from left to right are already in our target order. We only need to remove '=' and the index to its left to get the training data for pseudo-continuous constituency parsing.

B Action Graph Dataset Statistics

In Table 6, we list the statistical results of the action graph datasets. The forward and backward graphs based on the same treebank contain about the same number of edges. The number of label `SPAN` is 0.5 to 0.7 times that of `MOVE-L` (or `MOVE-R`), indicating that more than half of the words are moved by a form of continuous segments. Since the bi-directional action graph parser we used contains three second-order scores of edge, we count the number of occurrences of these three kinds of second-order pairs in the bi-directional action graph datasets. Because of the chain structure of uni-directional action graph, the number of grand-parent is very large, followed by the number of co-parent, and the number of sibling is small but not negligible. Therefore, it is reasonable to use the second-order parsing model as our bi-directional action parser.

C Hyper-parameters Detail

Architecture hyper-parameters	
BiLSTM encoder layers	3
LSTM layer dimension	600
LSTM layer dropout	0.33
Word embedding dimension	100
Char embedding dimension	50
Char LSTM hidden dimension	400
Char LSTM output dimension	100
$\text{BERT}_{\text{base}}$ embedding dimension	100
Embedding dropout	0.2
$\text{BERT/RoBERTa/XLNet}_{\text{large}}$ encoder dimension	1024
Pre-trained encoder dropout	0.1
Using BERT layers	4
Edge FNN dimension	600
Label FNN dimension	600
Pair FNN dimension	150
Edge/Pair FNN dropout	0.25
Label FNN dropout	0.33
Head MLP dropout	0.2
Head MLP activation function	LeakyRelu
Training hyper-parameters	
λ	0.1
β_1, β_2	0.9
Decay rate	0.75
Gradient clipping	5
Batch size (tokens)	5000
Initial learning rate for $\text{BERT}_{\text{base}}$	5e-4
Initial learning rate for $\text{BERT/RoBERTa/XLNet}_{\text{large}}$	2e-5
Warm up for $\text{BERT/RoBERTa/XLNet}_{\text{large}}$	0.1
Epoch for $\text{BERT/RoBERTa/XLNet}_{\text{large}}$	30

Table 5: The hyper-parameters of action graph parsing models.

D Action Graph Parsing Results

In Table 7, we show the results of the action graph parsers on dev splits. For the same kind of action graph, the trend of LF_1 score is positively correlated with the reordering accuracy. However, the scores of different kinds of action graph, including the bi-directional graph, the forward graph and the backward graph, cannot be directly compared. Therefore, although it is reasonable to use LF_1 as the metric to select the best action graph parser, it cannot be used as a stable measure of reordering accuracy.

Dataset		NEGRA Train	NEGRA Dev	NEGRA Test	TIGER Train	TIGER Dev	TIGER Test	DPTB Train	DPTB Dev	DPTB Test
	n sent	18,602	1,000	1,000	40,472	5,000	5,000	39,832	1,700	2,416
F	n MOVE-L	7,534	406	450	17,391	1,695	2,245	12,583	462	834
	n SPAN	5,511	298	320	12,882	1,225	1,663	5,517	191	362
B	n MOVE-R	6,752	359	392	15,244	1,543	1,968	12,770	461	854
	n SPAN	4,318	234	260	9,380	1,016	1,222	6,767	248	474
F & B	n sibling	470	16	28	1,281	69	150	1,172	29	96
	n co-parent	1,084	61	83	2,925	176	365	4,240	128	283
	n grandparent	14,527	796	886	34,267	3,109	4,412	25,156	814	1,723

Table 6: Statistics of action graph datasets.

Action Graph Parser		NEGRA L-Rec	NEGRA L-Prec	NEGRA LF_1	TIGER L-Rec	TIGER L-Prec	TIGER LF_1	DPTB L-Rec	DPTB L-Prec	DPTB LF_1
Bi + $BERT_{base}$ †	*F & B*	78.18	83.53	80.76	77.73	81.26	79.46	81.57	78.46	79.99
Bi + $BERT_{large}$	*F & B*	82.73	87.45	85.02	83.21	85.86	84.51	83.26	77.78	80.43
Bi + $RoBERTa_{large}$	*F & B*	81.46	83.35	82.39	81.40	83.30	82.34	84.64	79.25	81.85
Bi + $XLNet_{large}$	*F & B*	-	-	-	-	-	-	84.95	79.41	82.08
Uni + $BERT_{large}$	*F*	84.94	85.06	85.00	82.81	85.87	84.31	-	-	-
	B	84.49	84.77	84.63	82.26	86.31	84.23	-	-	-
Uni + $XLNet_{large}$	*F*	-	-	-	-	-	-	84.92	79.17	81.94
	B	-	-	-	-	-	-	89.84	79.82	84.54

Table 7: Action graph parsing results on the dev splits.