TOKYO METROPOLITAN UNIVERSITY

DOCTORAL THESIS

Neural Combinatory Constituency Parsing

Author:
Zhousi CHEN

Supervisor:
Prof. Mamoru KOMACHI

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Natural Language Processing Group
Graduate School of Systems Design

March 21, 2023

Declaration of Authorship

I, Zhousi CHEN, declare that this thesis titled, "Neural Combinatory Constituency Parsing" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Publication

Conference Paper

- Zhousi Chen, Longtu Zhang, Aizhan Imankulova, and Mamoru Komachi. 2021. Neural Combinatory Constituency Parsing. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2199 – 2213, Online.

Journal Paper

- Zhousi Chen and Mamoru Komachi. Discontinuous Combinatory Constituency Parsing. *Transactions of the Association for Computational Linguistics*. (accepted in 2022)

TOKYO METROPOLITAN UNIVERSITY

Abstract

Faculty of Systems Design
Graduate School of Systems Design

Doctor of Philosophy

Neural Combinatory Constituency Parsing

by Zhousi CHEN

Constituency parsing is a task for explicit grammatical structures in natural language, which raises questions such as what phrases are in a sentence (e.g., noun phrase, verb phrase etc.) and what relationship one phrase has with another (i.e., including, being included or none). This information is useful when we need to precisely understand the encoded syntactic and even semantic information (e.g., who did what to whom).

Constituency parsers generally exhibit higher computational complexity, comparing to many other non-structural natural language processing (NLP) models (e.g., language modeling, textual entailment, machine translation etc.) and even structural dependency parsers. The balance between *efficiency and effectiveness* is an unavoidable topic. Some constituency parsers still remains in concept because of their high computational costs. Meanwhile, explicit syntactic structure becomes less useful in the face of adaptive neural network creating internal implicit structure. Some tasks which were once informed by structural information, such as phrase-based machine translation, now becomes more independent. We call for a simple, versatile, efficient, and effective constituency parsing family of modern neural technology.

In this research, I contribute a *neural combinatory constituency parsing* (**NCCP**) family which not only fast and accurately builds parsing structures with input words and simple atomic actions, but also can perform other two structural tasks: *word segmentation* and *fine-grained sentiment analysis*. I implemented four members as my main contribution and propose two members in concept with data conversion for future study as extra contribution. To the best of my knowledge, all four implemented **NCCP** members stand out with the highest speed, small memory footprint, and near state-of-the-art accuracy for constituency parsing tasks. They also reflect linguistic properties during and after the training process.

The four implemented **NCCP** members correspond to four combinations of two aspects of tree-based constituency parsing for one of {*continuous*, *discontinuous*} task in either {*binary*, *multi-branching*} style. Both tasks assumed all parsing structures are trees, where each child phrase attaches to only one parent phrase. The difference lies in whether a phrase can have more than one continuous spans of input words: all phrases in continuous constituency parsing must have exact one continuous spans; any phrases in discontinuous constituency parsing may have more than one continuous spans. Discontinuous constituency parsing is a better formalism for natural language. However, it is also more complex and challenging. For the genre of graph-based parser, which searches through all parsing possibilities, the complexity explodes from polynomial $O(n^3)$ for continuous parsing to exponential $O(n^{3f})$ for discontinuous parsing, where f is the number of continuous spans of a phrase. **NCCP** parsers are not graph-based and their empirical complexities are close to linearity. By systematically exploring the choice of neural components, those parsers can be competitive to graph-based parsers and keep high parsing speed at the same time.

Specifically, all **NCCP** members are neural combinators, which compose phrase embeddings from word embeddings in an iterative bottom-up style. Each member is equipped with a few different neural components that produce binary yes-or-no actions to create the unlabeled tree structure and guide the neural combinators to compose embeddings for constituent category prediction. The neural components include $O(1)$ feedforward neural network, $O(n)$ recurrent neural network, and $O(n^2)$ biaffine attention. Without any imposed grammatical constraints, the parsing speeds soar because of a strong linear tendency of their empirical complexities theoretically bounded by either $O(n^2)$ or $O(n^3)$. The expensive biaffine attention

is used within small scales. Moreover, thanks to the versatility of the composed embeddings, each parser can be easily adapted for *multilingualism*. For the continuous parsers, my binary parser facilitates hierarchical *fine-grained sentiment analysis*, and my multi-branching parser integrates useful *word segmentation* for languages without white space (e.g., Chinese and Japanese).

All four implemented **NCCP** parsers benefit from *data augmentation* and *training tricks*. For data augmentation, I create random substructures as dynamic training samples. In the case of binary models, my binarization extends Chomsky normal form which reflects a language's branching tendency. The multi-branching models also benefit from the random substructures and provide evidence for phrase headedness, which further shows positive effect for the accuracy of discontinuous parsing. As for training tricks, I exploit the property of discontinuous trees for create additional loss function to increase model robustness.

Nevertheless, tree structure is still limited for capturing the complex structure in natural languages. There are a variety of linguistic phenomena, such as A-movement, gapping, and right node raising, which create structure of *directed acyclic graph* (DAG) by child sharing (i.e., multi-attachment). I offer the conversion procedure of Penn Treebank, Penn Chinese Treebank for those DAG structures as test beds for future DAG-based parsers. Finally, I offer ideas for a conceptual **NCCP** pair for DAG parsing. All three pairs of **NCCP** pairs are consecutively proposed through extension of new functionality for structures from continuous tree to discontinuous tree, finally toward DAG.

The organization of this thesis comes into seven chapters. Chapter 1 recaps the basics of constituency parsing and relevant conception. Chapter 2 reviews recent parsing technology and issues. Chapters 3–5 include my main contribution. In Chapter 3, I specify the published four NCCP models, data augmentation and training tricks, which is experimented in Chapter 4. Chapter 5 follows with the discussion for the experiment and results. Chapter 6 specifies the DAG conversion for PTB and CTB with a conceptual NCCP pair for DAG. Finally, Chapter 7 summarizes this thesis and describes the prospects of this research with my future study.

Acknowledgements

First, please allow me to heartily thank to my supervisor Professor Mamoru Komachi, who create the stable research environment and reliable relationship with all his students. He relentlessly offers me guidance and constructive advice, and helps my research and even part of my abroad life, as if we were his own family members. As addressed in an classic Chinese book (《禮記》, Book of Rites, Liji), "為人師者, 必先正其身, 方能教書育人, 此乃師德之本也。" (That is, when teaching and educating people, teachers should not only teach by words, but also by their own thoughts and practices as examples, which is the teachers' professional ethics.) The knowledge and attitude of Professor Komachi to research and life will always influence me in future study and career.

Moreover, I would like to sincerely thank Professor Yuji Matsumoto and Professor Yusuke Miyao who provided inspiring and careful suggestions to my research. The research topic of constituency parsing sould have been quite serious to new researchers. As the pioneers who significantly contributed to the community Professor Matsumoto and Professor Miyao also helped and influenced me with their deep thoughts and insights. It is my honor to have them during my five-year PhD study in Tokyo.

Meanwhile, I would like to extend my thanks to my classmates and friends in the laboratory: Hayahide Yamagishi, Michiki Kurosawa, Longtu Zhang, Aizhan Imankulova, Masahiro Kaneko, Yuting Zhao, Tosho Hirasawa, Teruaki Oka, Oryza Siti Khairunnisa, Hongfei Wang, Zizheng Zhang, Xiaomeng Pan, and many others I could not included here. It is my very luck to meet you and share common memories about our lives and studies in Western Tokyo. Specially, I thank Hayahide Yamagishi as well as other students in Komachi Lab for discussing the very initial idea of my research and encouraging me to proceed. Professor Komachi allows me to shift my research topic from machine translation to constituency parsing, which appears to be an intricating world to me.

Finally, I must heartily thank my family members and intimate friends in China. My parents raised me and always let me have access to better education without any hesitation. Beyond thousands of miles back to my hometown Guiyang City, they send me indispensable warmth of emotional and financial supports, so that I can focus on my research here. I really miss all of you. During the years of COVID-19 pandemic, when travelling and communication become less convenient, many people still keep on making the world a better place in their own ways. Please let me show my best respects to you, even though I might not know you.

Sincerely thank you.

Contents

Declaration of Authorship iii

Abstract viii

Acknowledgements xi

1 Introduction 1
- 1.1 Early Facts of Constituency Parsing . . . 1
- 1.2 Constituency Parsing . . . 2
  - 1.2.1 Tree Structure . . . 3
    - Continuous Tree by Context-Free Grammar . . . 3
    - Discontinuous Tree by Linear Context-Free Rewriting System 4
  - 1.2.2 DAG Structure . . . 6
  - 1.2.3 Headedness and Lexicalization . . . 7
    - Head-driven Phrase Structure Grammar . . . 8
    - Dependency Parsing: Non-projectivity vs. Discontinuity . . . 9
    - Combinatory Categorical Grammar . . . 10
- 1.3 Related Task . . . 11
  - 1.3.1 Chunking: Word Segmentation and Shallow Parsing . . . 11
  - 1.3.2 Structured Semantic Task: Sentiment Analysis . . . 11
  - 1.3.3 Task for DAG structure . . . 12
- 1.4 Thesis Structure . . . 13
- 1.5 Contribution . . . 13

2 Constituency Parser 15
- 2.1 Transition-based Parser . . . 15
  - 2.1.1 Finite Automata . . . 15
  - 2.1.2 Parsing with Sequence Labeling: Tree Linearization . . . 17
  - 2.1.3 Parsing with Iterative Chunking . . . 20
  - 2.1.4 Summary . . . 20
- 2.2 Chart-based Parser . . . 21
  - 2.2.1 Binary Chart Parser: from CKY Algorithm . . . 21
  - 2.2.2 N-ary Chart-based Parser: from Earley Algorithm . . . 22
- 2.3 Joint and Unsupervised Task . . . 24
  - 2.3.1 Joint Constituency and Dependency Parsing . . . 24
  - 2.3.2 Unsupervised Constituency Parsing . . . 24

3 Neural Combinatory Constituency Parsing 25
- 3.1 Continuous Ply . . . 25
  - 3.1.1 CB: Orientation . . . 26
  - 3.1.2 CM: Chunking . . . 27
- 3.2 Discontinuous Ply . . . 27
  - 3.2.1 DB: Swap and Joint . . . 27

3.2.2 DB: Medoid and Affinity Biaffine Attention . . . 28
3.3 Data and Augmentation . . . 29
3.3.1 Empty node and unary branch . . . 30
3.3.2 Binarization for CB and DB . . . 30
3.3.3 Medoid for DM . . . 31
3.3.4 Oracle . . . 31
3.4 Model Implementation . . . 32
3.4.1 Neural Component . . . 32
3.4.2 Pre-trained Word Embedding and Language Model . . . 34
3.4.3 Overall Architecture . . . 34
3.4.4 CB & DB: Binary Implementation . . . 34
3.4.5 CM & DM: Multi-branching Implementation . . . 37
3.4.6 Multilingualism and Structured Sentiment Analysis . . . 38
3.5 Model Robustness . . . 39
3.5.1 Additional Substructure with Empty Node . . . 39
3.5.2 Basic Loss Item . . . 40
3.5.3 DB Robustness Loss Item: Ply Shuffle . . . 40
3.5.4 DM Robustness Loss Item: Intra- and Interply . . . 41

4 Experiments **43**
4.1 General Setting . . . 43
4.2 Continuous Parser . . . 44
4.2.1 Constituency Parsing . . . 44
Overall Results . . . 44
Ablation Study . . . 46
Tree-Binarization Strategy . . . 47
Complexity and Speed . . . 48
4.2.2 Structured Sentiment Analysis . . . 48
Single Task Result . . . 49
Joint Task Result . . . 49
4.3 Discontinuous Constituency Parsing . . . 50
4.3.1 Overall Results . . . 51
4.3.2 Ablation Study . . . 53
4.3.3 Inference with Unsupervised Headedness . . . 54
4.4 Multilingualism and Word Segmentation for Chinese and Japanese . 54

5 Discussion **55**
5.1 Feature of NCCP Family . . . 55
5.1.1 Compact Neural Combinator . . . 55
5.1.2 CB & DB Orientation: Branching Tendency . . . 56
5.1.3 CM & DM Unsupervised Weight: Headedness . . . 57
5.1.4 Multilingualism . . . 58
5.1.5 Beyond Constituency Parsing . . . 59
5.2 Empirical Complexity and Speed . . . 60
5.3 Accuracy and Robustness . . . 61
5.3.1 Contextualization and Length . . . 61
5.3.2 Multi-branching Arity and Fan-out Degree . . . 62
5.3.3 Hyperparameter Tuning and Robustness . . . 62
5.3.4 Inference Error Rate . . . 64
5.4 Sample Analysis . . . 65
5.4.1 Continuous vs. Discontinuous Parsing . . . 65

5.4.2 Unsupervised Headedness in Japanese and Chinese 67
5.4.3 Structured Sentiment Analysis 67

6 DAG Conversion for PTB and CTB **69**
6.1 Conversion with Co-indexing 70
6.1.1 Trace with Null Element 70
6.1.2 Intra-sentential Gapping 71
6.2 Exception and Error Correction 72
6.2.1 Wrong Trace Type 73
6.2.2 Circle in Gapping 73
6.2.3 Ill-formed Coordination 74
6.2.4 Involving External Phrase 74
6.2.5 Difference of PTB and CTB 74
6.3 Summary and Discussion 74
6.3.1 Data Statistics 74
6.3.2 Comparison with Combinatory Categorical Grammar 74

7 Conclusion **77**
7.1 Conclusion Remarks 77
7.2 Potential Social Impact 78
7.3 Weakness and Limitation 78
7.4 Future Work 79
7.4.1 GB: Replicate Node 79
7.4.2 GM: Multi-medoid Biaffine Attention 80
7.4.3 Prospects of NCCP family. 81

Bibliography **83**

List of Figures

1.1 Evang and Kallmeyer (2011) recover (b) discontinuity from (a) continuous Penn Treebank with trace nodes (blue). We simplified the trace symbol for this demonstration. . . . 4
1.2 German is a verb-second word order (V2) language, where the second position from left of the main clause is occupied by the verb phrase. Object and other argument can relatively have free positions with their case markers and inflection as the sign of its grammatical role. . . . 5
1.3 PTB has traces for gapping remnants (a, noun phrases suffixed with =1) for recovering the DAG structure (b). We denote different remnant groups in different colors (i.e., blue, green, and yellow) and shared components with double lines. The initial coordinate in red contains the full structure. . . . 7
1.4 A PTB annotation with control verb *ask* of 6d. We keep the original annotation of SBJ, which shows *it* works as a different argument (i.e., an A-movement to be a SuBJect). There is a further A-movement because of the semantic requirement for the passive voice. Word *it* works again as an object for verb *gone*. . . . 8
1.5 Given the head information, constituency tree can be converted into dependency graph. Constituency discontinuity and dependency non-projectivity root in the same syntactic structure of this sentence. . . . 10
1.6 (a) contains samples from Chinese Treebank and Keyaki Treebank. (b) and (c) are from a sample from CoNLL 2003 dataset, which provides phrase chunk and NER annotations in parallel. . . . 12
1.7 Two semantic tasks. Left: fine-grained sentiment analysis with Stanford Sentiment Treebank. Right: DAG-based event detection with BioNLP CG shared task 2013 (Pyysalo et al., 2015; Espinosa, Miwa, and Ananiadou, 2019). . . . 12

2.1 Binary tree linearizations by (a) Kitaev and Klein (2020) and (b) Wei, Wu, and Lan (2020). Corresponding edges are denoted in the same colors. Numbers in (a) enumerates the build sequence: both are from left to right. Linearized structural actions are at the bottom of (a) and in green circles of (b). . . . 17
2.2 Binary tree linearizations by (c) Shen et al. (2018b) and Multi-branching by (d) Gómez-Rodríguez and Vilares (2018). Numbers of non-terminals enumerates the build sequence: the former (c) divides and conquers by the order of syntactic distance, while the latter (d) follows the default left-to-right order. Linearized structural actions are at the bottom of (c) and (d). . . . 18
2.3 Multi-branching tree linearization by (e) Yang and Deng (2020). Numbers enumerates the possible positions in *rightmost chain* where actions can take place. . . . 19

2.4 Concurrent bottom-up steps of CKY algorithm by span length in a chart. This exemplary chart has maximum span length four for four terminals. 21
2.5 Instead of description from the respective of spans, the combinatorics of both continuous and discontinuous are shown as tuples of bits representing the existence of terminal symbols. Different constituency dependencies are shown using different line types and colors. 22

3.1 Two adjacent continuous plies at heights h and $h+1$. Ply_h is the current layer of subtrees and their actions (i.e., **CB**'s *orientation*s or **CM**'s *chunk*s) result in an intermediate ply'_h with probably inconsecutive node indices. We reenumerate them into a new ply_{h+1} for the next actions. Empty nodes are placeholders marked by gray circles. 26
3.2 Two adjacent discontinuous plies. In addition to continuous plies, **DB** inherits the *chunk* for the swap-joint system and **DM** gets *biaffine attention* action graph described in a matrix. 28
3.3 Tree binarization with numerical $\rho^{\mathbf{B}}$. 30
3.4 Signals from a discontinuous tree. The original m-ary tree (a) is binarized and stratified into (b) and (c) with numeric factors $\rho^{\mathbf{B}}$, whereas (a) is stratified into (d) with a categorical *medoid* factor *random*. In (4), w_2 and w_5 are randomly selected as *medoids* for discontinuous parents l_1^2 and l_2^2 with more or less twisted descendant lines. I color constituent components and show *joints* with light blue "⬥" and "⋄" which are not activated by agreeing orientation condition. 32
3.5 Example of **CB**. The simplest member of the **NCCP** family solely based on *orientation* for the tree structure. 35
3.6 Example of **CM**. The *chunk* signals are calculated from the difference of the forward and backward states of the $\texttt{BiLSTM}_{ply}$. Exemplary fractions show the gate weights under Softmax constraints for vector compositionality, which are unsupervised and offer evidence for phrase headedness. The same mechanism works for **DM**. 35
3.7 Example of **DB**. **DB** produces *swap* to facilitate traveling of discontinuous nodes and *joint* to combine adjacent nodes. 36
3.8 Example of **DM**. **DM** leverages *affinity* biaffine attention to identify and combine discontinuous groups. 36
3.9 A example for the creation of a random ∅ branch from a flat tree. . . 39
3.10 (a) Constituent children get shuffled and create additional losses. (A_1, A_2, A_3), B_1, and (C_1, C_2) belongs to three different constituents. (b) Pick in-ply continuous nodes and interply nodes for **DM** biaffine attention. . . . 40

4.1 Grid search with an interval of 0.1 in a space of (tag, label, orientation) loss coefficients. The best was (0.2, 0.3, 0.5) indicated by an arrow. . . 47
4.2 Probabilistic interpolations of two CNF factors to F1 scores. The capacity of $\texttt{BiLSTM}_{cxt}$ is almost saturated with 6 or 8 layers. 47
4.3 Linear complexity vs. squared complexity. Redundancy with placeholder "<0>" helps maintain the triangular shape. 48

5.1 Top: **DB** signal polarity to $\rho^{\mathbf{B}}$ with *orientation right* "•" and *joint* "▲". Bottom: **DM** signal polarity to stratifying *medoid* factor $\rho^{\mathbf{M}}$ with *affinity* "•", *joint* "▲", and *discontinuity* "■". *Continuous* and *head* are referential only, looking for the least *discontinuity* and leveraging head information. (All $\rho_{\varnothing} = 0$.) 56

5.2 Beta distribution visualization for TIGER **DB** at **S2**. The line thickness corresponds to the development F1 score. See their other hyperparameters in Figure 5.6. 57
5.3 Empirical complexities of **NCCP** parsers for each corpus with linear regressions (LR) are shown on a light blue background when the quadratic terms are negative for **CB** and **CM** (left) or direct linear and quadratic coefficient scatter for **DB** and **DM** (right). For DPTB and TIGER, $\rho^{\mathrm{B}} \in \{0, 0.25, 0.5, 0.75, 1\}$ and $\rho^{\mathrm{M}} \in \{\textit{continuous}, \textit{head}, \textit{random}, \textit{leftmost}, \textit{rightmost}\}$ are used for the scatter. Cubic LR gives all negative cubic terms highly close to zero. 60
5.4 XLNet provides an overall improvement for **CB** and **CM** on length bins. All models find it challenging to handle long sentences. 60
5.5 **DB** and **DM**'s discontinuity and multi-branching performances. **DM** keeps some points ahead in terms of F1 scores on TIGER richer in discontinuity. 61
5.6 The BO process starts with **S1** dev F1 scores (i.e., a small dot at each legend bottom) and ends with a range of scores in **S2**. While the models are not sensitive to hyperparameters (e.g., all gains are less than 0.54), their preferences are different on respective corpora. 63
5.7 Failed parse from the multi-branching model. The model stops parsing and saves computations when it repeats the same chunking positions. 64
5.8 The numbers of tries to decompose *biaffine attention* matrices. `FAIL`s are marked with "▲". 64
5.9 An exact matched DPTB sample from PLM **DB** and **DM** models versus **CB** and **CM** on PTB. The parse contains complex nested clauses which **CM** must fail to capture, and it becomes ungrammatical in the continuous scenario. **DB**'s outputs include orientations depicted as arrows and their traveling traces colored for groups. Meanwhile, **DM** produces two *biaffine attention* matrices, one of which has a highly biased but correct threshold $\theta = 0.99$. Bar heights indicate values in matrices and their colors indicate the relationship to θ. 65
5.10 A TIGER parse. **DB** natively with ∅-subtrees achieved the exact match but **DM** erred with $\rho_{\varnothing} = 0$. 66
5.11 A semantic ∅-subtree by **DM** with $\rho_{\varnothing} > 0$. Copula "were" has less *affinity* than "when" and "due". 66
5.12 Chinese (top) and Japanese (bottom) parses from **CM** model. 67
5.13 Samples from SST (upper parts) and its PWE**CB** prediction (bottom parts). Sentiment labels are color in red for positive and blue for negative. Each **CB** prediction is a string of five labels sorted by their scores and the leftmost label is the result. 68

6.1 DAG conversion with both A-movement (*) and A-bar movement (***T***). 71
6.2 DAG conversion with gapping coindexing marks. 71
6.3 Two annotating errors in PTB causing the same index for different trace types. Top: `*ICH*` is supposed to be `*RNR*`. Bottom: `*T*-1` should be a null element for null complementizer *0* instead of *I*. . . . 73

6.4 Another three annotating error or exceptions in PTB. Top (c): the original annotation leads to a directed cyclic graph instead of DAG. This is a rare case in PTB and I choose to reduce it to DAG by manually removing one or more paths. Middle (d): some samples do not follow PTB guidelines and miss some phrases structure for coordination. I manually add such phrases by referring to the template phrase. Bottom (e): three samples involve both expletive (***EXP***) and coordination, which make Algorithms 5 & 6 miss key components. I manually change the coordination of `VP` into `S` and the construction for grammatical and symmetric coordination. 76

7.1 **GB** action cases. 79

List of Tables

2.1 An example of the RNNG shift-reduce parsing process. 16
2.2 Chunker-based parsing with BIOE prefix (i.e., for beginning, inside, outside, and ending). 20

4.1 Single-model results on PTB and CTB test datasets sorted by the F1 scores on PTB. Transition-based parsers, chart parser, and others are marked as T, C, and O, respectively; ↑ and ↓ denote bottom-up and top-down. The number in brackets indicates the beam size. Speeds are measured in sentences per second. Kitaev and Klein (2018) used Tesla K80, and the CTB scores are cited from Kitaev, Cao, and Klein (2019). Zhou and Zhao (2019) used GeForce GTX 1080 Ti (same condition). 45
4.2 Improvements with pre-trained language models. I used a greedy search algorithm on single GeForce GTX 1080 Ti. Rows 6–8 are reported by Yang and Deng (2020) using GeForce GTX 2080 Ti. Kitaev and Klein (2020) used a cloud TPU with a beam search algorithm and a larger batch size. 45
4.3 Effectiveness of using frozen static word embeddings or dynamic subword language model and corresponding peak speed. 45
4.4 Results of ablation studies on fastText (top) and $\texttt{BiLSTM}_{ply}$ (bottom) of the binary model. **CB**/V is the main experimented **CB** variant in other subsections. 46
4.5 Training time and memory consumed by my two data formats. The time column indicates the time used for 150 training epochs with validations. Development F1 scores are approximately 92.4. The OOM column lists the length limit for preventing an out-of-memory error. Kitaev and Klein (2018) took 10 hours for 93 training epochs on GeForce GTX 1080 Ti to yield their results. 48
4.6 Models on under weakly comparable conditions. All models are trained or fine-tuned only on the Stanford Sentiment Treebank without joint training on other treebanks. All **CB** models are trained with no structural loss (i.e., $\gamma = 1$). 49
4.7 Syntactic information seems not helping sentiment classification. Both induction of orientation loss and joint training with PTB decrease the sentiment classification accuracy. The F1 scores are 92.5 and 95.7, respectively for PWE and PLM **CB**. 50
4.8 Overall performances from recent works. Speeds in sentences per second are reported on incomparable hardware and software platforms. Ours and Vilares and Gómez-Rodríguez, 2020, VG20 are on GeForce GTX 1080 Ti with PyTorch implementation and Fernández-González and Gómez-Rodríguez, 2021, FG21 on GeForce RTX 3090. Fernández-González and Gómez-Rodríguez, 2022, FG22 involved lexical dependency information†. 52

4.9 Means and standard deviations of five runs on test sets with four significant digits. **DM** ourperforms **DB**. Development sets reflect similar variability. 52
4.10 Ablation in two-stage training with dev scores. Triplets in $\{0,1\}$ stand for turning on and off ($\rho_\varnothing$, $\rho^{\mathbf{B}} \sim \text{Beta}(1,1)$, *shuffle*) for **DB** and ($\rho_\varnothing$, β_c, β_x) for **DM**. Variants of "‡" are **S1** — the start of **S2**. 53
4.11 **DM** is sensitive to $\rho_\varnothing$ with dev F1 scores. 53
4.12 **DM** medoid factor $\rho^{\mathbf{DM}}$ = *uhead* offers stable gains even without head information during training. I tested $\rho^{\mathbf{DM}}$ = *random* five times. 53
4.13 F1 scores of monolingual and multilingual continuous **NCCP** parsers. For CTB and KTB, **CM** extends its chunking function for word segmentation during training and **CM** is character-based for Chinese and Japanese but word-based for English. 54
4.14 F1 scores of monolingual and multilingual discontinuous **NCCP** parsers. 54

5.1 Parameter sizes of PWE **NCCP** parsers. 55
5.2 Frequencies of orientation with different CNF (biased) and referential only non-CNF (more balanced) $\rho^{\mathrm{B}} = 0.5$ in different stratified continuous treebanks for **CB**. 56
5.3 English headedness selection with my multi-branching model **CM** on PTB test set. **DM** also provide very similar statistics. "`*`" marks the absence of a `DT` child for its `NP` sisters. For quantifier phrases (`QP`), some non-quantifiers are more likely to be heads if they appear; e.g., adverbs (`RB`; e.g., "approximately"), prepositions (`IN`; e.g., "about"), and relative adjectives (`JJR`; e.g., "more than"). 58
5.4 Similar labels in different treebanks by their Euclidean distances in multilingual parsers' $\texttt{FFNN}^{k}_{label}$. 59
5.5 Multi-branching and discontinuous F1 scores of **DB** and **DM** on (D)PTB test sets. We grouped $k > 1$ because only one tree has fan-out $k = 2$ in the test set. The scores of **CB** and **CM** are from Chen et al. (2021). . . 61
5.6 Multi-branching and discontinuous test F1 scores of **DB** and **DM** on TIGER. Fan-out is detailed in k. 62
5.7 Errors in discontinuous **NCCP** PLM models with $\rho_\varnothing > 0$. A `FAIL` causes a matrix of ones, whereas a θ close to one gives an identity matrix — an expensive null action. 64

6.1 Statistics of PTB after my DAG conversion. PTB (2.0) has 49,208 sentences. 75
6.2 Statistics of CTB after my DAG conversion. CTB (9.0) has 132,080 sentences. 75

List of Abbreviations

CFG	**C**ontext-**F**ree **G**rammar
CNF	**C**homsky **N**ormal **F**orm
LCFRS	**L**inear **C**ontext-**F**ree **R**ewriting **S**ystem
GCFG	**G**eneralized **C**ontext-**F**ree **G**rammar
HPSG	**H**ead-driven **P**hrase **S**tructure **G**rammar
CCG	**C**ombinatory **C**ategorical **G**rammar
PoS	**P**art-**o**f-**S**peech
RNR	**R**ight **N**ode **R**aising
DAG	**D**irected **A**cyclic **G**raph
FFNN	**F**eed**f**orward **N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
LSTM	**L**ong **S**hort-**T**erm **M**emory
BiLSTM	**Bi**directional **L**ong **S**hort-**T**erm **M**emory
PWE	**P**re-trained **W**ord **E**mbeddings
PLM	**P**re-trained **L**anguage **M**odel
RNNG	**R**ecurrent **N**eural **N**etwork **G**rammars
NCCP	**N**eural **C**ombinatory **C**onsituency **P**arsing
CB	**C**ontinuous **B**inary
CM	**C**ontinuous **M**ulti-branching
DB	**D**iscontinuous **B**inary
DM	**D**iscontinuous **M**ulti-branching
PTB	**P**enn **T**ree**b**ank
CTB	**C**hinese Penn **T**ree**b**ank
KTB	**K**eyaki **T**ree**b**ank
DPTB	**D**iscontinuous **P**enn **T**ree**b**ank

Chapter 1

Introduction

1.1 Early Facts of Constituency Parsing

Talking about parsing, we usually mean extracting some form(s) of structures from strings of plain text. For human languages, the forms of syntactic structures are the first subject to the reign of grammar, which defines different languages. Noam Chomsky characterizes a language as "*a set (finite or infinite) of sentences, each of finite length, all constructed from a finite alphabet of symbols.*"(Chomsky, 1956) He provides a formal system, the Chomsky Hierarchy, which includes four levels of grammar to organize the finite alphabet symbols (i.e., words) into grammatical structures. From regular grammars (type-3) to context-free grammars (type-2) to context-sensitive (type-1) and finally to unrestricted grammars (type-0), the higher the level (i.e., lower n of type-n) becomes, the more complex syntactic structures of a language can be.

The Chomsky Hierarchy has a huge impact on parsing techniques for computer languages, because the formal grammars behind them are essential tools to interpret human inputs for controlling and interacting with computers. A parser is said to be Chomskyan when it meets the description of context-free grammar. Around 1950s and 1960s, many parsers (Lucas, 1978; Knuth and Pardo, 1980) got invented to convert formal languages (e.g., arithmetic expressions and statements) into low-level machine code (as complilers). Because context-free grammars are relatively simple, early parsers are rule-based with limited token string for programming purpose. Parsing natural languages was not quite feasible by grammar design or computational power at that time. However, the notion of phrase structure grammar became central, which led to constituency parsing for natural language.

Constituency parsing is thus one of the long-lived research topics. Two technical paradigms, graph-based parsing and transition-based parsing, have their roots back in 1950s and 1960s. Oettinger (1961) discovers deterministic pushdown automata (DPDA) by using stack-driven recursiveness for a small range of parsing results, which is essentially how modern neural transition-based parsers works. Sakai (1961) describes a table-driven parser, which systematically searches through all input combinations for all results. Several years later, his algorithm had a popular name after the names of its rediscovers, CKY algorithm (Cocke, 1969; Kasami, 1966; Younger, 1967). The term "table" was gradually placed by synonym "chart". These two paradigms of transition-based and graph-based parsing are examples for the trade off between speed and accuracy, which is still a main topic in this dissertation.

There are other prominent pioneers who laid the foundation of the modern constituency parsing literature from a general aspect, to name a few: Andrey Markov with Markov chain and its application (Markov, 1906; Markov, 1913), Claude Shannon with information theory (Shannon, 1948), Alan Turing with his discovery of

stack (Carpenter and Doran, 1977), John Backus and Peter Naur with their notation (Backus, 1959), and artificial neural network pioneers with their methodologies (David E. Rumelhart and Williams, 1986). The discovery of parsing methodologies for context-free grammars, such as top-down LL(k) (Lewis and Stearns, 1968), bottom-up LR(k) (Knuth, 1965), and Earley Algorithm (Earley, 1970), also helped the understanding of complexities in formal grammars. Modern researchers constantly learn and directly or indirectly gain insights from them.

1.2 Constituency Parsing

When referring to constituency parsing, we talk about *recursive phase structures* nested to form hierarchical syntactic structure. A grammatical sentence have at least one whole syntactic structure or one parse. For example, the following English sentence 1a has nested phrases 1b–1e combining into a parse:

1. (a) *A quick brown fox jumps over the lazy dog.* (as a sentence)
   (b) *a quick brown fox* (as a noun phrase)
   (c) *jumps over the lazy dog* (as a verb phrase)
   (d) *over the lazy dog* (as a preposition phrase)
   (e) *the lazy dog* (as a noun phrase)

As demonstrated above, sentence 1a contains a noun phrase 1b and a verb phrase 1c. Further, 1c contains a preposition phrase 1d which in turn consists of a preposition *over* and 1e. It looks certain to our eyes, whereas the following examples remind us that there are more ungrammatical interpretations than grammatical ones:

2. (a) *a quick brown* (incomplete)
   (b) *fox the* (incompatible, discontinuous)
   (c) *jump over* (not in the right context)

On the flip side, different grammatical parses for the same sentence reflect different interpretations of its meaning:

3. (a) *I shot an elephant in my pajamas.*
   (b) *shot in my pajamas* (a discontinuous verb phrase)
   (c) *an elephant in my pajamas* (a continuous noun phrase)

The verb phrase 3b indicates that the agent was wearing pajamas during the action. Meanwhile, the noun phrase 3c describes an unusual scene that an elephant appeared in the agent's pajamas, which is not a preferable parse.

Generally speaking, the task of constituency parsing is about finding such recursiveness of phrases toward terminal words in a sentence. Grammars are sets of rules that define and judge the correctness of phrasal recursiveness. Their types result in different levels of recursive structures: from *continuous tree*, to *discontinuous tree*, to *directed acyclic graph* (DAG). I will discuss the former two tree levels with their corresponding grammar formulations and explain how grammar types lead to different levels according to their definitions. Then, I briefly mention the DAG with examples and explanations. My focusing is not the grammars but the structures they form, which finally leads to my grammar-less parsers in Chapter 3.

1.2.1 Tree Structure

A tree structure embodies a parse when each child phrase belongs to exactly one parent phrase. The tree structure is the most common case in constituency parsing, which further divides into *continuous* and *discontinuous* trees, as exemplified by 3b and 3c. (Meanwhile, discontinuity in the context of dependency parsing refers to lexical *projectivity* and *non-projectivity*. See section 1.2.3.)

Informally speaking, continuous trees only contain continuous phrases, where each phrase has one continuous spans of words. In contrast, discontinuous trees can have discontinuous phrases, each of which can contain multiple spans of words.

Continuous Tree by Context-Free Grammar

To formally introduce continuous tree, I first refer to context-free grammar (Chomsky, 1956, CFG) which defines a formal language L and produce only continuous phrases. A CFG grammar G is defined by a 4-tuple $G = (\Sigma, N, R, S)$, where

$G =$
- Σ is a finite set of terminal symbols, which are actual words of a sentence;
- N is a finite set of non-terminal symbols, which generalizes the categories of phrases, e.g., a verb phrase denoted in `VP` and a noun phrase in `NP`;
- R is a finite set of production rules, each in the form of $A \to \alpha$, where $A \in N$ and $\alpha \in (N \cup \Sigma)^\star$ defines how the single left-hand side symbol generates a string of zero, one, or multiple terminal or non-terminal symbols;
- S is a designated start symbol with $S \in N$, which can only appear on the left-hand side of the production rules.

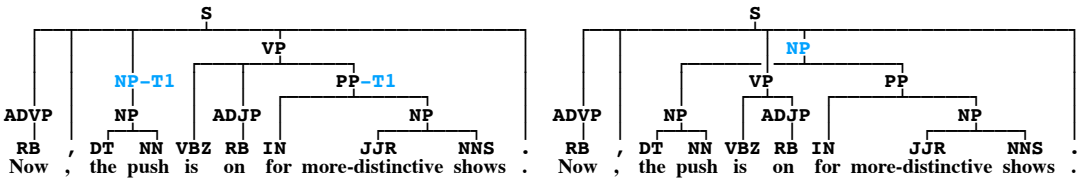
The essence of CFG grammar lies in the simple form $A \to \alpha$ of each production rule in R which not only defines the maximum one-to-many tree structure but also determines the adjacency and the order of right-hand side symbols. For example, rules of $A \to BC$, $B \to b$, and $C \to c$ generate the string bc. Under the constraint, G only generates continuous trees.

Syntactic ambiguities exist in CFG formal languages, where more than one grammatical derivation accounts for the exact same yield of words. For example,

4. (a) `NP` → *the lazy dog* (general)
   (b) `NP` → `A` *dog* → *the lazy dog* (left recursive with rule `A` → *the lazy*)
   (c) `NP` → *the* `B` → *the lazy dog* (right recursive with rule `B` → *lazy dog*)

`NP` of three words has three approaches to yield its content, where non-terminal `A` as well as `B` are intermediate steps. Such parsing ambiguities prevent some parser from being deterministic. Especially, binary parsers that relies on production rules whose right-hand side has no more than two symbols cannot direct handle 4a. Thus, the Chomsky normal form (CNF) facilitates breaking down the large grammar rules into smaller binary ones with their right-hand side having at most two symbols, in either form similar 4b or 4c. (CNF in the context for formal language has more conversion restrictions beyond our examples.)

The notion of *context-free* indicates that each production rule can be applied regardless of the context of a non-terminal, where nature languages have properties that are beyond context-sensitive languages in the Chomsky Hierarchy. In the 1980s, it became clear that natural languages are not context-free (Pullum and Gazdar, 1982). However, the continuous tree structure behind $A \to \alpha$ is very handy and captures many cases in English. Thus, the early creation of annotated parsing corpora



(a) Simplified PTB annotation with traces. (b) Discontinuous tree after conversion.

FIGURE 1.1: Evang and Kallmeyer (2011) recover (b) discontinuity from (a) continuous Penn Treebank with trace nodes (blue). We simplified the trace symbol for this demonstration.

(i.e., treebanks) largely uses parentheses to annotate continuous phrases as labelled spans. Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993, PTB) is among such practices of creating treebank. For example, the tree in 1a serializes into a string of `(S (NP A quick brown fox) (VP jumps (PP over (NP the lazy dog))))`.

Treebanks enable data-driven models for parsing nature languages. In addition to the bracketing format for the majority of continuous trees, PTB leverages co-indexing trace marks (Evang, 2011) to address discontinuity, as we will cover in the next subsection.

Discontinuous Tree by Linear Context-Free Rewriting System

Discontinuity of PTB is partly caused by temporal syntactic movement, whereas that of TIGER Treebank (Brants et al., 2004) is more due to the language's innate property. In the case of PTB, phrase components displace from the position where they receive their original grammatical role to form discontinuous phrases.

For example, as shown in Figure 1.1, the preposition phrase *for more-distinctive shows* does not act as any grammatical role for the verb phrase *is on*. Rather, it is a modifier for the noun phrase *the push*, specifying what purpose *the push* is for. Thus, there is a discontinuous noun phrase including a left-hand side noun phrase *the push* and a right-hand side *for more-distinctive show*, which both are marked by the simplified trace mark `T1`.

The original trace symbols used by PTB contain the types of the movements, such as `*T*` for *non-argument movement* (A-bar movement) with a type `T` and * for any *argument movement* (A-movement), and a trace identity number, such as `1`, to identify movements in different phrases. Arguments are grammatical roles played by each components for their parent phrases. For example, a noun phrase in English can play as a subject for a sentence or a clause, or as an object for a verb phrase, preposition phrase. We will cover this topic again in the next sections 1.2.2 and 1.2.3. In this section, we focus on one type description of discontinuity, Linear Context-Free Rewriting System (Vijay-Shanker, Weir, and Joshi, 1987; Weir, 1988, LCFRS).

LCFRS is a type of generalized context-free grammars (GCFG), which expands on CFG by adding potentially non-context-free composition functions to rewrite rules. Thus, LCFRS is not context-free as its name suggested. Specifically, a LCFRS G' is a 5-tuple $(\Sigma, N, R, \varphi, S)$, where

$G' =$ Σ is a finite set of terminal symbols; (the same definition as Σ of CFG.)

N is a finite set of non-terminal symbols; (the same definition as N of CFG.)

φ is a function that specifies a fan-out degree for each non-terminal, $\varphi : N \to \mathbb{N}$, which indicates the number of continuous spans within a discontinuous phrase (e.g., the discontinuous `NP` in Figure 1.1 has fan-out degree 2.)

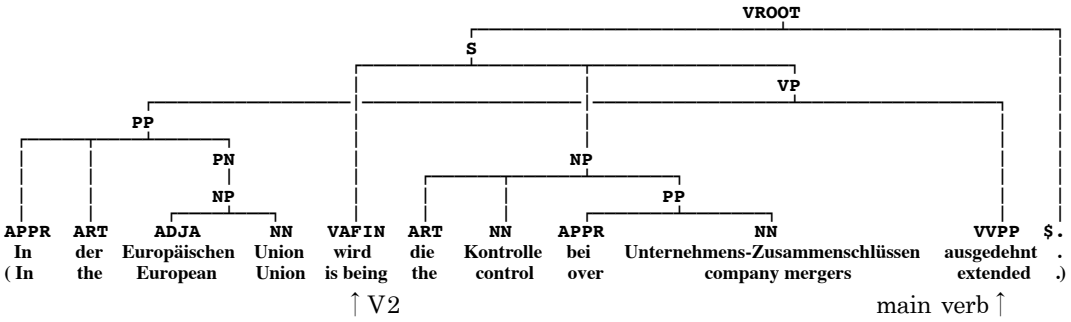


FIGURE 1.2: German is a verb-second word order (V2) language, where the second position from left of the main clause is occupied by the verb phrase. Object and other argument can relatively have free positions with their case markers and inflection as the sign of its grammatical role.

R is a finite set of production rules, each in the form of $A \to g[A_1, \ldots, A_n]$, where $A, A_1, \ldots, A_n \in N$ have an one-to-many relation with $n \geq 0$. g is a composition function $g : (N^\star)^{\varphi(A_1)} \times \cdots \times (N^\star)^{\varphi(A_n)} \to (N^\star)^{\varphi(A)}$, which corresponds to how $\varphi(A)$ spans of A are rearranged among $\sum_i \varphi(A_i)$ spans of children $(A_1, \ldots, A_n)$. Rules may take the form $A \to g[]$, where g returns a string from Σ.

S is a designated start symbol with $S \in N$, which can only appear on the left-hand side of the production rules. (the same definition as S of CFG.)

The application of an LCFRS $G' = (\Sigma, N, R, \varphi, S)$ is further defined for every non-terminal $A \in \Sigma$ and the set of A's derivation $yield(A)$:

- $g() \in yield(A)$ for $(A \to g[]) \in R$;
  (i.e., when rewriting terminal symbols for A, write its derivation directly.)
- $g(\alpha_1, \ldots, \alpha_n) \in yield(A)$ for $(A \to g[A_1, \ldots, A_n]) \in R$ and $\alpha_i \in yield(A_i)$;
  (i.e., when rewriting non-terminal symbols for A, the yields of A must be rearranged by g and redistributed to children $(A_1, \ldots, A_n)$.)
- The function g must be linear and non-erasing, which indicates that the rearrange function g never repeats or reduces strings from input to output.

As already mentioned, the key extension from CFG to LCFRS lies in the rearrange function g. For example, $g(\langle \alpha_1 \alpha_2 \rangle, \langle \beta_1 \rangle) = \langle \alpha_1 \beta_1 \alpha_2 \rangle$ can achieve the injection of $\beta_1 \in yield(B)$ into $\alpha_i \in yield(A)$. Meanwhile, the additional complexity of LCFRS is obviously seen in the different types of $g[A_1, \ldots, A_n]$ associated to each non-terminal A, whereas CFG only has $g[A_1]$ or $g[]$ associated. LCFRS has multiple equivalent mildly context-sensitive grammars for discontinuity in constituency parsing. All of them has similar complexities for the rearrangement. When considering all possible parsing trees, the complexity of LCFRS is $O(n^{3f})$ and $O(n^3)$ for CFG in CNF, where n is the input size (i.e., sentence length and numbers of words), f is the fan-out degree.

Similar to CNF, LCFRS has reduced binary version, which reduces the complexity to polynomial $O(n^6)$. However, unlike CFG, LCFRS is mainly for natural languages and LCFRS parsers struggle more than CFG parsers to reach linear complexity and high parsing speed. We cover recent tree structure constituency parsers in Chapter 2.

Finally, we show an annotated German parsing sample in TIGER Treebank. Unlike English samples in DPTB, German innately have free word order and discontinuous phrases because of its complex case system. For example, modes, voices, or even separable verbs have their particle locating at the end of the main clause.

1.2.2 DAG Structure

A DAG structure embodies a parse when multiply parent phrases share common phrase components. In English, there are both syntactic and semantic grounds for this level of structure.

On the syntactic side, there are two elliptical coordination phenomena called *gapping* and *right node raising* (RNR). Typically, RNR involves but is not limited to the coordination structures. For example, 5a & 5b are gapping and 5c–5e are RNR. We use small bracketed font to denote the ellipses:

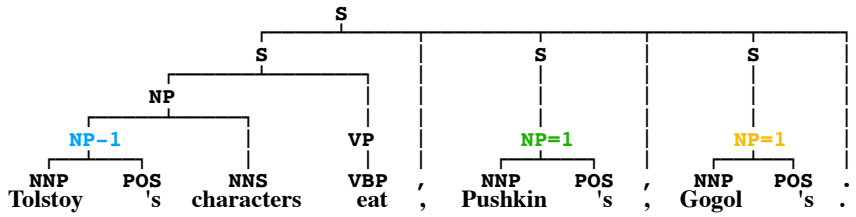
5. (a) *Tolstoy's characters eat, Pushkin's* [characters eat], *Gogol's* [characters eat].
   (b) *The plant was evacuated and workers* [were] *sent home.*
   (c) *I haven't seen* [him] *or talked to him in several years.*
   (d) *He nodded to* [her] *without talking to her.* (without coordination)
   (e) *It was nothing more* [than a feeding frenzy] *or nothing less than a feeding frenzy.*

In English, gapping occurs in the non-initial conjuncts of the coordinations, where ellipses are usually verbs. The words or phrases in the incomplete structures are called *remnants*, such as *Pushkin's* and *Gogol's* in 5a. Meanwhile RNR occurs at the final conjunction of the coordinations, where ellipses can be structures which do not involve with verbs. Both phenomena challenge the theories of syntax in significant ways because the incomplete structure with ellipses are not qualified as phrases. For example, the final coordinated sentence *Gogol's* in Figure 1.3 (a) yields a noun phrase via a strange rule `S→NP` without including any verbs. A more common English grammar for 5a would look like Figure 1.3 (b), where the ellipses are borrowed from the first complete sentence. The result is a DAG.

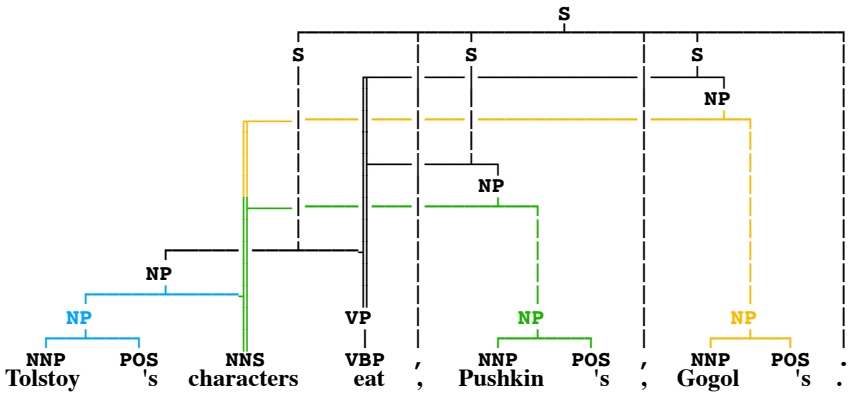
On the semantic side, a verb cluster, where two or more verbs align in adjacent order and function as a whole, also creates DAGs. Each single verb in the clusters requires their own predicate-argument structure and selects the existing phrases taken by the cluster. Not every single verb creates its own predicate-argument structures. Superordinate verbs, such as control and raising, can redirect its arguments to its subordinate verbs. Although the verb clusters are formed by syntax, the redirection of argument is semantically motivated as following:

6. (a) *The wind never seems* [the wind] *to stop.*
   (Raising verb *seem* redirects its subject *the wind* to *stop*.)
   (b) *They want [me] to buy reinsurance.*
   (Raising verb *want* redirects its object *me* to *buy*.)
   (c) *An uninvited guest tried* [the uninvited guest] *to crash the party.*
   (Control verb *try* redirects its subject *an uninvited guest* to *crush*.)
   (d) *You ask [it] to be gone.*
   (Control verb *ask* redirects its object *it* to *be gone*.)

Control predicates semantically select their arguments, whereas raising predicates do not. We do not go further for the difference but highlight these words can



(a) Original PTB annotation with traces for gapping.



(b) DAG after conversion with original traces.

FIGURE 1.3: PTB has traces for gapping remnants (a, noun phrases suffixed with `=1`) for recovering the DAG structure (b). We denote different remnant groups in different colors (i.e., blue, green, and yellow) and shared components with double lines. The initial coordinate in red contains the full structure.

create clusters as a whole and can individually select and share argument from the context to from DAGs. For example, the noun phrase *me* in 6b works as an object for *want* and as a subject for *buy* at the same time.

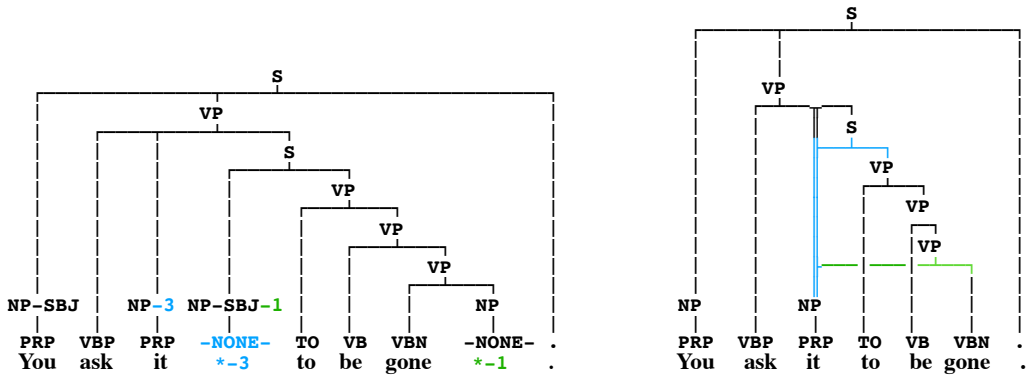
Noticeably, 6b also includes another phenomenon for DAG creation, the passive voice. Beside **it** being the object of *ask* and the subject of *be gone*, passive voice also require the syntactic subject to be the object of *gone*.

By far, We have investigated some causes for the creation of DAGs. However, we do neither cover more the details nor cover a formal grammar for DAG structures (Kamimura and Slutzki, 1979) in this thesis. There are reasons. On one hand, my work focus on grammar-less parsers for the general structures of trees and DAGs. My methods do not rely on complex symbol logic based on grammars. On the other hand, formal grammars for DAG are too complex and are not as widely adopted as CFG and LCFRS grammars.

1.2.3 Headedness and Lexicalization

We see `S`→`NP` as a strange rule because we expect an verb or a verb phrase to be the essential part of a complete sentence.

Headedness is the argument that each phrase has one child as its *head* from which the phrase receives the major grammatical information that the head projects. Among non-terminal symbols, there is a proper subset of symbols, called pre-terminals, each of which directly yields one and only one terminal. We also call them lexical symbols, word classes, lexicons, or *part-of-speech* (PoS). Each symbol may include specific inflectional information of its yield. For example, `VBP` for word *ask* of 6d indicates a



(a) Original PTB annotation with traces and empty nodes (b) DAG conversion

FIGURE 1.4: A PTB annotation with control verb *ask* of 6d. We keep the original annotation of SBJ, which shows *it* works as a different argument (i.e., an A-movement to be a SUBJect). There is a further A-movement because of the semantic requirement for the passive voice. Word *it* works again as an object for verb *gone*.

non-3rd person singular present verb in PTB annotation scheme. Then, headedness makes sense for the rule VP→VBP NP S.

There are debates on the headedness argument. For example,

7. (a) *The old* (a noun phrase without a noun)
   (b) *Right and wrong* (coordination with an ambiguous head resolution)

in 7a, if the noun is not the essential part of the noun phrase, the phrase should be called a determiner phrase (DT) rather than a noun phrase (NP). Meanwhile, in 7b, the coordination should not be call a coordination phrase because each subordinate phrase has clear grammatical function than the coordinating conjunction (CC→*and*) for the superior structure. Apart from the debates, let us move on to some grammars that highly depend on headedness.

Head-driven Phrase Structure Grammar

Head-driven phrase structure grammar (Carl Pollard, 1988, HPSG) is an symbolic feature enrichment for CFG production rules, which specifies and relies on the head information. It does not facilitate non-context-free, like LCFRS extension to CFG.

To be more concrete, each production rule is equipped with a set of grammatical constraints:

$$A \to \alpha$$
$$\{\text{set of constraints for } \{A\} \cup \alpha\}$$

where the set of constraints consists of feature checking for each symbol. For example, the following checks the compatibility for feature of agreement

$$\begin{gathered} \texttt{S} \to \texttt{NP VP} \\ \langle \texttt{NP}_{\text{AGREEMENT}} \rangle = \langle \texttt{VP}_{\text{AGREEMENT}} \rangle \end{gathered} \tag{1.1}$$

and

$$\begin{gathered}\texttt{NP} \rightarrow \texttt{DT NN} \\ \langle \texttt{DTAGREEMENT} \rangle = \langle \texttt{NNAGREEMENT} \rangle \\ \langle \texttt{NPAGREEMENT} \rangle = \langle \texttt{NNAGREEMENT} \rangle\end{gathered}$$

for sentences, such as *we jump* and *he cries*, and noun phrases, such as *a cat* and *those dogs*. The feature names inside $\langle\rangle$ are paths towards a primitive value or a compound feature description structure, called attribute-value matrix (AVM). A flat AVM is a set of primitive feature-value pairs:

$$\begin{bmatrix} \text{FEATURE}_1 & \text{VALUE}_1 \\ & \vdots \\ \text{FEATURE}_n & \text{VALUE}_n \end{bmatrix}$$

An AVM can nest in another AVM under one of its feature path:

$$\begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SINGLE} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \end{bmatrix}$$

If both AVMs of `NP` and `VP` in 1.1 contain the above AMV, then constraint is met. As a result, the two AVMs can safely merge by the unification operation following the CFG rule. Under path $\langle \text{AGREEMENT} \rangle$, a new AVM for `S` forms by unifying `NP`'s and `VP`'s AVMs,

$$\begin{bmatrix} \text{NUMBER} & \text{SINGLE} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} \sqcup \begin{bmatrix} \text{NUMBER} & \text{SINGLE} \\ \text{PERSON} & \text{THIRD} \end{bmatrix} = \begin{bmatrix} \text{NUMBER} & \text{SINGLE} \\ \text{PERSON} & \text{THIRD} \end{bmatrix}$$

where $\sqcup$ is the unification operator. In addition, when two operands AVM contain mutually different features, those feature-value pairs will be kept in the new AVM. Otherwise, the unification, like

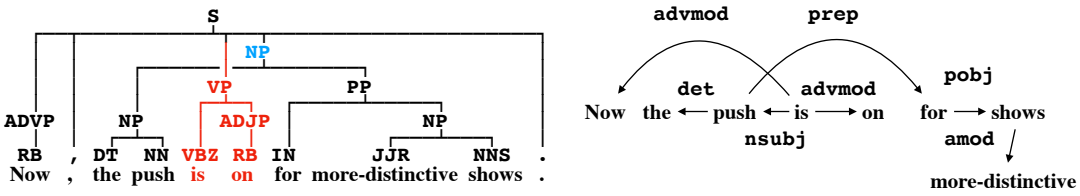
$$\begin{bmatrix} \text{NUMBER} & \text{PLURAL} \\ \text{PERSON} & \text{FIRST} \end{bmatrix} \sqcup \begin{bmatrix} \text{NUMBER} & \text{SINGLE} \\ \text{PERSON} & \text{THIRD} \end{bmatrix}$$

fails and thus the current parsing step fails.

HPSG decomposes features into primitive values and organize them into AVMs so that the constraint checking and feature passing can be become concise, like the above AGREEMENT example. During parsing, the feature information flows from the bottom terminals to upper non-terminals. It naturally requires a HEAD feature that majorly passes from one of the phrase children to the parent, highlighting the impact of headedness and lexical information (Pustejovsky, 1998).

Dependency Parsing: Non-projectivity vs. Discontinuity

Dependency parsing (Tesnière, 1959; Hudson, 2021) is another early parsing formalism that develops in parallel with constituency parsing. In contrast to creating phrasal structure as parsing, dependency parsing directly link words to their arguments, which are binary semantic or syntactic lexical relations between words with



(a) Discontinuous Constituency Parsing. (b) Non-projective Dependency Parsing.

FIGURE 1.5: Given the head information, constituency tree can be converted into dependency graph. Constituency discontinuity and dependency non-projectivity root in the same syntactic structure of this sentence.

no focusing on phrases. (However, there are some labeling tricks to have dependency graph bearing phrasal information.) Predicate-argument relationship plays an important role, as shown in Figure 1.5 (b). Many English verb phrases have head verbs as the predicates in relation to arguments inside or near their phrases. Recent efforts (Zhou and Zhao, 2019) combine head information of dependency and phrase information of constituency into a simplified HPSG parsing and witness mutual benefits on both tasks. Similar to constituency parsing, there are also levels from tree to DAG for dependency parsing (Sagae and Tsujii, 2008; Fancellu et al., 2019).

Given the head information, a constituency parsing tree can also be converted into a corresponding dependency tree. The *projectivity* and *non-projectivity* of dependency tree commonly corresponds to *continuity* and *discontinuity* of constituency tree. When two edges among four words in a dependency tree get crossed the dependency tree becomes *non-projective*. In Figure 1.5 (b), the edge between "*Now*" and "*is*" comes across with the edge between "*push*" and "*for*". In its corresponding Figure 1.5 (a), the discontinuous constituency parsing reflects this non-projectivity as constituency discontinuity. Without considering the creation of phrase structure, earlier dependency parsing approaches has shown their advantages of native support for non-projectivity. For example, McDonald et al. (2005) formulize dependency parsing as finding *maximum spanning trees* (MST) in directed graphs at $O(n^3)$ or lower complexities, pioneering the genre of *graph-based* parsing. Finding global optima is similar to the characteristics of *chart-based* parsing for constituency parsing, which I will introduced in Section 2.2.

Combinatory Categorical Grammar

Along with HPSG that assigns information-rich AVM to each word, combinatory categorical grammar (Steedman, 1997; Steedman, 2004, CCG) assigns a category as a string of constituency labels and operating direction symbols to each terminal. The direction symbols indicate where its relation is about (i.e., to its left or right), somewhat like the function of edges in dependency parsing.

CCG is one of the lexicalized grammars whose compound category interact with each other under symbolic unification rules, such as forward application.

$$\texttt{X} \rightarrow \texttt{X/Y Y}$$
$$\texttt{X} \rightarrow \texttt{Y X\Y}$$

For example, an English intransitive verb receives a category `S\NP` which means it takes an `NP` as its subject on its left side (i.e., "\") to form a sentence `S`. Likewise, a ditransitive verb receives a category `((S\NP)/NP)/NP` with `(.../NP)/NP` indicating

the direct and indirect objects on the right side (i.e., "/"). Derivations look like

$$\texttt{S} \rightarrow \texttt{NP}\ \ \texttt{S\NP} \rightarrow \textit{He smiles}$$

$$\texttt{S} \rightarrow \texttt{NP}\ \ \texttt{S\NP} \rightarrow \textit{I}\ \ \texttt{(S\NP)/NP}\ \ \texttt{NP} \rightarrow \textit{I}\ \ \texttt{((S\NP)/NP)/NP}\ \ \texttt{NP}\ \ \textit{tea} \rightarrow \textit{I offer him tea}$$

Note that the exemplary ditransitive verb `((S\NP)/NP)/NP` as a predicate refers to its three argument, creating a semantic frame. CCG is often in the middle ground between syntactic and semantic parsing, which involves deeper predicate-argument structure (Baldridge and Kruijff, 2002).

Futhermore, CCG has other advanced rules, such as type raising, composition, and decomposition, which allow terminals to have local categories and exchange them in respond to the context. These rules cover and can differentiate long-distance dependencies, such as gapping, raising, control, and right-node raising. Despite that its parsing process is in a context-free style with binary rule application to adjacent categories, those flexible compound symbolic categories allows some discontinuity or non-projectivity to be addressed (Little, 2010).

1.3 Related Task

1.3.1 Chunking: Word Segmentation and Shallow Parsing

The full parse tree is not always necessary, which bring shallow parsing or partial parsing to the topic. Chunking tasks, such as word segmentation and named entity recognition (NER), require the identification of non-overlapping continuous spans in the input plain string.

Word segmentation can be seen as a chunking task. Although word segmentation in English can be relatively easy to tackle with regular expression, it is more demanding for other languages without using explicit delimiters (e.g., white space), such as Chinese and Japanese, as shown in Figure 1.6 (b). Many tasks for those languages start with a word segmentation pre-processing.

In the case of NER, the named entities are scattered continuous spans of words in the sentences, which refer to entities in the real world, such as person names, organizations, locations, etc. Shallow phrase chunks and named entities are not necessarily in the relation of container and content. Some phrase chunks and named entities has both common parts and respective parts. As shown in Figure 1.6 (c), the organization *People's Daily* is split into two `NP`s with one of it containing a determiner *the*.

1.3.2 Structured Semantic Task: Sentiment Analysis

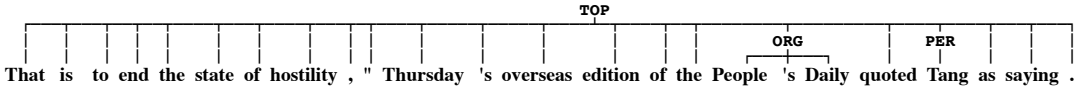
Sentiment analysis is a task to identify or quantify affective states and subjective information from natural languages. There are sentiment corpora of different domains, such social media sentiment (Pak and Paroubek, 2010), product review (Keung et al., 2020), and etc. Among them, Stanford Sentiment Treebank (Socher et al., 2013b, SST) stands out with sentiment organized in binary tree structure, as shown on the left side of Figure 1.7. The sentiment or semantics of a phrase is affected by its child phrases or words.

The tree stuctures in SST are syntactically generated by a constituency parser (Klein and Manning, 2003). Instead of a constituency label, each phrase has a sentiment label, which has five ordered categories ranking from very negative (label 0) to neural (label 2) to very positive (label 4). Socher et al. (2013b) also provide a

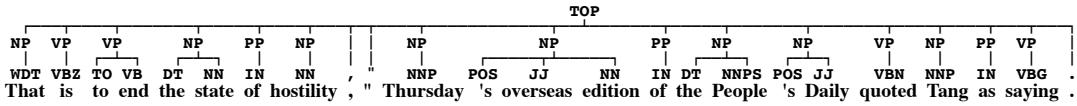
|この|ため|、|唐代|の|祖師たち|は|苦闘|を|重ね|ながら|悟っ|て|き|た|の|で|ある|。|

|狩野|把|日本|代表团|的|团旗|授给|了|团长|、|东京都|残疾人|综合|体育|中心|主任|伴敏彦|。|

(a) Word Segmentation



(b) Named Entity Recognition



(c) Phrase Chunking

FIGURE 1.6: (a) contains samples from Chinese Treebank and Keyaki Treebank. (b) and (c) are from a sample from CoNLL 2003 dataset, which provides phrase chunk and NER annotations in parallel.

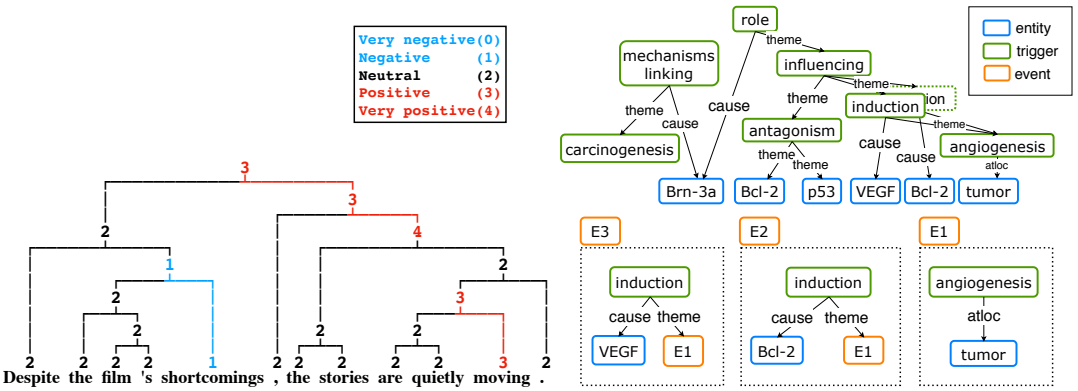


FIGURE 1.7: Two semantic tasks. Left: fine-grained sentiment analysis with Stanford Sentiment Treebank. Right: DAG-based event detection with BioNLP CG shared task 2013 (Pyysalo et al., 2015; Espinosa, Miwa, and Ananiadou, 2019).

model with a binary vector combinator to minitor the sentiment change along the tree branches in a bottom-up style. With the specialized component, the accuracy of the root sentiment is significantly higher than those approaches of flat models (Moore and Barnes, 2021) (except for those with pre-trained language models).

1.3.3 Task for DAG structure

Comparing to tree structure, DAG structure provide more flexible and accurate information, which is typically semantic. For example, DAG dependency parsing captures more complete predicate-argument structure (Sagae and Tsujii, 2008; Tokgöz and Eryigit, 2015; Hershcovich, Abend, and Rappoport, 2017), DAG semantic parsing (Fancellu et al., 2019; Fancellu et al., 2020), and DAG event detection (Pyysalo et al., 2015; Espinosa, Miwa, and Ananiadou, 2019; Espinosa et al., 2022), as shown on the right side of Figure 1.7.

To my knowledge, DAG constituency parsing still lack published researches. One reason might be lacking of large scale constituency graphbanks. However, the trace systems in PTB and CTB introduce DAG structures. Previous works focused on the conversion within tree structure (Ficler and Goldberg, 2016; Evang and Kallmeyer, 2011). In Chapter 6, I specify the conversion of PTB and CTB for DAG with code and manual check.

1.4 Thesis Structure

In the following chapters, we will recap recent state-of-the-art constituency parsers in Chapter 2. My research on parsing family based a neural combinator is specified in Chapter 3 and the experiments follows in Chapter 4. Chapters 3 & 4 also extend our parsers for non-parsing tasks, including structural sentiment analysis, NER, and word segmentation. Also, there are efforts for multilingualism. In Chapter 5, we focus on providing future DAG conversion for PTB and CTB Treebanks.

1.5 Contribution

My main contributions are the following:

- Propose a neural combinator family, neural combinatory constituency parsing (**NCCP**), for continuous and discontinuous tree structure constituency parsing. They include binary and multi-branching parsers in four variants: continuous binary (**CB**), continuous multi-branching (**CM**), discontinuous binary (**DB**), and discontinuous multi-branching (**DM**). They have low empirical complexities and new state-of-the-art parsing speeds with comparable accuracies to recent parsers. **DM** without pre-trained language models possesses the new state-of-the-art accuracy on TIGER Treebank.
- Demonstrate training and inferring methods with roots in ideas of grammars and linguistic phenomena. Unlike parsers that resort to the single tree transformation from normal forms, we embrace all valid tree transformations, including but not limited to Chomsky normal forms. These tricks with tree transformation is effective for my models and can be adopted by other parsers.

Based on the main contribution, I provide extra contributions:

- Combine monolingual **NCCP** models into multilingual parsers which do not exhibit significant performance changes.
- Demonstrate joint tasks with non-constituency structural tasks, including sentiment analysis and word-segmentation.
- Provide DAG conversion from PTB and CTB to graphbanks for future works.

Chapter 2

Constituency Parser

In this chapter, I introduce recent constituency parsers for tree structure. In the recent decade, NLP researches shift from using statistical methods to using artificial neural networks and significantly improve the empirical performance with the constant emergence of large annotated corpora. Neural networks are known to be data-driven and show the significant improvement when data is abundant. Despite the demand for data size, neural approaches are self-adaptive and save expensive labor on manual feature engineering. I focus mainly on fully supervised neural parsers whose experiment are conducted at least on PTB.

As already mentioned, constituency parsing demands hierarchical structures. It requires a model a serial of steps to either achieve a successful parse or recognize a failure. A *transition-based* parser does not explore all the possible parsing structures and may be misled by its previous steps (Dyer et al., 2016; Kitaev and Klein, 2020; Wei, Wu, and Lan, 2020), whereas a *chart-based* (i.e., *tabular-based* or *table-driven*) parser systematically explores all the possibilities and chooses the most plausible one (Kitaev and Klein, 2018; Corro, 2020). When a language can be generated by a formal grammar, both transition-based and chart-based parsers can achieve perfect results. However, natural languages always produce exceptions and the assumptions of grammar generating language may not even stand sometimes. Thus, transition-based and chart-based parsers have their respective relative advantages: a balance between accuracy and complexity. In the following content, a parser by default refers to a fully supervised neural constituency parser.

2.1 Transition-based Parser

I generalize transition-based parsers to be the category that relies on a series of *local* actions to incrementally build a parse. First, the numbers of inputs and actions for classic *finite automata* are not essentially equal and the order of the inputs and actions are sequential. Next, a genre via *sequence labeling* or *supertagging* requires the numbers of inputs and actions to equal. The inputs can be simultaneously mapped into concurrent actions which are applied in some order. Finally, the genre of parsing with *intertive chunking* applies a group of multiple non-conflicting actions in parallel and repeats until the ending requirements are met.

2.1.1 Finite Automata

Typical finite automaton parsers exploit a stack to store unfinished parse fragments. Each action is triggered by their state and input, reminiscent of early ideals, such as the Turing machine, top-down LL(k), and bottom-up LR(k) parsers.

#	Stack	Queue	Action
0	-	over | the | lazy | dog	NT(PP)
1	(PP	over | the | lazy | dog	SHIFT)
2	(PP | over	the | lazy | dog	NT(NP)
3	(PP | over (NP	the | lazy | dog	SHIFT
4	(PP | over (NP | the	lazy | dog	SHIFT
5	(PP | over (NP | the | lazy	dog	SHIFT
6	(PP | over (NP the lazy dog)	-	REDUCE
7	(PP over (NP the lazy dog))	-	REDUCE

TABLE 2.1: An example of the RNNG shift-reduce parsing process.

Recurrent Neural Network Grammars (RNNG). RNNG (Dyer et al., 2016) is a parser-generator pair. The generator is a generative language model based on the state of parsing. Whereas typical language model does not involve parsing structure, RNNG acts as a language model with additional parsing tree generation. The essence lies in how RNNG's parser produce and encode parsing trees.

A continuous constituency tree (w, n) can be uniquely serialized into a sequence of actions $a_i \in a(w, n)$ by a pre-order tree traversal. With input terminal symbols w (i.e., words) in a queue Q, non-terminal symbols n, and an empty stack S, an action is among three types,

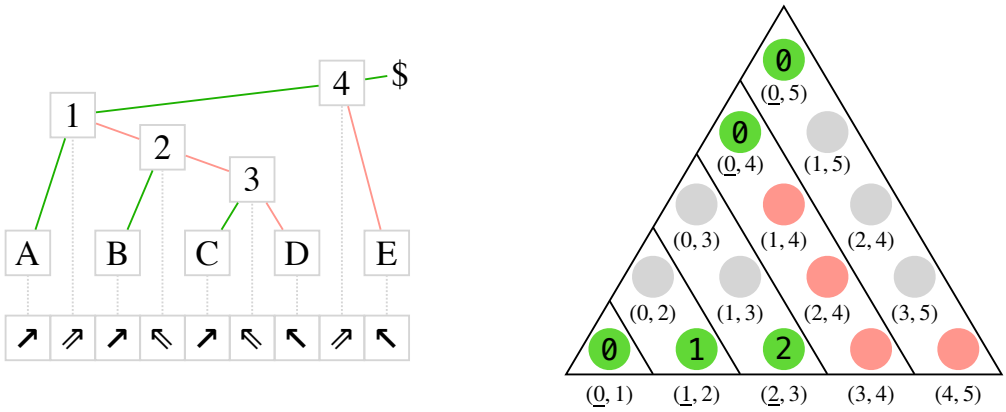
- NT(X) opens a non-terminal X and pushes it to S for the following actions,
- SHIFT shifts a terminal from Q and pushes it to S,
- REDUCE pops the top elements til an open non-terminal, create a subtree with the popped elements, and pushes the new subtree to S.

A parsing process of a multi-branching (or m-ary, $m \in \mathbb{N}$) tree `(PP over (NP the lazy dog))` is demonstrated in Table 2.1. The current action a_t is determined by the parser state consisting of action history embedding $a_{<t}$, stack embedding S_t, and queue embedding Q_t, which are all summarized by stack LSTM, a specialized RNN. RNNG parser terminates parsing when there is only one closed element in S.

Instead of action pairs of NT(X) and REDUCE, Fernández-González and Gómez-Rodríguez (2019) proposed another action REDUCE#m to support flexible multi-branching m-ary trees.

RNNG parsing has certain problems and issues as a finite automaton parser. For instance, actions of NT(X) and REDUCE (or push and pop) may not paired and the number of SHIFT may not equal to the number of words. Furthermore, the distribution of actions can be very uneven for the input words w. For example, a deep tree structure causes multiple NT(X)-REDUCE pairs concentrating on a small number of words. RNNG is for continuous parsing and there are many properties of continuous tree serialization that are not exploited by RNNG. Later on, the parsing genre with sequence-labeling which relies mainly on properties of continuous trees and reduces the structural task into a simpler sequential task (i.e., tree linearization).

To extending finite automaton parsers for discontinuity, there are approaches of adding *swap* or *gap* actions which allow top elements at the stack to be locally reordered. However, for long syntactic dependency, multiple discontinuous actions are necessary for the element to travel inside the stack, increase the parsing complexity.



(a) Orientation of Tetra-tagging. (b) Left child (green), right child (red), and none (gray).

FIGURE 2.1: Binary tree linearizations by (a) Kitaev and Klein (2020) and (b) Wei, Wu, and Lan (2020). Corresponding edges are denoted in the same colors. Numbers in (a) enumerates the build sequence: both are from left to right. Linearized structural actions are at the bottom of (a) and in green circles of (b).

2.1.2 Parsing with Sequence Labeling: Tree Linearization

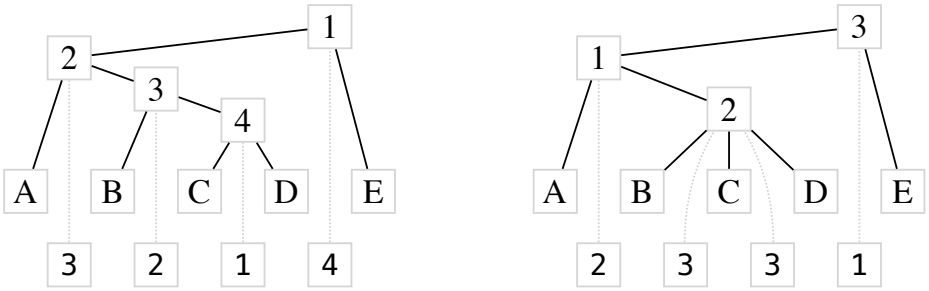
Sequence labeling parsers fix the size of the tree-constructing actions proportional to the size of the input words. Hence, by calculating for each input word, all the necessary actions can be evenly obtained from each input. This genre is diverse, whose mechanisms exploit properties of binary tree (i.e., CNF), such as orientation, left child boundary, and syntactic distance (lowest common ancestor for binary tree). general multi-branching tree properties, such as number of common ancestor and strong incremental of each terminal. I omit constituency labels for the simplicity of the specification.

Orientation. Kitaev and Klein (2020) proposed a linearization method to map an unlabeld binary tree without unary branches of n words into $2n-1$ *orientation* (i.e., they call *direction*) tags. As shown in Figure 2.1 (a), each orientation is either leftward or rightward; except for each terminal, there is orientation at their "space" in between, which we call it interstice. Orientations at interstices are for the binary non-terminals.

Clearly, Tetra-tagging has linear parsing complexity with $2n-1$ actions. When taking the action sequence from left to right, the right-left orientations act like push-pop (or NT(X)-REDUCE of RNNG) pairs. Tetra-tagging is reported having the high continuous parsing speed, despite their incomparable cloud hardware.

Hightlight on left children. Wei, Wu, and Lan (2020) proposed a tree linearization strategy focusing on the left children of a binary tree, as depicted in Figure 2.1 (b). Because each child is on either left or right side for its parent in a binary tree, when slicing all tree nodes correspond to their spans, orientation-like uniqueness appears.

For instance, they chose focusing the left children and fixing the right endpoints of spans. The uniqueness of left child in each span group appears. The top node is treated as a left child to fit in mechanism, like the Tetra-tagging. On the flip side, if one focuses on the right children and fixing the left endpoints of spans, the uniqueness of right child in each span group also appears, when the top is treated as a right child. Based on the position of the uniqueness in each span groups, the binary tree is linearized as a position sequence of length n.



(c) Syntactic distance.
(Height of lowest common ancestor)

(d) Numbers of common ancestors.
(In absolute scale)

FIGURE 2.2: Binary tree linearizations by (c) Shen et al. (2018b) and Multi-branching by (d) Gómez-Rodríguez and Vilares (2018). Numbers of non-terminals enumerates the build sequence: the former (c) divides and conquers by the order of syntactic distance, while the latter (d) follows the default left-to-right order. Linearized structural actions are at the bottom of (c) and (d).

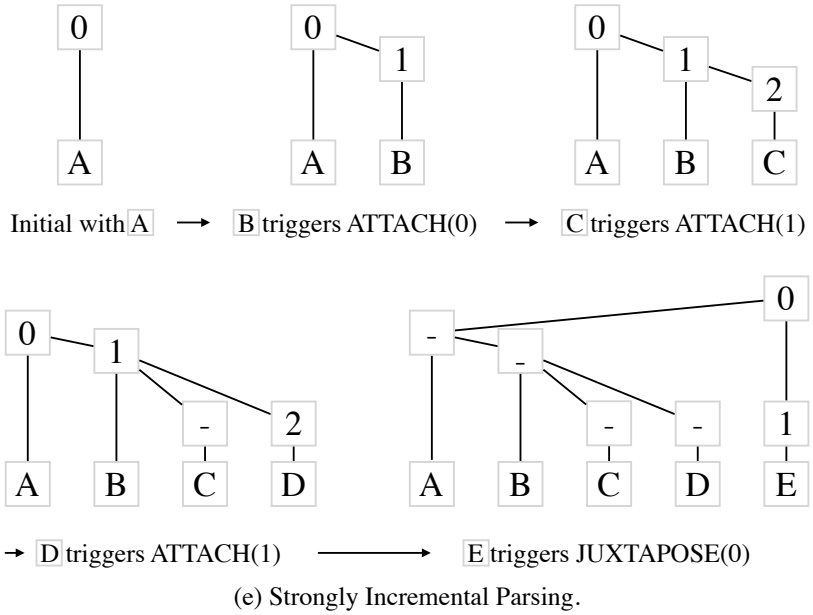
This method is also an example for balance between complexity and speed. Finding the uniqueness of a group is somehow more than a local decision and less than the global optimum for chart-based parsers. Thus, with a complexity of $O(n \log n)$, their accuracy is higher than some of the recent local parsers.

Syntactic distance. The notion of syntactic distance is proposed as Parsing-Reading-Predict Networks (PRPN) by Shen et al. (2018a) for language modeling based on syntactic structure, like RNNG. However, unlike the explicit parsing based on supervision in RNNG, PRPN's parsing is unsupervised and targeted at inducing syntactic information from language modeling. Syntactic distance is the information produced during the process of RNN adapting to predicting tokens, when RNN tries to gain or "forget" information from previous reading via a gating function. The idea becomes more concrete with a specialized RNN by Shen et al. (2019).

Shen et al. (2018b) explicitly refers syntactic distance to height of lowest common ancestor in continuous binary tree, as shown in Figure 2.2. Specifically, instead of regarding the integers as discrete labels, their parser actually regards the order of the integers. This means, the syntactic distances are predicted as continuous values that follow the order in the tree linearization sequence. The actual value of each output is not importance, while the relationship is crucial.

The building sequence of Shen et al. (2018b) is also different from previously mentioned (a) and (b). It greedily selects the highest syntactic distance, breaks the string into two parts, and iterates the process in a swift divide-and-conquer strategy of complexity $O(n \log n)$. Although a method by Stern, Andreas, and Klein (2017) does not model syntactic distance, their models include a similar top-down divide-and-conquer splitting strategy based on span score. This strategy is also observed in a genre, easy-first parsing, mainly for dependency parsing (Goldberg and Elhadad, 2010; Nivre, Kuhlmann, and Hall, 2009; Versley, 2014).

Lowest common ancestor. In constrast, Gómez-Rodríguez and Vilares (2018) focused on modeling the discrete height and common ancestor information. Because height is an unbounded discrete value, controlling it within a range becomes necessary. Thus, they also proposed a relative scale by the difference of adjacent heights



(e) Strongly Incremental Parsing.

FIGURE 2.3: Multi-branching tree linearization by (e) Yang and Deng (2020). Numbers enumerates the possible positions in *rightmost chain* where actions can take place.

and adopted for the experiment. Their complexity is linear and has a parsing speed on PTB closely behind my parser.

Methods of (a), (b), and (c) still stay only for continuous parsing, probably because they highly rely on properties dedicated to continuity. However, Vilares and Gómez-Rodríguez (2020) extended (d) for discontinuous parsing by adding Lehmer code to encode the sparse discontinuity. Lehmer code for each terminal stays at zero in the continuous case, while it becomes positive for local permutation for discontinuity.

Strong incremental. Incremental parsing is a style rather than a concrete method. In contrast to having tree fragments in stacks of finite automaton parsers at mediate steps, incremental parsers, at any steps, always have one and only one well-formed tree under construction. Yang and Deng (2020) further hightlights the notion of *strongly incremental* that each input triggers one and only one action, which, in their system, ATTACH or JUXTAPOSE the input to the working tree. This method strongly belongs to tree linearization with input and action proportion one-to-one, which is slightly reminiscent of tree-adjoining grammar (Joshi, 1983, TAG). (However, mildly context-sensitive TAG may assign a terminal with a tree fraction.) Specifically, they indicated *rightmost chain* to the chain of nodes starting from the root and iteratively descending to the rightmost child, where actions can take place. Each action takes an integer argument i referring to a position in *rightmost chain* and has two types,

- ATTACH(i) attaches the current terminal to the i-th node in the chain and inserts a new node for further possible actions.
- JUXTAPOSE(i) juxtaposes the current terminal with the i-th node in the chain, insert two new nodes – one for the terminal and the other for the new common parent of the two siblings.

I show how the model parses multi-branching trees (d) with some empty dummy nodes in Figure 2.3. Each terminal triggers an action to build the only continuous

Level #	Chunk Tags								
4	B_S	I_S	I_S	I_S	I_S	I_S	I_S	I_S	E_S
3	O	O	O	O	B_VP	I_VP	I_VP	I_VP	E_VP
2	O	O	O	O	O	B_PP	I_PP	I_PP	E_PP
1	B_NP	I_NP	I_NP	E_NP	O	O	B_NP	I_NP	E_NP
0	A	quick	brown	fox	jumps	over	the	lazy	dog

TABLE 2.2: Chunker-based parsing with BIOE prefix (i.e., for beginning, inside, outside, and ending).

tree with the growing margin limited to the *rightmost chain*, which resembles the span groups by their right endpoints of (b). They share a complexity of $O(n \log n)$. Using empty dummy nodes for the tree shape is not a serious problem because all models in this subsection use mediate dummy nodes for unary and binary branches.

2.1.3 Parsing with Iterative Chunking

A chunker breaks a continuous sequence into multiple consecutive spans. From the opposite viewpoint, it connects consective units into larger spans. Collobert (2011) iteratively exploits a chunker and turns it into bottom-up combinatory parser, as demonstrated in Table 2.2.

Collobert (2011) adopts neural networks like previously introduced local parsers. The parser supports multi-branching trees. However, it has a certain drawback: the complexity is bounded by $O(n^2)$ instead of $O(n \log n)$ in the case of left or right recursive trees of height $n-1$. The culprit is that the chunker lacks a compose function to reduce multiple children into a new terminal node. The forth level in Table 2.2 could have only two nodes of (B_S, E_S) for (NP, VP), respetively, if a compose function were introduced.

On the flip side, the pioneering Ratnaparkhi (1997) adopts a statistic approach with a compose function. Although the complexity is still bounded by $O(n^2)$ in the left or right recursive case, there is fewer nodes to calculate and the empirical complexity stays linear. But neither of them does not support discontinuity.

2.1.4 Summary

I covered three types of local neural constituency parsers, including finite automata, sequence-labeling, and chunker-based. Besides their common locality, they share another profound trait: less grammar. They focus more on the general structural side with constituency label handled by a simple neural classifier based on its location in the structure. However, this trait does little or no harm to the performance with adaptive neural approaches. We will see this trait extending to recent chart-based neural parsers.

All parsers support multi-branching trees, but most parsers managed indirectly via binary tree with dummy nodes and some parsers directly support multi-branching. This is probably because supporting flexible m-arity demands a type of an unbounded number of actions, like REDUCE#m (Fernández-González and Gómez-Rodríguez, 2019). However, Fernández-González and Gómez-Rodríguez (2019) demonstrated that multi-branching parsing can have some advantages, at least some speed-up.

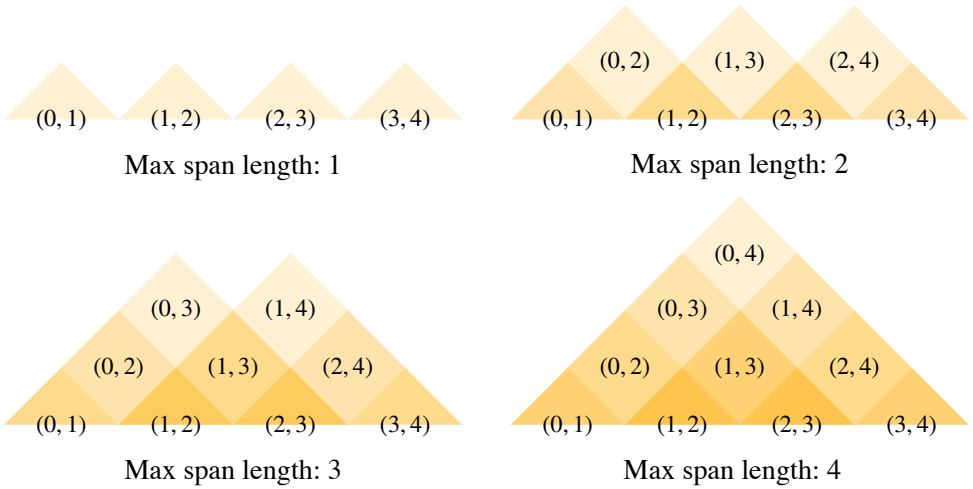


FIGURE 2.4: Concurrent bottom-up steps of CKY algorithm by span length in a chart. This exemplary chart has maximum span length four for four terminals.

In terms of constituent discontinuity, some approaches is extended for the challenging reign of higher complexity. Transition-based are the majority for the extension, while sequence-labeling is not only possible but feasible. To my knowledge, Chunker-based parsing is native in continuous parsing with no research for discontinuous parsing.

2.2 Chart-based Parser

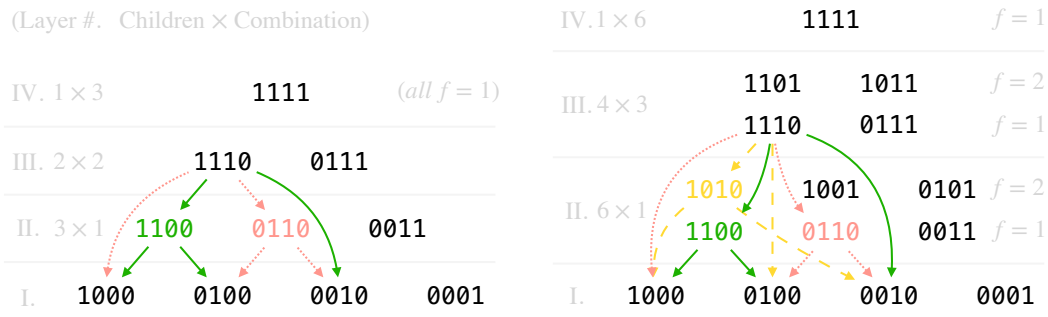
2.2.1 Binary Chart Parser: from CKY Algorithm

CKY Algorithm. CKY algorithm is first discovered by Sakai (1961) and named after rediscoverers of Cocke (1969), Kasami (1966), and Younger (1967). It requires input to be a binary tree, typically in CNF, and work with bottom-up dynamic programming to find the optimal structure, as shown in Figure 2.4.

Each triangle represents a span denoted by the 2-tuple. Larger spans (i.e., larger triangles) contain smaller spans (i.e., smaller triangles), which creates the constituent hierarchy. At first, the optimal solution lies in each individual spans and the calculation is trivial. At the second step, spans of length 2 depends on two child spans it contains. the optimal solution is the sole combination of the two children's optimal solutions (e.g., $(0,2)$ from $\{(0,1),(1,2)\}$). From the third step, more combinations appear (e.g., $(0,3)$ from $\{(0,1),(1,3)\}$ or $\{(0,2),(2,3)\}$) and the search complexity for optimal combination of the top span grow in cubic time.

Formally, for n terminals, spans of length l share combinatorics of complexity $l-1$ because the left span and the right span are continuous and non-overlapping. There are $n-l+1$ concurrent spans at l. Consequently, the binary continuous chart has a fixed $\Sigma_{l=1}^{n}(n-l+1)\cdot(l-1) \sim O(n^3)$ complexity for CKY decoding algorithm.

Neural continuous binary chart. Stern, Andreas, and Klein (2017) and Kitaev and Klein (2018) proposed models with neural components trained for providing scores for spans and labels for CKY decoding process. Their neural backends are RNN and BERT, respectively. They outperformed previous neural architectures with CKY decoding (Durrett and Klein, 2015). Instead of introducing grammatical production rules, contextualization of terminal nodes with RNN and BERT plays a key role. The



(f) Continuous Binary Chart (g) Discontinuous Binary Chart

FIGURE 2.5: Instead of description from the respective of spans, the combinatorics of both continuous and discontinuous are shown as tuples of bits representing the existence of terminal symbols. Different constituency dependencies are shown using different line types and colors.

trend of less grammar is followed recent neural models with CKY algorithm (Zhou and Zhao, 2019; Zhang, Zhou, and Li, 2020; Mrini et al., 2020).

Neural discontinuous binary chart. In the same vein for discontinuous parsing, Corro (2020) proposed a chart parser family based on discontinuous spans for binary LCFRS garmmars. The unrestricted member has $O(n^6)$ complexity and supports the full grammar type coverage and the very restricted member has $O(n^3)$ complexity and support a few rudiment discontinuous grammar types, which are adapted from continuous grammar rules via label tricks. Although the $O(n^3)$ member does not cover many discontinuous grammar types, the it empirically covers the continuous and rudiment discontinuous grammars in great number (98% of constituents observed in linguistic treebanks). The $O(n^3)$ member also achieves the best overall performance thanks to the empirical facts. However, their discontinuous accuracies are lower than other recent parsers.

I show an instance for how discontinuous binary chart has higher complexity, as depicted in Figure 2.5. In (f) continuous binary chart, node `1110` has two types of binary combinations, namely $\{$`1000,0110`$\}$ and $\{$`1100,0010`$\}$. Node `1110` is one of two children in the third layer (with maximum span length 3). Meanwhile, in (g) discontinuous binary chart, node `1110` has one more discontinuous combination, $\{$`1010,0100`$\}$. It is also one of four children in the third layer. The numbers of both children and its combination grows, generally resulting in total complexity grow. As mentioned in 1, the complexity grows exponentially with the fan-out degree f. All observed discontinuous chart-based parsers limit $f \leq 2$ with binary combination (Corro, 2020; Stanojevic and Steedman, 2020).

2.2.2 N-ary Chart-based Parser: from Earley Algorithm

Earley algorithm. Earley algorithm (Earley, 1970) is a flexible top-down left-corner algorithm with dynamic program searching for optimum. Unlike CKY algorithm with fixed complexity, Earley algorithm's complexity starts from linear time to cubic time, depending on the grammar type (i.e., respectively in the cases of deterministic context-free grammars, unambiguous context-free grammars, and general context-free grammars).

Term *top-down* means that it starts with the top non-terminal on the left-hand side of the production rules; and term *left-corner* indicates that it then first "processes"

symbols on the right-hand side of the rules with a left-to-right progress marked with "•". To "process" means to `predict`, to `scan`, and finally to `complete` each rule. All "processes" are organized in a chart of $n+1$ levels (n for number of terminals). Each step contains a flexible numbers of "processes". Some rules at level i may be "processed" and allowed to enter level $i+1$ and to trigger `predict` for its right-hand side symbols; some rules will `complete` as being accepted or discarded. Level 1 starts with a designated start rule and each level $i > 1$ starts with the sole `scan` to take in a terminal symbol.

For example, given a phrase *at home* and rules as the following,

- `PP` → `IN NP` (start rule)
- `NP` → `NN | NN NN`
- `IN` → *at* | *with*
- `NN` → *home* | *work*

the inference with the Earley algorithm's chart of length 2 operates as the following,

1. • *at home*
   (a) `Predict: PP` → • `IN NP` (start rule)
   (b) `Predict: IN` → • *at* (from 1a)
   (c) `Predict: IN` → • *with* (from 1a)
2. *at* • *home*
   (a) `Scan: IN` → *at* •
   (b) `Predict: PP` → `IN` • `NP` (from 1a and 2a)
   (c) `Predict: NP` → • `NN` (from 2b)
   (d) `Predict: NP` → • `NN NN` (from 2b)
   (e) `Predict: NN` → • *home* (from 2c and 2d)
   (f) `Predict: NN` → • *work* (from 2c and 2d)
3. *at home* •
   (a) `Scan: NN` → *home* •
   (b) `Complete:` `NP` → `NN` • (from 2c and 3a)
   (c) `Complete:` `PP` → `IN PP` • (from 2b and 3b, final accept)
   (d) `Complete:` `NP` → `NN` • `NN` (from 2d and 3a, discard)

At level 1, the start rule 1a triggers 1b and 1c, which all `predicts` will be examined in later levels. Level 2 `scans` a terminal and processes the previous `predicts`. Finally, level 3 `scans` the final terminal and screens one `complete` rule 3c with a dynamic programming backtrack retrieving the full tree derivation (i.e., 3c, 3a, and 2a).

Earley algorithm shares some common points with finite automaton parsing with strongly incremental tree linearization. 1) Actions. Earley algorithm have `predict`, `scan`, and `complete` organized in $n+1$ levels, whereas a strongly incremental finite automaton parser has a sequence of actions of length n. 2) States. A state for each rule in Earley algorithm determines the effect of `scan` and the result of `complete`, whereas a state of a finite automaton parser determines what action(s) it can take. 3) They support general (or multi-branching) continuous tree structure. The sheer difference is that Earley algorithm is global with all possible parsing derivations included at each level. 4) They create left-to-right dependency that do not allow full concurrency as CKY algorithm.

Recursive Semi-Markov Model. Concurrency is one main reason for the prevalence of neural network. The fact that Earley algorithm relies on top-down symbolic deduction may also make it hard for neural concurrency. Based on the idea of concurrency for CKY algorithm, Xin, Li, and Tan (2021) brought continuous multi-branching chart-based parsing with a CKY-like recursive Semi-Markov model. For instance, node `111` can be any one of combinations {`100,010,001`}, {`100,011`}, and {`110,001`} in their chart. As a continuous constituency parser, their complexity is $O(n^4)$ with a slight advantage of multi-branching accuracy. Like Fernández-González and Gómez-Rodríguez (2019), they suggest that supporting native multi-branching bring some advantages.

2.3 Joint and Unsupervised Task

2.3.1 Joint Constituency and Dependency Parsing

As the initial attempt to formulate a simplified HPSG (Zhou and Zhao, 2019) that integrates constituent and dependency representations into head-driven phrase structure, they merge the functionality of the CKY-based parser (Kitaev and Klein, 2018) and a chart-based dependency parser (Dozat and Manning, 2017) for joint training. The work includes two variant parsers, one encode head child position in constituent span with prefix and the other explicitly locate the head child. Although the full HPSG is much more informative than their formalism with only indication of head, their result shows that both constituency and dependency parsing are compatible and offer extra gain for each other. Mrini et al. (2020) demonstrate separately trained models for either parsing formalism. Zhang, Zhou, and Li (2020) borrow dependency parsing model architecture for continuous constituency parsing. Discontinuous constituency parsing also take a similar approach and show significant gain from additional head information. (Fernández-González and Gómez-Rodríguez, 2022).

2.3.2 Unsupervised Constituency Parsing

Unsupervised constituency parsing is a task to find phrase structure from plain languages without manual annotation as modeling supervision. The task is also known as grammar induction (or tree induction in the case of unlabeled tree). This is also an active research topic for neural models.

As introduced in previous subsections, RNNG is a pair of supervised parser and generative language model relying on parsing actions. By adding a CRF parser which produces a distribution over binary trees to train RNNG, Kim et al. (2019) extended RNNG into unsupervised RNNG (URNNG) for tree induction. The CRF is aimed to maximize the evidence lower bound for the tree structure and RNNG receives random trees for the identical input sentence and learn from them as supervision. PRPN (Shen et al., 2018a, DIORA) can directly infer binary trees with syntactic distance. Finally, based on embedding, Drozdov et al. (2019) obtains labeled tree structure by leveraging inside-outside algorithm (Baker, 1979) to repeatedly encode a sentence into tree nodes and recover the sentence from the tree nodes. Htut, Cho, and Bowman (2018) and Li et al. (2020) compared their performances. While DIORA exceeds in many metrics, there still exists a huge gap between unsupervised and supervised constituency parsers.

Chapter 3

Neural Combinatory Constituency Parsing

This chapter specifies neural combinator-based constituency parsers for *{continuous, discontinuous}* contituency parsing in *{binary, multi-branching}* styles. The combination of properties produces four models: **CB**, **CM**, **DB**, and **DM**, where the binary are special cases of the multi-branching models. The binary are specialized in some tasks (e.g., sentiment analysis) which provide only samples in binary tree format and they show interesting phenomena during the experiments.

All four **NCCP** models contain a common recursive layer-wise loop of 1) concurrent yes-or-no structural actions, 2) vector composition functions based on 1), and 3) multi-class prediction for tag or label based on vectors of 2) with no grammatical constraints. Specifically, each layer is called a *ply* consisting of a sequence of nodes. Initially, each node is an input word represented as an embedding vector. During parsing, the ply nodes become partial derivations or subtrees with their roots still arranged as a sequence and represented as embedding vectors. When there is a single node in the ply, the parse terminates with a parse tree.

NCCP is transition-based. Similar to a finite automaton parser, each the state of which leads to an action, the ply is the state to **NCCP**, each of which leads to concurrent actions to modify the ply itself. Also, similar to a chunker-based parser, **NCCP** iterates with concurrent actions. My implementation utilizes neural recurrent components to make those greedy actions for bottom-up tree construction.

3.1 Continuous Ply

Two continuous parsers are base models for discontinuous parsers. I call continuous binary parser **CB** and continuous multi-branching parser **CM**. **CB**'s binary structural action is called *orientation* and **CM**'s binary structural action is called *chunk*. I denote,

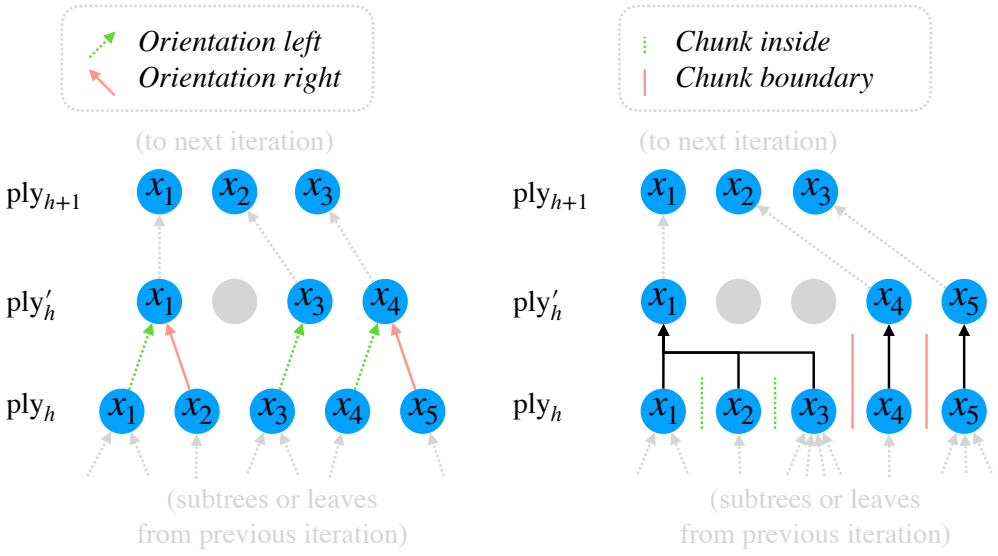
$$orientation(x_i) \in \{0, 1\}, \tag{3.1}$$

where 0 is for *orientation left* and 1 for *orientation right*, and

$$chunk(x_i \oplus x_{i+1}) \in \{0, 1\}, \tag{3.2}$$

where 0 is for *chunk boundary* and 1 for *chunk inside* and $\oplus$ is the concatenation operator just for two adjacent nodes. For **CB**, *orientation* action is designated for each node n_i in the ply. For **CM**, *chunk* action is for each interstice between n_i and n_{i+1}.

Both models iteratively work on a $ply = (x_1, \ldots, x_n)$, each node of which represents a leaf or a subtree. Their bottom-up parsing process with ply height is illustrated in Figure 3.1. The current snapshot ply_h containing $(x_1, \ldots, x_5)$ nodes can be



(a) Binary ply (b) Multi-branching ply

FIGURE 3.1: Two adjacent continuous plies at heights h and $h+1$. Ply$_h$ is the current layer of subtrees and their actions (i.e., **CB**'s *orientation*s or **CM**'s *chunk*s) result in an intermediate ply'_h with probably inconsecutive node indices. We reenumerate them into a new ply_{h+1} for the next actions. Empty nodes are placeholders marked by gray circles.

leaf termials if $h = 0$ or non-terminal if $h > 0$ (tree height $h \in [0, H)$). The combination of nodes is conducted by the guidance of respective structural actions and results in an intermediate ply'_h. We rearrange the nodes of inconsecutive indices into ordered ones of ply_{h+1} by

$$\texttt{CONDENSE} : (n \; ply \text{ nodes}) \rightarrow (x_1, \ldots, x_n),$$

so that a new ply_{h+1} is ready for the next iteration (i.e., $\texttt{CONDENSE}(ply'_h) = ply_{h+1}$). The `CONDENSE` function is a simple linear and regular function that never changes the order of, nor adds, nor drops non-empty nodes.

Similar to recent grammarless neural parsers, I assume that all constituency category information be encoded in each node x_i. The implementation details will be covered in Section 3.4.

3.1.1 CB: Orientation

Adjacent agreeing orientations in **CB** lead two node to combine into a new node. Specifically, if

$$orientation(x_i) - orientation(x_{i+1}) = 1, \tag{3.3}$$

then

$$compose : (x_i, x_{i+1}) \rightarrow x_i, \tag{3.4}$$

where *compose* is an abstract binary combinator for vectors, which will be implemented in Section 3.4. Otherwise, nodes stay intact regardless of its orientation,

$$x_i \rightarrow x_i. \tag{3.5}$$

Unlike the orientation system of tree linearization by Kitaev and Klein (2020), **CB** does not have the actions for interstices. Although this difference brings more

theoretical complexity to **CB**, it makes **CB** more easy to be extended for the following discontinuous parsing. Moreover, **CB**'s observed or empirical complexity is bounded by the theoretical complexity. Low empirical complexity leads to higher parsing speed regardless of the theoretical complexity.

Complexity. Clearly, the theoretical complexity of **CB** is $O(n^2)$ by worse case that the binary tree is purely left recursive or right recursive (i.e., linguistically left-branching or right-branching) where each ply_h has only one pair of agreeing orientation pair and thus ply_h has length of $n-h$. It takes $n-1$ plies for complete a tree with the total number of nodes across all plies as the complexity of **CB**:

$$\sum_{h=0}^{n-1}(n-h) \sim O(n^2).$$

3.1.2 CM: Chunking

In the case of **CM**, each closest *chunk boundary* pair $(lb,\ rb)$ groups multiple nodes in-between $(x_{lb+1},\ \ldots,\ x_{rb})$ into a new node x_{lb+1}. Specifically, for $i \in [lb,\ rb]$, if

$$chunk(x_i \oplus x_{i+1}) = \mathbb{1}(i \in (lb,\ rb)), \tag{3.6}$$

then

$$compose : (x_{lb+1}, \ldots, x_{rb}) \to x_{lb+1}, \tag{3.7}$$

where $\mathbb{1}(\cdot)$ is the indicator function and *compose* is an abstract flexible m-ary combinator for vectors with $m > 0$. The indicator function returns 1 if the statement inside is true; otherwise, it returns 0. Function *compose* will be implemented in Section 3.4.

In terms of iterative chunking for constituency parsing, **CM** is more close to Ratnaparkhi (1997) which combines phrase children to a parent instead of Collobert (2011) without combining. Anyhow, all chunker-based parsers share the common native support for multi-branching.

Complexity. The theoretical and empirical complexities of **CM** are bounded by those of **CB**. This is because **CM**'s *compose* function groups equal or more nodes than **CB**'s *compose*. Thus, there must be equal or less nodes in all **CM**'s plies. The theoretical complexity of **CM** is $O(n^2)$.

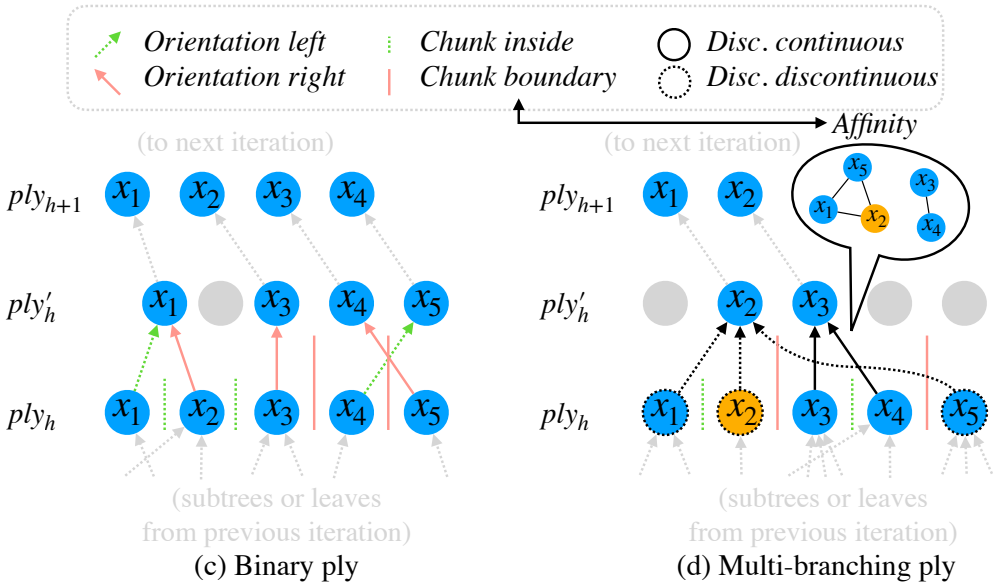
3.2 Discontinuous Ply

DB refers to discontinuous binary parser and **DM** to discontinuous multi-branching parser. They are still ply-centered with iteration based on continuous ones. **DB** and **DM** inherit all functionalities from **CB** and **CM**. As extension for discontinuity, I equip **DB** with a swap-joint system and **DM** with a biaffine attention mechanism.

3.2.1 DB: Swap and Joint

For **DB**, we extend the agreeing orientation composition with a swap-joint switch

$$\begin{aligned} &action(x_i \oplus x_{i+1}) \in \{joint,\ swap\} \\ &joint : compose(x_i,\ x_{i+1}) \to x_i \\ &swap : (x_i,\ x_{i+1}) \to (x_{i+1},\ x_i)\,. \end{aligned} \tag{3.8}$$



(c) Binary ply (d) Multi-branching ply

FIGURE 3.2: Two adjacent discontinuous plies. In addition to continuous plies, **DB** inherits the *chunk* for the swap-joint system and **DM** gets *biaffine attention* action graph described in a matrix.

One *joint* reduces the ply length by one and the *swap* does not affect the length but affects its order. The function *compose* is a binary combinator inherited from **CB**.

Without the agreeing orientation condition, concurrent adjacent actions would conflict in a ply (e.g., two *swap*s for (x_1, x_2, x_3) leaves an undecidable x_2.) Thus, **DB** activates *joint* and *swap* if

$$orientation(x_i) - orientation(x_{i+1}) = 1\,, \tag{3.3}$$

then

$$action(x_i \oplus x_{i+1})\,. \tag{3.8}$$

Otherwise, nodes stay intact regardless of its orientation as in Formula 3.5.

As exemplified in Figure 3.2 (c), (x_1, x_2) and (x_4, x_5) meet the condition in Formula 3.3. Then, *joint* and *swap* are respectively triggered for (x_1, x_2) and (x_4, x_5) because of $action(x_0 \oplus x_1) = joint$ and $action(x_3 \oplus x_4) = swap$. Node x_2 is relayed with neither *joint* or *swap*.

Complexity. In extreme cases, **DB** takes $\frac{n}{2}$ fully swapping plies and $\frac{n}{2}$ fully joining plies. Each ply costs $O(n)$ computation resulting in the upper bound $O(n^2)$.

3.2.2 DB: Medoid and Affinity Biaffine Attention

For **DM**, I first characterize whether x_i and x_j from a ply are two siblings of the same parent constituent as

$$affinity(x_i, x_j) \in \{0,\ 1\}\,, \tag{3.9}$$

where 0 for false and 1 for true. Thus, **DM** decides a *discontinuity* action of x_i and then forwards it to a group action for either a *discontinuous* or a *continuous* constituent as in Formula 3.10:

$$
\begin{aligned}
&\mathit{action}(x_i) \in \{\mathit{discontinuous},\ \mathit{continuous}\} \\
&\mathit{discontinuous}: \\
&\qquad \mathcal{G} = \{x_j \mid \mathit{affinity}(x_i,\ x_j) = 1\} \\
&\qquad\qquad \mathit{medoid} \in \{j \mid x_j \in \mathcal{G}\} \\
&\qquad\qquad \mathit{compose}(\mathcal{G}) \to x_{\mathit{medoid}} \\
&\quad \mathit{continuous}: \\
&\mathcal{G} = \{x_j \mid lb < i \leq rb,\ lb < j \leq rb, \text{ and } \mathit{affinity}(x_j,\ x_{j+1}) = \mathbb{1}(j \notin \{lb,\ rb\})\} \\
&\qquad\qquad \mathit{compose}(\mathcal{G}) \to x_{lb+1}\,.
\end{aligned}
\tag{3.10}
$$

We select one *medoid* for each *discontinuous* constituent to determine its position in the modified ply, whereas the choice of *medoid* for *continuous* constituents makes no difference. *Continuous* nodes split into segments of $(x_{lb+1},\ \cdots,\ x_{rb})$ with $(lb,\ lb+1)$ and $(rb,\ rb+1)$ as boundaries as **CM**. Function *compose* is a flexible m-ary neural combinator inherited from **CM**.

Dozat and Manning (2017) characterized each dependency tree as a sparse asymmetric matrix via biaffine attention, with each sole positive signal in a row (or column) indicating a lexical dependency (from a word to its head or vice versa). Nevertheless, lexical dependency is not available for constituency parsing, and biaffine attention becomes expensive at $O(n^2)$ complexity.

In contrast, we designate *discontinuous affinity* as a small dense symmetric biaffine attention matrix and control its computational size of $O(n^2)$. Otherwise, *continuous affinity* for adjacent nodes takes a special form of

$$\mathit{chunk}(x_i \oplus x_{i+1}) = \mathit{affinity}(x_i, x_{i+1})$$

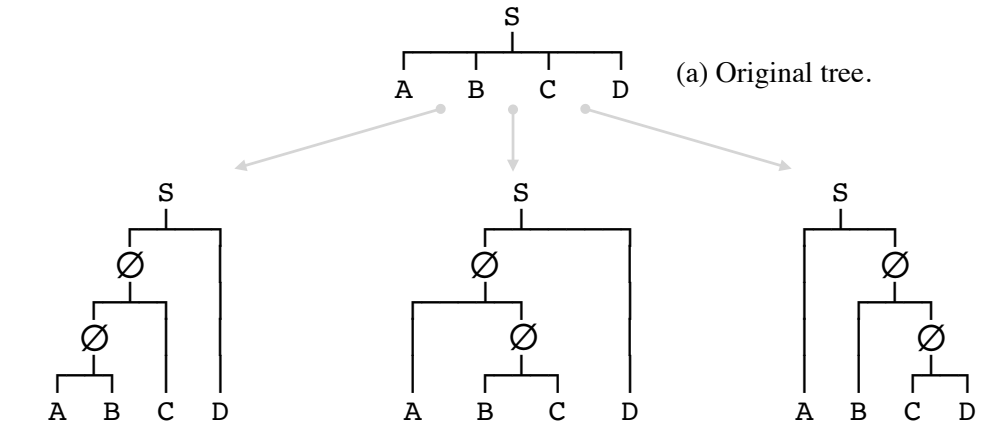
with a simpler $O(n)$ complexity.

Via fast chunking or a small biaffine attention matrix, **DM** balances for its efficiency. As exemplified in Figure 3.2 (d), *discontinuous* (x_1, x_2, x_5) are grouped as one because of their mutual *affinity*, which is equivalent to a 3×3 biaffine attention matrix of ones. Node x_2 is selected as *medoid* for the constituent's location in the new ply. Meanwhile, *continuous* (x_3, x_4) forms a constituent for $\mathit{chunk}(x_i \oplus x_{i+1}) = \mathit{affinity}(x_i,\ x_{i+1}) = \mathbb{1}(i \notin \{2, 4\})$ and $i \in [2, 4]$.

Complexity. In extreme cases, every **DM** ply involves a matrix of *affinity* for all nodes and decreases ply length n only by one. Assume that we can handle decomposing *affinity* matrix within $O(n^2)$. Each ply costs $O(n^2)$ with the upper bound $O(n^3)$.

3.3 Data and Augmentation

I state the conversion from tree into layers of action signals for fully supervised training. In convenience, I merge **CM** and **DM**'s *chunk* semantics into *joint* to unify the interstice signal. Before introducing oracle, tree preprocessing is necessary, as specified in the following three subsections.



(b) $\rho^{\mathrm{B}} = 0$ binarization (c) $\rho^{\mathrm{B}} = 0.5$ binarization (d) $\rho^{\mathrm{B}} = 1$ binarization

FIGURE 3.3: Tree binarization with numerical $\rho^{\mathbf{B}}$.

3.3.1 Empty node and unary branch

Similar to a range of previous works, such as Chen et al. (2021), Shen et al. (2018b), Kitaev and Klein (2020), and Corro (2020), we adopt an empty label "∅" for substructure (e.g., binarization). Besides, we collapse each unary branch into a single node and join their constituent labels regarding their hierarchical order (e.g., `S+VP` as derivation `S→VP`) with easy restoration during inference.

3.3.2 Binarization for CB and DB

Many previous works (Shen et al., 2018b; Kitaev and Klein, 2020; Corro, 2020) binarize tree with one factor (commonly using either CNF left or right) for training. However, neural models are known to be hungry for training data and even pseudo corpus yield by themselves leads to better accuracy performance. In the case of machine translation (Sennrich, Haddow, and Birch, 2016a), pseudo corpus also helps reduce the error propagation (i.e., exposure bias).

Iterating with ply as state, **NCCP** works in a similar way to neural machine translation models. For more variation in the training set which brings higher accuracy, I propose to extend the two CNF factors into a continuous numeric factor. I do not leverage extra head information for binarization because such information is not included in other parsers in comparison and its binarization produce only one static variation, which does not significantly augment the training set.

Specifically, C children of a constituent join one by one in **CB** and **DB** via their *orientation* and *joint* signals. For $c \in [1, C)$, $[1, c]$-th children are set to *orientation right* (1) and $(c, C]$-th are *left* (0). Neighboring children have positive *joint* signals if they are siblings. Otherwise, negative *joints* swap them toward their siblings. I normalize a factor

$$\rho^{\mathbf{B}} = \frac{c-1}{C-2} \in [0, 1] \tag{3.11}$$

for treebank binarization. In continuous parsing, $\rho^{\mathbf{B}} \in \{0, 1\}$ implies CNF. As an effect of binarization, new phrase label ∅ is introduced, as exemplified in Figure 3.3. The results are in Section 5.1.2. I further leverage substructure creation with this new label in Section 3.5.1.

3.3.3 Medoid for DM

Similar to binarization which is not only a must preprocessing to **CB** and **DB** but also a augmentation trick, the choice of medoid for **DM** also have these roles. For training and inference, I introduce a set of categorical *medoid* factors

$$\rho^{\mathbf{M}} \in \{random, leftmost, rightmost\} \tag{3.12}$$

to stratify a multi-branching tree: 1) *random* picks a random child with uniform probability, whereas 2) *leftmost* and 3) *rightmost* take the two ends of a discontinuous group.

Factors *leftmost* and *rightmost* are like CNF left and right which only produce static variation of trees. However, *random* yield stochastic combination of medoids of discontinuous phrases in a parse. Like the previous binarization, I do not introduce extra head information for **DM** as medoid. However, the model is able to produce what is very close to the phrasal head. The experiment is in Section 4.3.3.

3.3.4 Oracle

NCCP parsers are fully supervised models with ply action layers for each model component. Herein, I define signals in the follow format of each stratified tree,

- a flat sequence of signals as $x_{1:n}$ for $(x_1, \ldots, x_n)$, where n is the sequence length.
- H flat sequences of signals as $x^{1:H}_{1:n_h}$, where $h \in [1, H]$ is the index (i.e., height) of a flat sequence and $x^{h}_{n_h}$ is the h-th flat sequence of length n_h.

At minimum, a tree has three common types of signals:

- $x_{1:n}$ the sequence of terminals (i.e., words),
- $t_{1:n}$ the sequence of pre-terminals (i.e., PoS tags),
- $l^{1:H}_{1:n_h}$ the sequences of non-terminals excluding pre-terminals (i.e., sequences of constituent labels).

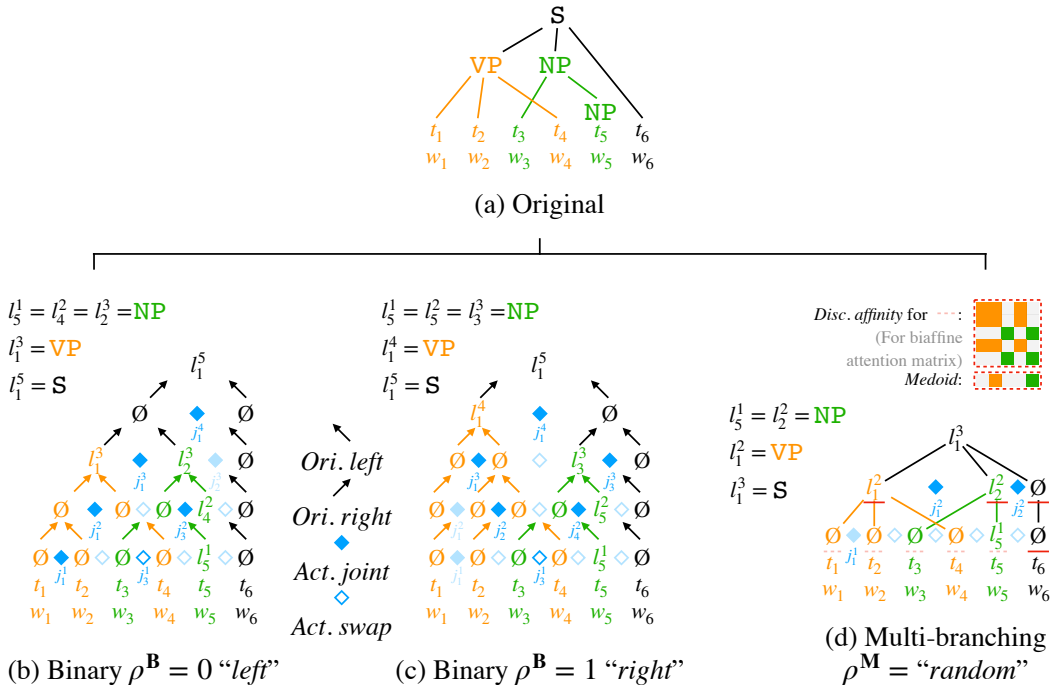
PoS tag and constituent label are multi-class signals. Noticeable, constituent label always has $\varnothing$ nodes when some non-terminals has both pre-terminals and non-terminals as children, as shown in Figure 3.4 (d). In addition to the general signals, I specify signals for each **NCCP** member as the following.

Signal from a binary tree. **CB** and **DB** commonly feature in *orientation* $o^{1:H-1}_{1:n_h}$ and **DB** has extended *joint* $j^{1:H-1}_{1:n_h-1}$. Both *orientation* and *chunk* are binary signals, as defined by Formulas 3.1 and 3.2.

$$\textbf{CB signals} : (x_{1:n}, t_{1:n}, l^{1:H}_{1:n_h}, o^{1:H-1}_{1:n_h}) \tag{3.13}$$

$$\textbf{DB signals} : (x_{1:n}, t_{1:n}, l^{1:H}_{1:n_h}, o^{1:H-1}_{1:n_h}, j^{1:H-1}_{1:n_h-1}) \tag{3.14}$$

Signal from a multi-branching tree. **CM** and **DM** commonly feature in *joint*. For **DM**, the extension includes *discontinuity* and *affinity* biaffine attention matrix. *Discontinuity* is defined by Fomula 3.10.



(a) Original

(b) Binary $\rho^{\mathbf{B}} = 0$ "*left*"

(c) Binary $\rho^{\mathbf{B}} = 1$ "*right*"

(d) Multi-branching $\rho^{\mathbf{M}} =$ "*random*"

FIGURE 3.4: Signals from a discontinuous tree. The original m-ary tree (a) is binarized and stratified into (b) and (c) with numeric factors $\rho^{\mathbf{B}}$, whereas (a) is stratified into (d) with a categorical *medoid* factor *random*. In (4), w_2 and w_5 are randomly selected as *medoids* for discontinuous parents l_1^2 and l_2^2 with more or less twisted descendant lines. I color constituent components and show *joints* with light blue "◆" and "◇" which are not activated by agreeing orientation condition.

$$\textbf{CM signals}: (x_{1:n}, t_{1:n}, l_{1:n_h}^{1:H}, j_{1:n_h-1}^{1:H-1}) \tag{3.15}$$

$$\textbf{DM signals}: (x_{1:n}, t_{1:n}, l_{1:n_h}^{1:H}, j_{1:n_h-1}^{1:H-1}, d_{1:n_h}^{1:H-1}, a_{[1:d_h,1:d_h]}^{1:H-1}), \tag{3.16}$$

where $d_h = \sum_{i=1}^{n_h} d_i^h \leq n_h$ indicates the number of discontinuous nodes that is no larger than the number of total nodes in layer h.

I visualize an instance of the signals from a discontinuous tree in Figure 3.4. In the case of a continuous tree, the stratification produces fewer signals with simpler structure. I can view **CB** as a special case of **DB** where $action(x_i \oplus x_{i+1}) = joint$ stands for each i. Thus, $j_{1:n_h-1}^{1:H-1} = chunk\ inside$ stands for the supervision signals of **CB**. In the same vein, **CM** as a special case of **DM** has $action(x_i) = continuous$ for the model and $d_{1:n_h}^{1:H-1} = continuous$ for the supervision signals.

3.4 Model Implementation

3.4.1 Neural Component

Feedforward Neural Network. Feedforward neural network (FFNN) is the simplest type of artificial neural network. It connects every unit in an input vector (i.e., input vector as an input layer) and weights it to every unit in its output vector (i.e., output vector as an out layer). An FFNN can be a single-layer perceptron (SLP) without a hidden vector (i.e., a hidden layer) or be a multi-layer perceptron (MLP) by stacking multiple SLP which creates one or more hidden layers.

Specifically, I show the case of SLP as following,

$$y = activation(Wx + b)\,,$$

where $x, y \in \mathbb{R}^{\mathbb{N}}$ are the input and the output vectors, respectively. $W \in \mathbb{R}^{\mathbb{N}\times\mathbb{N}}$ and $b \in \mathbb{R}^{\mathbb{N}}$ are weight matrix and bias vector, which belong to model parameters that adapt to the input and output during the training process with backpropagation. The $activation : \mathbb{R} \to \mathbb{R}$ function is differentiable, which maps $Wx + b$ to activated output y. By linking the output of a SLP to the input of the other SLP, I get an MLP with a hidden layer. The *activation* function plays an curcial role in MLP. Without it (or with an linear activation function), the MLP will reduce to an equivalent SLP.

I use `FFNN` referring to a 2-layer MLP by default in this thesis.

Long Short-Term Memory. Long short-term memory (`LSTM`) is special recurrent neural network (RNN) for modeling a sequence of input vectors $(x_1, \ldots, x_n)$ and a sequence of output vectors $(y_1, \ldots, y_n)$, where n is the sequence length. A single-layer `LSTM` network contains four SLPs as the gate functions to modify the state vector sequence $(h_0, \ldots, h_n)$ and cell vectors $(c_0, \ldots, c_n)$ through time steps. Vectors of h_0 and c_0 are the initial vector, which can be zero vectors or adaptive vectors of the model parameters. Any $x_i \in \mathbb{R}^{\mathbb{N}}$ can have a different dimensionality from other vectors in an `LSTM`.

Specifically, a single-layer `LSTM` network is organized as following,

$$\begin{aligned}
i_i &= \sigma_i(W_i x_i + U_i h_{i-1} + b_i) \\
f_i &= \sigma_f(W_f x_i + U_f h_{i-1} + b_f) \\
y_i &= \sigma_o(W_o x_i + U_o h_{i-1} + b_o) \\
\hat{c}_i &= \sigma_c(W_c x_i + U_c h_{i-1} + b_c) \\
c_i &= f_i \odot c_{i-1} + i_i \odot \hat{c}_i \\
h_i &= y_i \odot \sigma_h(c_t)\,.
\end{aligned}$$

The SLPs for i_i, f_i, and y_i are input gate, forget gate, and output gate, which regulate time intervals in order to compose cell vector c_i (with cell candidate $\hat{c}_i$) and the hidden state h_i. Functions of σs are activation function for different gates or the other vectors. Typically, gate activation σ_i, σ_f and σ_o are sigmoid function and the others are hyperbolic tangent function. The model parameters are updated via back-propagation through time.

Like MLP, multiple `LSTM` layers can be stacked. An `LSTM` is usually unidirectional and forward, which modeling the sequence from left to right (or vice versa). Otherwise, by concatenating a forward and a backward `LSTM`, a bidirectional `LSTM` (`BiLSTM`) layer forms. I use `BiLSTM` referring to a RNN module of one or more stacked bidirectional `LSTM` layers.

When make comparison, I also use bidirectional Quasi-Recurrent Neural Network (Bradbury et al., 2017, `BiQRNN`). This network is a member of the RNN family with a simpler gating mechanism.

Loss Function. Meanwhile, a loss function for each single signal comes in respective two types: binary and multi-class. I take `HINGE-LOSS` as the elementary loss function for binary signals,

$$\texttt{HINGE-LOSS}(t, y) = \max(0, 1 - t \cdot y)\,,$$

where $t \in \{0, 1\}$ is a target gold answer from supervision and $y \in \mathbb{R}$ is the model prediction. I choose CROSS-ENTROPY as the elementary loss function for multi-class signals,

$$\texttt{CROSS-ENTROPY}(t, y_{1:n}) = -\sum_{i=1}^{n} \mathbb{1}(t \neq i) \cdot \log(y_i),$$

where $t \in [0, n)$ is a target gold answer from supervision and $y \in (0, 1)^n$ is the model prediction of a distribution for n classes with $\sum_{i=1}^{n} y_i = 1$. I refer BCE-LOSS to the binary version of CROSS-ENTROPY.

3.4.2 Pre-trained Word Embedding and Language Model

The process of training is the process of encoding the relationship between input and output into the model parameters to form inductive bias. Different training processes encode different levels of information into the models.

Given a sequence of $(x_1, \ldots, x_n)$, *pre-trained word embeddings* (Mikolov et al., 2013; Bojanowski et al., 2017, i.e., PWE) encode the node relationship within a span of a fixed-length n, whereas *pre-trained language models* (Devlin et al., 2019; Yang et al., 2019, i.e., PLM) encode the node relationship with in a full span of a variable length n. In other words, PWE encodes local information in a small range (at sub-phrase level) but PLM further globally contextualizes information of the input sequence (at sentence level). An embedding vector in PWE is static no matter what context it has. In contrast, an embedding vector in PWE depends on its context.

In terms of efficiency and effectiveness, a PWE is usually compact and fast. Furthermore, fastText (Bojanowski et al., 2017) encodes subword information and eliminates the effect of unknown words at inference phase. Meanwhile, a PLM costs more computation for both training and inferring, which usually perform better than a PWE except for some very short sentences.

Finally, just mildly mention that classic *generative language modeling* is a way of producing strings not from a formal grammar but from the perspective of probability. Such a model can both generate (via sampling) and evaluate (via assigning probabilities to) strings for a language. Some PLMs fail to be generative because they are bidirectional, such as ELMo and BERT.

3.4.3 Overall Architecture

All **NCCP** members share the same outer loop and neural components introduced in Section 3.4.1, as shown in Algorithm 1. The PARSE function takes word embeddings $e_{1:n}$ from words $x_{1:n}$ and contextualizes them with $\texttt{BiLSTM}_{ctx}$ as the 0-th layer for the next PoS tagging and interative structural labelling processes. I implemented $\texttt{FFNN}_{tag}$ and $\texttt{FFNN}_{label}$ as 2-layer MLP with parameters in their first layers shared. Supervision signals, $t_{1:n}$ and $l^{1:H}_{1:n_h}$, meet model prediction, $\hat{t}_{1:n}$ and $\hat{l}^{1:H}_{1:n_h}$, creating losses of L_{tag} and L_{label} with CROSS-ENTROPY loss function. For each plies, the FOLD function differs for different **NCCP** members. $\texttt{BiLSTM}_{ply}$ provide further layer-wise contextualization $z^h_{1:n_h}$ for $x^h_{1:n_h}$.

Again, the *chunk* component of **CM** and **DM** is denoted as *joint* for their common interstice position with **DB**, following the previous signal unification.

3.4.4 CB & DB: Binary Implementation

CB and **DB** share identical structure except $\texttt{FFNN}_{chunk}$ and L_{chk}, which are exclusively for **DB**, as shown in Algorithm 2. The orientation function is hinted by $\texttt{BiLSTM}_{ply}$. A

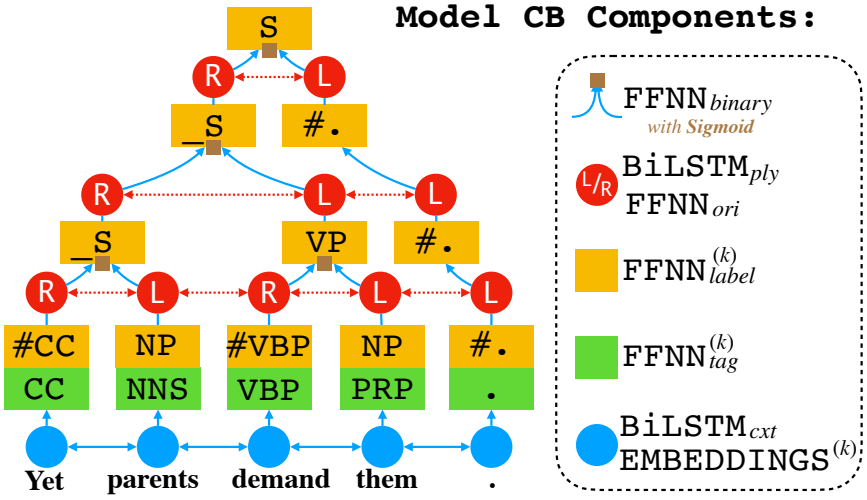


FIGURE 3.5: Example of **CB**. The simplest member of the **NCCP** family solely based on *orientation* for the tree structure.

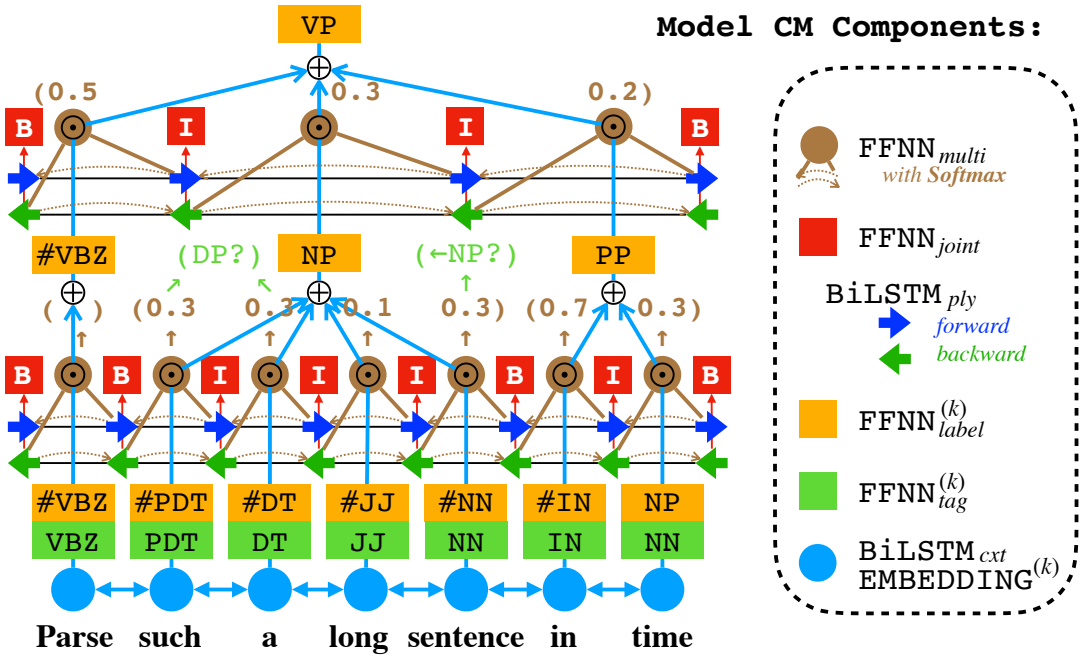


FIGURE 3.6: Example of **CM**. The *chunk* signals are calculated from the difference of the forward and backward states of the $\texttt{BiLSTM}_{ply}$. Exemplary fractions show the gate weights under Softmax constraints for vector compositionality, which are unsupervised and offer evidence for phrase headedness. The same mechanism works for **DM**.

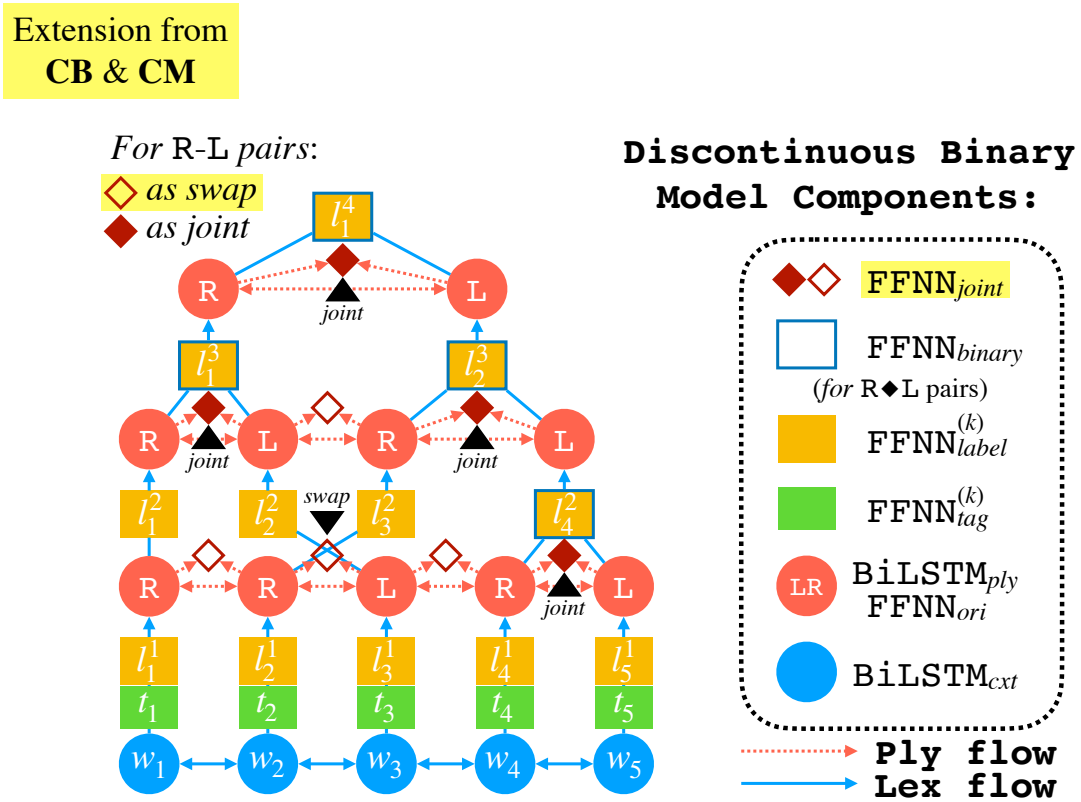


FIGURE 3.7: Example of **DB**. **DB** produces *swap* to facilitate traveling of discontinuous nodes and *joint* to combine adjacent nodes.

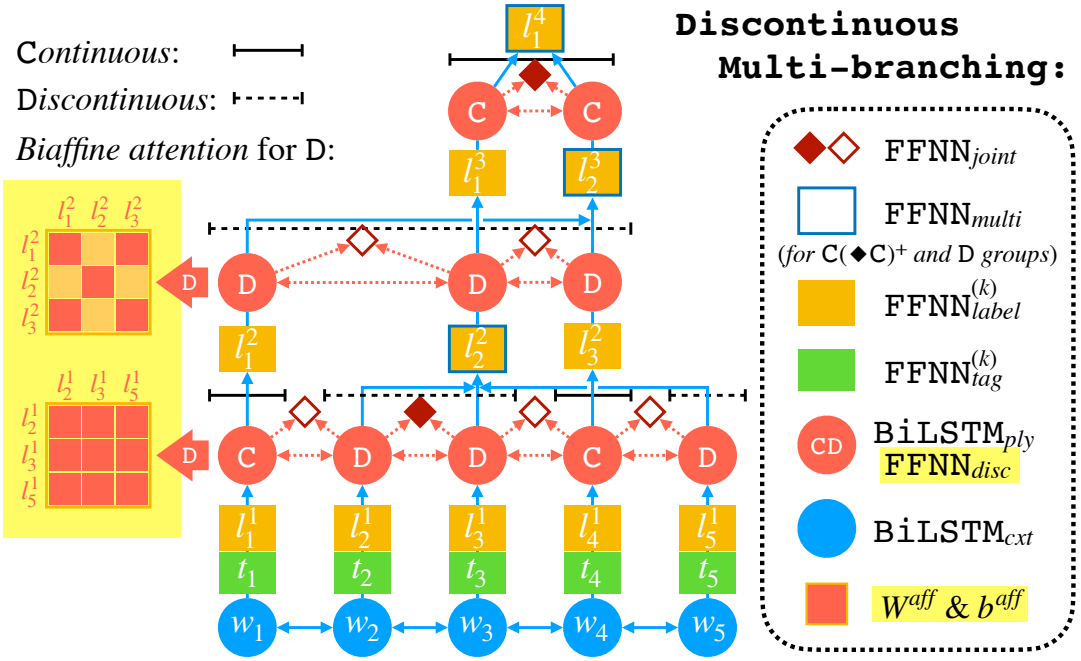


FIGURE 3.8: Example of **DM**. **DM** leverages *affinity* biaffine attention to identify and combine discontinuous groups.

Algorithm 1: Combinatory Parsing

Function $\texttt{PARSE}(x_{1:n})$**:**
  $ply \leftarrow []$ and $n_h \leftarrow n$ with height $h \leftarrow 1$;
  $x^h_{1:n_h} \leftarrow \texttt{BiLSTM}_{cxt}(\textit{embeddings of } x_{1:n})$;
  for $i \leftarrow 1$ **to** n **do**
    $\hat{t}_i \leftarrow \texttt{FFNN}_{tag}(x^h_i)$;
    $\hat{l}^h_i \leftarrow \texttt{FFNN}_{label}(x^h_i)$;
    append a tree with $(x_i, \hat{t}_i, \hat{l}^h_i)$ to ply;
  while $n_h > 1$ **do**
    $z^h_{1:n_h} \leftarrow \texttt{BiLSTM}_{ply}(x^h_{1:n_h})$;
    $x^{h+1}_{1:n_{h+1}} \leftarrow \texttt{FOLD}(ply,\ x^h_{1:n_h},\ z^h_{1:n_h})$;
    $h \leftarrow h + 1$;
    for $i \leftarrow 1$ **to** n_h **do**
      $\hat{l}^h_i \leftarrow \texttt{FFNN}_{label}(x^h_i)$;
      label i-th tree of ply with $\hat{l}^h_i$;
  return the sole tree in ply;

Algorithm 2: Binary Ply

Function $\texttt{FOLD}(ply,\ x_{1:n},\ z_{1:n})$**:**
  for $i \leftarrow 1$ **to** n **do**
    $\hat{o}^h_i \leftarrow \texttt{FFNN}_{ori}(z_i)$;
    $\hat{j}^h_i \leftarrow \texttt{FFNN}_{joint}(z_i \oplus z_{i+1})$; `// Only for DB`
    w.r.t. Section 3.1.1 for **CB** or Section 3.2.1 for **DB**, apply actions to ply and $x_i \leftarrow \texttt{COMPOSE}(x_i, x_{i+1})$;

Function $\texttt{COMPOSE}(x_L,\ x_R)$**:**
  $\lambda \leftarrow \sigma\ \texttt{FFNN}_{binary}(x_L \oplus x_R)$;
  return $\lambda \odot x_L + (1 - \lambda) \odot x_R$;

single-layer $\texttt{FFNN}_{ori}$ with a threshold reduces the outputs to an integer of either 0 or 1 to indicate two possible orientations. The COMPOSE function uses sigmoid activation "σ" to create a pair of complementary gates for x_L, x_R from λ, which is a vector of the same size as the embeddings. "$\odot$" represents pointwise multiplication.

CB and **DB** only use $O(n)$ components, as shown in Figures 3.5 & 3.7.

3.4.5 CM & DM: Multi-branching Implementation

To resemble the binary gate λ, I use the Softmax function for each chunk or group, as described in Algorithm 3. Binary interpolation λ is a special case of the multi-branching λ_{group} because Sigmoid and Softmax functions are closely related. Examples of **CM** and **DM** are shown in Figures 3.6 & 3.8.

Vectors of $z_{1:n}$ and $\Delta_{1:n}$ are exchangeable. I choose the configuration using $\Delta_{1:n}$ in Algorithm 3 because it performs slightly optimal in my experiments. Similarly, λ_i in COMPOSE function is a vector. I consider average of λ_i (i.e., $\bar{\lambda}_i$) for inference and headedness visualization. **DM** infers with a special factor

$$\rho^{\mathbf{M}} = \textit{uhead}\,, \tag{3.17}$$

Algorithm 3: Multi-branching Ply

Function FOLD(ply, $x_{1:n}$, $z_{1:n}$):
  $(\vec{z}_{1:n} \oplus \overleftarrow{z}_{1:n}) \leftarrow z_{1:n}$ (forward & backward);
  for $i \leftarrow 1$ **to** n **do**
    $\hat{d}_i^h \leftarrow \texttt{FFNN}_{disc}(z_i)$; // Only for DM
    $\Delta_i \leftarrow (\vec{z}_i - \vec{z}_{i-1}) \oplus (\overleftarrow{z}_i - \overleftarrow{z}_{i+1})$;
    $\hat{j}_i^h \leftarrow \texttt{FFNN}_{joint}(\Delta_i \oplus \Delta_{i+1})$;
  foreach i, j-th *discontinuous* $\hat{d}_{v,w}^h$ **do** // Only for DM
    $\hat{a}_{[i,j]}^h \leftarrow x_v^\top \cdot W^{aff} \cdot \Delta_w + b^{aff}$; // Only for DM
  find *discontinuous* groups by **check**ing $\hat{a}^h$; // Only for DM
  foreach group indices $\mathcal{G}$ and its *medoid* **do**
    w.r.t Section 3.1.2 for **CM** or Section 3.2.2 for **DM**, apply actions to ply and $x_{medoid} \leftarrow \texttt{COMPOSE}(\Delta_\mathcal{G}, x_\mathcal{G})$;
  return CONDENSE($x_{1:n}$);
Function COMPOSE($\Delta_\mathcal{G}$, $x_\mathcal{G}$):
  $\lambda_\mathcal{G} \leftarrow Softmax(\texttt{FFNN}_{multi}(\Delta_\mathcal{G}))$;
  return $\sum_i^\mathcal{G} \lambda_i \odot x_i$;

which takes $x_{medoid} \leftarrow \arg\max_{i \in \mathcal{G}} \bar{\lambda}_i$ as the group medoid.

To identify discontinuous groups in *affinity biaffine attention* matrix, **DM** takes $M = \sigma \hat{b}_{[1:D,1:D]}^h$ in range $(0,1)$ ($D = \sum_i \hat{d}_i^h$) and booleanizes it into $B \leftarrow M > \theta$. It 1) tries default threshold $\theta = 0.5$ as natural selection for sigmoid activation and **check** whether all following statements are true:

- B is symmetric,
- any rows $v, w \in B$ are $v \neq 0$,
- either $v = w$ or $v^\top \cdot w = 0$.

It succeeds in most cases. Otherwise, it 2) tries a value from M as θ, **check**s and loops again. I order the thresholds by their distances to the default 0.5. If all 2) fail, it 3) simply falls back in grouping all nodes as one and count one FAIL.

3.4.6 Multilingualism and Structured Sentiment Analysis

For both continuous and discontinuous constituency parsing, there are large annotated treebanks in different languages. Clearly, a common aspect of those treebanks is the phrase structure with different PoS tagging and constituency tagging schemes.

Because my approach decomposes tagging and labelling from the structure construction, I can achieve multilingualism with a very minor modification on Algorithm 1. As highlight in Algorithm 4, I pass an additional argument k for the PARSE function to indicate the language and the scheme of the current treebank. Because the original $\texttt{FFNN}_{tag}$ and $\texttt{FFNN}_{label}$ are 2-layer MLPs which share their first layer, $\texttt{FFNN}_{tag}^k$ and $\texttt{FFNN}_{label}^k$ only differ in their second layers.

Algorithm 4: Multilingual Neural Combinatory Parsing

Function PARSE$(k, x_{1:n})$**:**
  $x^h_{1:n_h} \leftarrow \texttt{BiLSTM}_{cxt}$(*embeddings of* $x_{1:n}$ *for corpus* k)
  ...
  for $i \leftarrow 0$ **to** $n-1$ **do**
    $\hat{t}_i \leftarrow \texttt{FFNN}^{\,k}_{tag}(x^0_i)$
    $\hat{l}_i \leftarrow \texttt{FFNN}^{\,k}_{label}(x^0_i)$
  ...
  while $n_h > 1$ **do**
    for $i \leftarrow 1$ **to** n_h **do**
      $\hat{l}^h_i \leftarrow \texttt{FFNN}^{\,k}_{label}(x^h_i)$
  ...

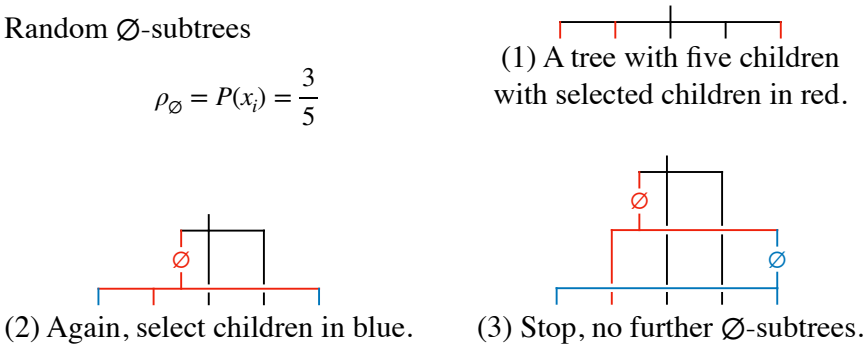


FIGURE 3.9: A example for the creation of a random ∅ branch from a flat tree.

3.5 Model Robustness

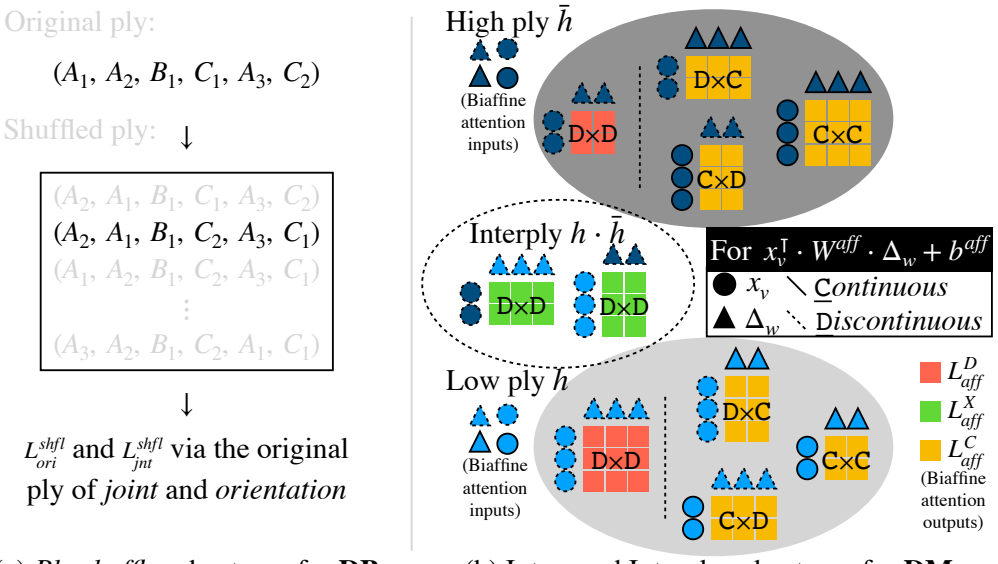
3.5.1 Additional Substructure with Empty Node

Innate empty node. Structure of ∅-tree has several causes. First, both processes of binarization and stratification cause innate ∅ nodes, as specified in Sections 3.3.2 and 3.3.4.

$$\rho^{\mathbf{B}} \sim \begin{cases} \text{Bernoulli}(\alpha_{right}) & \text{for } \mathbf{CB} \\ \text{Beta}(\alpha_{left}, \alpha_{right}) & \text{for } \mathbf{DB} \end{cases} \tag{3.18}$$

As for binarization, Formula 3.18 illustrates sampling binarization factor for **CB** & **DB** and *medoid* for **DM**. I use Bernoulli distribution for **CB** and beta distribution for **DB** to resemble **DM**'s uniform $\rho^{\mathbf{M}}$ = *random* distribution or reflect linguistic branching tendency with **CB** & **DB**'s $\rho^{\mathbf{B}}$. The choice of different distribution functions for **CB** & **DB** is based on my publications.

Random empty node. Then, besides the innate ∅-subtrees, I create more the intermediate ∅-subtrees with a random process. The ∅-subtree is an inspiration of **CM**'s deterministic _SUB node. The latter balances subtrees without increasing their heights and improves accuracy with efficiency. However, **(c)** ∅-subtree is random and creates imbalance. It creates only one stretching branch by iteratively grouping nodes with possibility $\rho_\varnothing$, which has three significant aspects:



(a) *Ply shuffle* robustness for **DB** (b) Intra- and Interply robustness for **DM**

FIGURE 3.10: (a) Constituent children get shuffled and create additional losses. (A_1, A_2, A_3), B_1, and (C_1, C_2) belongs to three different constituents. (b) Pick in-ply continuous nodes and interply nodes for **DM** biaffine attention.

- Random stretching branches add mild variations to the context as state for robust ply actions in `FOLD`.
- Random discontinuity creates **DB** orientation layers that are impossible by $\rho^{\mathbf{DB}}$ binarization.
- They reduce large (possibly continuous) constituents into smaller (possibly discontinuous) pieces without adding a large payload to biaffine attention, which narrows the gap between **DB** and **DM** (**DM** is more vulnerable to dramatic many-to-one compose).

Taking `NP` "a good day" for instance, any of "a day", "a good", and "good day" can be an intermediate option for creating the `NP`. On the one hand, these options create varied contexts for the remaining parts of a ply. On the other hand, assume that "a day" (which is not a $\rho^{\mathbf{DB}}$ product) is selected. **DM** learns to discern it with the other possible "a day" in *biaffine attention* based on their context.

3.5.2 Basic Loss Item

We choose `HINGE-LOSS` for binary prediction and `CROSS-ENTROPY` for multi-class prediction, following **NCCP**. Respecting the context in Algorithms 1–3, our basic loss items are

$$L_{tag},\ L_{label},\ L_{ori},\ L_{jnt},\ L_{disc},\ L^D_{aff}$$

by accumulating their items across all layers. E.g., $L^D_{aff} \leftarrow \sum_{i,j,h} \texttt{HINGE-LOSS}(a^h_{[i,j]}, \hat{a}^h_{[i,j]})$ with D for *discontinuous affinity*. We have additional loss items in the following subsections.

3.5.3 DB Robustness Loss Item: Ply Shuffle

To further let **DB** cover more ply variants, I introduce (a) *shuffle* and its resultant losses L^{shfl}_{ori} and L^{shfl}_{jnt}. It shuffles child order of a constituent, takes the new sequence

to `FOLD`, and reuses the ply of *orientation* and *joint* for additional losses. For example, a `VP` to the left of a `NP` gets shuffled to the right with the same ply "*right* ◆ *left*" producing the additional losses. **DB**'s additional loss items are

$$L_{ori}^{shfl},\ L_{jnt}^{shfl}.$$

3.5.4 DM Robustness Loss Item: Intra- and Interply

Continuity and discontinuity in **DM** undergo different identification processes via *discontinuity*. To minimize the difference, in addition to L_{ba}^{D} for cardinal *discontinuous* nodes from the same ply, I introduce (b) L_{ba}^{C} and L_{ba}^{X} for continuous and interply *affinity*. These reduce the risk of *biaffine attention* forwarding incorrect *discontinuity* to weird actions because of unseen cases. I use $\beta_c \cdot \sigma(\hat{d}_i^h)$ and $\beta_x \cdot \sigma(x_w^{h\top} \cdot W^{ba} \cdot \Delta_w^{\bar{h}} + b^{ba})$ to limit the sample size, where $\bar{h}$ is the next ply after h containing *discontinuous* nodes. Fallible signals are more likely to form losses via `HINGE-LOSS`. **DB**'s additional loss items are

$$L_{aff}^{C},\ L_{aff}^{X}.$$

Chapter 4

Experiments

4.1 General Setting

Data. For continuous parsing, treebanks PTB and CTB are widely chosen for experiments. I follow previous data splits for PTB, CTB, and KTB. Specifically, PTB's sections 2–21 were used for training, section 22 for development, and section 23 for testing. CTB's articles 001–270 and 440–1151 were used for training, 301–325 for development, and 271–300 for testing. There is no widely accepted data split for the KTB corpus, except for some random divisions, because KTB contains mixed data from sources such as newswires, book digests, and Wikipedia. I randomly reserve 2,075 samples for development, 1,863 samples for testing, and the remaining 3.3 million as training samples. (Check my code[1] for the random seed and other details.) The Evalb program[2] provides F1 scores for evaluation.

For discontinuous parsing, treebanks DPTB and German TIGER are the most commonly used testbeds. The data split of DPTB follows PTB as described above, whereas TIGER's common splits are sentences 1–40474 as train set, 40475–45474 as development set, and 45475–50474 as test set. The Discontinuous DOP program [3] offers both overall and discontinuous F1 scores for evaluation. The preprocessing of data is described in Section 3.3.4.

Model. To compare with other parsers with a lexical component and exploit contextualization, **NCCP** takes frozen pre-trained fastText as static word embedding (PWE) or fine-tuned 12-layer pre-trained XLNet or BERT as contextualized embedding of pre-trained language models (PLM) as lexical input.

I choose fastText (Bojanowski et al., 2017) as PWE because it can be easily obtained either online or via pre-training from scratch with the corpora at hand. The difference is examined in Section 4.2.1. When training from scratch, PTB texts are fed into *cbow* instead of *skipgram* with their default settings for 50 epochs (denoted as **CB**/E). Meanwhile, online official pre-trained embeddings provide the default lexical inputs. English (*wiki.en.bin*), Chinese (*cc.zh.300.bin*), German (*cc.de.300.bin*) and Japanese (*cc.ja.300.bin*) embeddings[4] are used for (D)PTB, CTB, TIGER, and KTB, respectively.

Meanwhile, PLM models are useful for various tasks (Kitaev and Klein, 2018; Kitaev and Klein, 2020; Zhou and Zhao, 2019; Yang and Deng, 2020; Mrini et al., 2020). I chose XLNet (Yang et al., 2019) for English and BERT for German[5]. Specifically, either a 1-layer is used to convert the 768-unit output to the model size. **NCCP** model

[1] https://github.com/tmu-nlp/UniTP
[2] https://nlp.cs.nyu.edu/evalb/
[3] https://github.com/andreasvc/disco-dop
[4] https://fasttext.cc/
[5] https://github.com/huggingface/transformers and https://www.deepset.ai/german-bert

size is 300 for vector compositionality. The $\texttt{BiLSTM}_{cxt}$ $(/n)$ has six layers by default. The hidden sizes for 2-layer $\texttt{FFNN}_{label}$, $\texttt{FFNN}_{ori}$, and $\texttt{FFNN}_{chunk}$ are less than 300. I use a GeForce GTX 1080 Ti with 11 GB to test inference speed. The batch size for training is 80 and gets doubled to 160 at the inference phase.

I used the Adam optimizer with a default learning rate of 10^{-3}, while I opted for the XLNet's Adam hyperparameters when tuning the pre-trained XLNet (e.g., their learning rate was 10^{-5}). `HINGE-LOSS` was the default criterion for orientation while binary cross-entropy (`BCE-LOSS`) was tested. I adopted a warm-up period for one epoch and a linear decrease after the 15-th decrease since the last best evaluation. The recurrent dropout rate was 0.2; other dropout probabilities for FFNNs were set to 0.4. For model selection, the training process terminated when the development set did not improve above the highest score after 100 consecutive evaluations.

4.2 Continuous Parser

Specific settings. For the binary model **CB**, I explored training with dynamic datasets via sampling two CNF factored datasets

$$\rho^{\mathbf{B}} \sim \text{Bernoulli}(\alpha_{right}) \in \{0, 1\} \tag{3.18}$$

indicating CNF-left (0) and CNF-right (1). This is because of the following: 1) The experiments with the non-CNF factors (i.e., **midin** and **midout** by Chen et al. (2021)) did not yield any promising results; thus, I have not reported them. 2) The language was loosely left-branched, right-branched, or did not show a noticeable tendency. Moreover, the use of a single static dataset may introduce a severe orientation bias. 3) All factors are intermediate variables and equally correct. I denote the Bernoulli sampling strategies with two static CNF-factored datasets at certain ratios and named each strategy in the format "$\text{L}(1-\alpha_{right}\%)\text{R}(\alpha_{right}\%)$" according to the ratio percentages. My experiments mainly focus on **CB** because of **CM** do not have corresponding data augmentation and is less accurate than **CB**.

The coefficients of the three losses were explored and the default were

$$L_{\text{CB}} = 0.2 \cdot L_{tag} + 0.3 \cdot L_{label} + 0.5 \cdot L_{ori} \tag{4.1}$$

$$L_{\text{CM}} = 0.2 \cdot L_{tag} + 0.3 \cdot L_{label} + 0.5 \cdot L_{chk} \tag{4.2}$$

The binary model **CB** is a representative **NCCP** model on which I did the major ablation work for neural configuration to understand the characteristics of other models. For example, the influence of extra information from official pre-trained fastText, different numbers of $\texttt{BiLSTM}_{cxt}$ $(/n)$ layers, not fine-tuning but leveraging contextualization of PLM with a 1-layer `FFNN` $(/0)$ or an n-layer `BiLSTM` $(/n^{+})$ for model adaptation, and the dimension of vector compositionality are explored.

4.2.1 Constituency Parsing

Overall Results

Table 4.1 lists the parsing accuracy and speeds of the single models in ascending order according to their F1 scores for the PTB corpus. The transition-based parsers with $O(n)$ complexity appear at the top of the table, followed by other types of models, and the chart parsers running in $O(n^3)$ time are at the bottom of the table. The models exhibit similar trends for the CTB. Shen et al. (2018b) and my models belong to type O and have similar complexities. Generally, the accuracy follows the

Corpus	Penn Treebank					Chinese Treebank			
Single Model without PLM	**Type**	**Speed**	**LP**	**LR**	**F1**	**Type**	**LP**	**LR**	**F1**
Watanabe and Sumita (2015)	T↑ (32)	-	-	-	90.7	T↑ (64)	-	-	84.3
Gómez-Rodríguez and Vilares (2018)	O	898	-	-	90.7	O	-	-	83.1
Cross and Huang (2016)	T↑ (1)	-	92.1	90.5	91.3	-	-	-	-
Liu and Zhang (2017)	T↓ (16)	79.2	92.1	91.3	91.7	T↓ (16)	85.9	85.2	85.5
Stern, Andreas, and Klein (2017)	C	75.5	93.0	90.6	91.8	-	-	-	-
Shen et al. (2018b)	O↓ (1)	111.1	92.0	91.7	91.8	O↓ (1)	86.6	86.4	86.5
Charniak and Johnson (2005)	C	-	-	-	92.1	-	-	-	-
PWE **NCCP CM** with fastText	O↑ (1)	1122.6	92.1	92.1	92.1	O↑ (1)	86.0	84.7	85.3
PWE **NCCP CB** with fastText	O↑ (1)	**1327.2**	92.8	92.3	92.5	O↑ (1)	85.8	86.2	86.0
Nguyen et al. (2020)	O↓ (1)	130.2	92.8	92.8	92.8	-	-	-	-
Kitaev and Klein (2018)	C	212.5	93.9	93.2	93.6	C	91.9	91.5	91.7
Wei, Wu, and Lan (2020)	O↓ (1)	155	94.1	93.3	93.7	O↓ (1)	89.9	87.4	88.7
Zhou and Zhao (2019)	C	226.3	93.9	93.6	93.7	C	92.3	92.0	92.2
Zhang, Zhou, and Li (2020)	C	1092	94.2	94.0	94.1	C	89.7	89.9	89.8

TABLE 4.1: Single-model results on PTB and CTB test datasets sorted by the F1 scores on PTB. Transition-based parsers, chart parser, and others are marked as T, C, and O, respectively; ↑ and ↓ denote bottom-up and top-down. The number in brackets indicates the beam size. Speeds are measured in sentences per second. Kitaev and Klein (2018) used Tesla K80, and the CTB scores are cited from Kitaev, Cao, and Klein (2019). Zhou and Zhao (2019) used GeForce GTX 1080 Ti (same condition).

Fine-Tuned Model	**F1**	**sents/sec**	**Type**
Kitaev and Klein (2018)	95.13	70.8	C
Kitaev and Klein (2020)	95.44	1200	T
Nguyen et al. (2020)	95.48	-	O↓
Zhang, Zhou, and Li (2020)	95.69	-	C
Wei, Wu, and Lan (2020)	95.8	-	O↓
Xin, Li, and Tan (2021)	95.9	26	C
Zhou and Zhao (2019)	96.33	64.8	C
Yang and Deng (2020)	96.34	71.3	T
Mrini et al. (2020)	96.38	59.2	C
PLM **CB** with XLNet	95.72	411.2	O↑
PLM **CM** with XLNet	95.44	369.4	O↑
PLM **CB** with XLNet & 2-layer $\texttt{BiLSTM}_{cxt}$	94.67	398.4	O↑

TABLE 4.2: Improvements with pre-trained language models. I used a greedy search algorithm on single GeForce GTX 1080 Ti. Rows 6–8 are reported by Yang and Deng (2020) using GeForce GTX 2080 Ti. Kitaev and Klein (2020) used a cloud TPU with a beam search algorithm and a larger batch size.

CB with	**Frozen fastText**		**Frozen XLNet**	
n-layer $\texttt{BiLSTM}_{cxt}$	**F1**	**sents/sec**	**F1**	**sents/sec**
0	65.02	1386.6	89.24	411.2
2	91.34	1350.0	93.74	398.4
6	92.54	1327.2	93.89	382.7

TABLE 4.3: Effectiveness of using frozen static word embeddings or dynamic sub-word language model and corresponding peak speed.

PWE	Specification	F1
CB/r	with random initialization.	91.73
CB/e	with tuned official fastText.	91.69
CB/E	with frozen pre-trained fastText from PTB.	92.31
CB/F	$\texttt{BiLSTM}_{ply}$ into $\texttt{FFNN}_{ply}$.	88.97
CB/R	$\texttt{BiLSTM}_{ply}$ into $\texttt{BiQRNN}_{ply}$.	92.42
CB/L	$\texttt{BiLSTM}_{ply}$ with `BCE-LOSS`.	92.32
CB/B	Biaffine inputs for vector λ in cubic time.	92.53
CB/V	$x_L \oplus x_R$ as input for vector interpolation λ.	**92.54**
CB/S	$x_L \oplus x_R$ as input for scalar interpolation λ.	91.83
CB/v	No input; bias vector interpolation λ.	92.36
CB/s	No input; bias scalar interpolation λ.	91.95
CB/+	Simple adding inputs by $x_L + x_R$	91.86

TABLE 4.4: Results of ablation studies on fastText (top) and $\texttt{BiLSTM}_{ply}$ (bottom) of the binary model. **CB**/V is the main experimented **CB** variant in other subsections.

complexity, whereas the speed roughly follows the year of publication rather than complexity or type.

I fine-tuned PLM models[6] and compare them with other parsers using fine-tuned language models. These are listed in Table 4.2. Owing to XLNet, model complexities grow to $O(n^2)$.

Ablation Study

Models with PWE (fastText). I investigate the binary model through ablation. The impacts of fastText are presented in the upper part of Table 4.4. **CB**/r does not require any external data beyond PTB, which is comparable to models without a pre-trained GloVe Pennington, Socher, and Manning (2014).

Then, I replace $\texttt{BiLSTM}_{ply}$ with an $\texttt{FFNN}_{ply}$ to examine its effect. The results are in the bottom rows. The comparison proves whether the embeddings are collaborative for the orientation signals because `FFNN` regards each input independently.

Lastly, I examine the binary combinator of **CB** in the lower part of Table 4.4. When replacing the `COMPOSE` Algorithm 2 with $x_L + x_R$, additive vector compositionality is retained (Mikolov et al., 2013) as the naïve **CB/+** variant. The model can infer a full tensor tree; however, **ADD** causes the vector magnitude to increase with the tree height cumulatively. This is unwanted in the recurrent or recursive neural network. I also examine other variants that do not cause such recursive problem.

In terms of the F1 score, the most competitive variants of the main **CB**/V are **CB**/B and **CB**/v, suggesting that fine interpolation can effectively facilitate vector compositionality. The similarity in results of **CB**/S, **CB**/s, and **CB**/+ validate this suggestion. This indicates that vector compositionality is not as trivial as an additive function at the scalar level, and a matrix operation is sufficient. **CB**/B is the costliest variant with a tensor operation that runs very slowly (30 sents/sec).

Finally, I used a grid search to explore the hyperparameter space of the three-loss coefficients. Figure 4.1 shows that the performance correlates to the orientation loss the most, but it is not overly sensitive to the hyperparameters.

[6]For the fine-tuned XLNet, using either the leftmost or rightmost sub-word yielded similar results earlier. However, averaging sub-words produced F1 scores under 94.

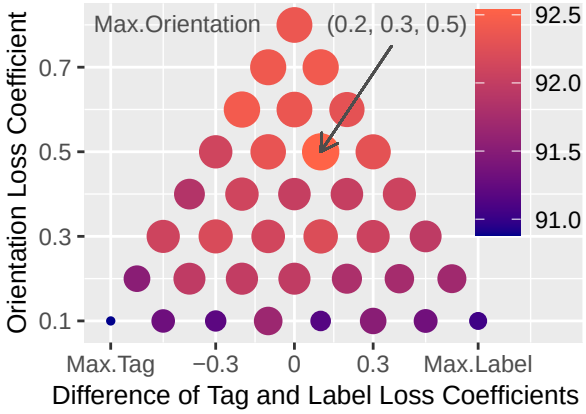


FIGURE 4.1: Grid search with an interval of 0.1 in a space of (tag, label, orientation) loss coefficients. The best was (0.2, 0.3, 0.5) indicated by an arrow.

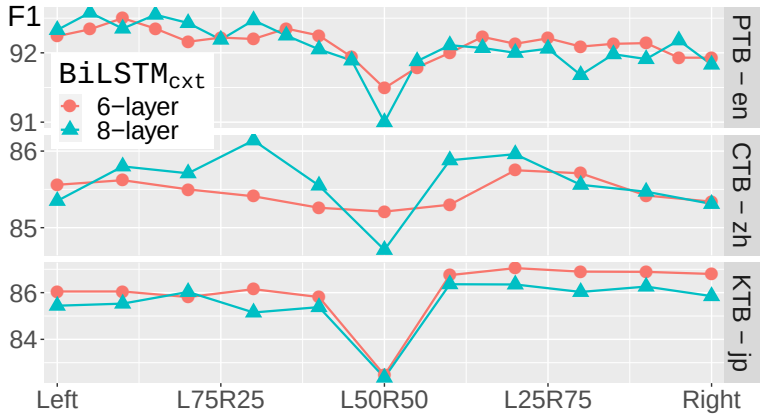


FIGURE 4.2: Probabilistic interpolations of two CNF factors to F1 scores. The capacity of $\texttt{BiLSTM}_{cxt}$ is almost saturated with 6 or 8 layers.

PWE and PLM Contextualization. I compared the results using frozen fastText with those using frozen XLNet in Table 4.3. The accuracy of the model increased along with the depth of $\texttt{BiLSTM}_{cxt}$, and it exhibited the most significant increase across all variants. XLNet tokenizes words into sub-word fractions. For the frozen XLNet, using leftmost, rightmost, or averaged sub-word embeddings as the word input yielded similar results.

Tree-Binarization Strategy

To reflect the branching tendency, the best single model for PTB was obtained on the dynamic L95R05 dataset. This dataset is a probabilistic interpolation between the left-factored dataset (for 95% chances) and a right-factored dataset (for α_{right} = 5% chances) in Figure 4.2. The best model for CTB appeared on the left side at L70R30, scoring 86.14, whereas the best for KTB was on the L30R70 dataset, scoring 87.05 with a 6-layer $\texttt{BiLSTM}_{cxt}$. Typically, the results for all the corpora had a minimum at L50R50. For English, the left "wing" was higher than the right; the opposite trend was observed for Japanese. For Chinese, no clear trend was obtained.

All studies described in the previous sections were conducted on the PTB L85R15 dataset.

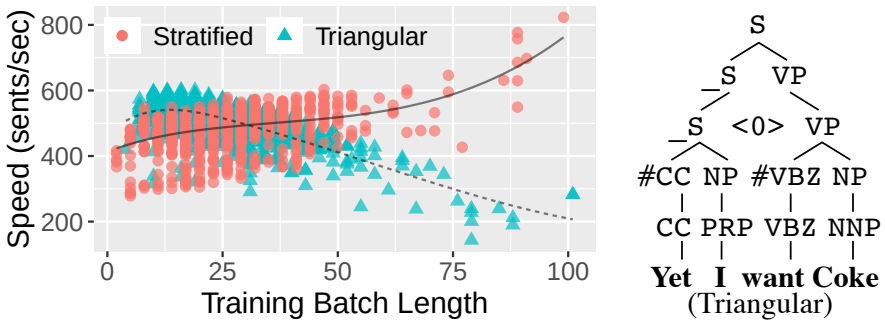


FIGURE 4.3: Linear complexity vs. squared complexity. Redundancy with placeholder "<0>" helps maintain the triangular shape.

Complexity and Speed

Format	Time/150	Memory	OOM
Stratified	7.5 hours	3.3 GB	-
Triangular	15.9 hours	8.2 GB	100

TABLE 4.5: Training time and memory consumed by my two data formats. The time column indicates the time used for 150 training epochs with validations. Development F1 scores are approximately 92.4. The OOM column lists the length limit for preventing an out-of-memory error. Kitaev and Klein (2018) took 10 hours for 93 training epochs on GeForce GTX 1080 Ti to yield their results.

To examine the empirical linear speed advantage, I inflated the training data with redundant nodes to resemble the triangular chart of CYK algorithm, as depicted in Figure 4.3 and Table 4.5. The parse in the triangular inflation is the worst-case complexity of $O(n^2)$. Meanwhile, training with linearity halved the training time, reduced memory usage, and canceled the length limit for the three treebanks. There is a sheer difference between linearity and squared complexity.

4.2.2 Structured Sentiment Analysis

Specific settings. I follow the default Stanford Sentiment Treebank (SST) data split as my training and testing corpus for **CB**. Optionally, I jointly train it with PTB to check the compatibility of the syntactic and semantic treebanks. The model is similar to the multilingual model introduced in 3.4.6, except that $\texttt{FFNN}^{\text{SST}}_{label}$ has an independent hidden layer that produce sentiment labels in $\{0,1,2,3,4\}$.

SST has sentiment polarity overturns from positive to negative or vise versa, including negation and other semantic accounts. For example,

- "*Hilariously*" as `(3)`, "*inept*" as `(1)`, "*and* as "`(2)`, "*ridiculous*" as `(1)`
- "*inept and*" as `(1)`
- "*inept and ridiculous*" as `(1)`
- "*Hilariously inept and ridiculous*" as `(0)`

where $\{0,1\}$ are negative, 2 is neutral, and $\{3,4\}$ are positive. Because a parent node's sentiment depend on those of its children, the tree structure is expected to

Model	**binary**		**5-class**	
without pre-trained language model	**All**	**Root**	**All**	**Root**
RNTN (Socher et al., 2013b)	87.6	85.4	**80.7**	45.7
BiLSTM$_{dim=300}$ (Barnes, Klinger, and Walde, 2017)	-	82.6	-	45.6
Tree-LSTM$_{dim=300}$ (Tai, Socher, and Manning, 2015)	-	88.0	-	**51.0**
PWE **CB**$_{dim=300}$ with 0-layer BiLSTM$_{cxt}$	88.9	83.4	80.2	42.8
PWE **CB**$_{dim=300}$ with 1-layer BiLSTM$_{cxt}$	89.3	85.5	79.8	47.8
PWE **CB**$_{dim=300}$ with 6-layer BiLSTM$_{cxt}$	**89.9**	**86.0**	**80.7**	48.9
with pre-trained language model				
XLNet$_{\text{LARGE}}$ (Yang et al., 2019)	-	**94.4**	-	-
RoBERTa$_{\text{LARGE}}$ (Sun et al., 2020)	-	-	-	**59.1**
BERT$_{\text{BASE}}$ (Munikar, Shakya, and Shrestha, 2019)	94.0	91.2	**83.9**	53.2
PLM **CB**$_{dim=300}$ with XLNet$_{\text{BASE}}$	**94.3**	93.7	82.6	56.2

TABLE 4.6: Models on under weakly comparable conditions. All models are trained or fine-tuned only on the Stanford Sentiment Treebank without joint training on other treebanks. All **CB** models are trained with no structural loss (i.e., $\gamma = 1$).

be better capture the sentiment for the root node (i.e., the sentence's sentiment) in a bottom-up style. Thus, loss function is

$$L^{\text{SST}}_{\text{CB}} = \gamma \cdot L^{\text{SST}}_{label} + (1-\gamma) \cdot L^{\text{SST}}_{ori} \,,$$

where there is $\gamma \in (0, 1]$.

Note that syntactic approaches (e.g., RNTN and ours in Table 4.6) tested with SST follow the original syntactic structures. RNTN cannot predict the structures whereas ours can be trained to predict the structures. However, using the original syntactic structures causes some troubles as shown in the next two sections.

Single Task Result

PWE and PLM **CB** variants trained with $\gamma = 1$ are listed in Table 4.6 along with recent strong sentiment models. There is a significant gap between models that adopt PLM or not. Beside the accuracy, RNTN as well as **CB** models can cover all four accuracy categories for a tree in one pass and leverage structural vector compositionality following the original syntactic structures. Meanwhile, BERT$_{\text{BASE}}$ (Munikar, Shakya, and Shrestha, 2019) also managed but in several passes. Sequential BiLSTM and bare PLM models (i.e., BERT, RoBERTa, and XLNet) can not leverage the original syntactic structures and thus provide only accuracy for roots at phrase or sentence level.

Joint Task Result

As an ablation study on the semantic task, I examine $0 \leq \gamma < 1$ on both PWE and PLM **CB** models, as shown in Table 4.7. Moreover, joint training with PTB is also tested whose results are reported.

I report that the accuracy of **CB** decrease when the orientation loss L^{SST}_{ori} are included in the overall loss function. These syntactic structures as an additional loss item seems not to cooperate well with the semantic labeling part. Furthermore, joint task with syntactic PTB also shows the similar results, especially with PWE models. Beside the promising results from individual tasks on PTB and SST, I do not observe

Model	$L_{\text{CB}}^{\text{SST}}$	binary		5-class		PTB
without pre-trained language model	λ	**All**	**Root**	**All**	**Root**	**F1**
PWE **CB** (only w/ SST)	0.1	86.3	79.9	78.6	41.9	-
PWE **CB** (only w/ SST)	0.5	87.8	82.0	79.6	45.1	-
PWE **CB** (only w/ SST)	0.9	88.7	85.4	80.2	47.2	-
PWE **CB** (only w/ SST)	0.99	88.7	85.3	80.4	48.4	-
PWE **CB** (only w/ SST)	1	**89.9**	**86.0**	**80.7**	**48.9**	-
PWE **CB** (w/ SST & PTB)	0.5	87.8	84.1	79.2	44.7	91.7
PWE **CB** (w/ SST & PTB)	1	88.1	84.4	79.2	44.4	91.7
with pre-trained language model						
PLM **CB** (only w/ SST)	0.5	93.8	92.6	**82.9**	**55.4**	-
PLM **CB** (w/ SST & PTB)	0.5	92.8	91.2	82.6	51.9	95.4
PLM **CB** (only w/ SST)	1	**94.3**	**93.7**	82.6	54.2	-
PLM **CB** (w/ SST & PTB)	1	93.3	92.7	82.8	54.1	95.6

TABLE 4.7: Syntactic information seems not helping sentiment classification. Both induction of orientation loss and joint training with PTB decrease the sentiment classification accuracy. The F1 scores are 92.5 and 95.7, respectively for PWE and PLM **CB**.

mutual benefits by joint training **CB** on SST with PTB at corpus level or the semantic part with the syntactic part at structure level. See the discussion in Section 5.4.3.

4.3 Discontinuous Constituency Parsing

Two-stage training for PWE model. The first stage (**S1**) requires approximately 300 epochs with general hyperparameters. Loss functions are sum of all loss items:

$$L_{\text{DB}}^{\text{S1}} = L_{tag} + L_{label} + L_{jnt} + L_{ori} + L_{ori}^{shfl} + L_{jnt}^{shfl}$$
$$L_{\text{DM}}^{\text{S1}} = L_{tag} + L_{label} + L_{jnt} + L_{disc} + \sum_{i \in D,X,C} L_{ba}^{i}$$

Adam optimizer's learning rate is $\gamma = 10^{-3}$. **DB** uses uniform binarization $\alpha_{left} = \alpha_{right} = 1$. **DM** uses $\varnothing$-subtree ratio $\rho_{\varnothing} = 0.25$, robustness $\beta_c = 0.1$, and $\beta_x = 1$ for both efficiency and accuracy.

The second stage (**S2**) involves 100 short trials with a Bayesian optimization (BO) tool Akiba et al., 2019, *optuna*; each trial requires less than 30 epochs and brings hyperparameter adjustment:

$$L_{\text{DB}}^{\text{S2}} = \alpha_{tag} \cdot L_{tag} + \alpha_{label} \cdot L_{label} + \cdots + \alpha_{jnt}^{shfl} \cdot L_{jnt}^{shfl}$$
$$L_{\text{DM}}^{\text{S2}} = \beta_{tag} \cdot L_{tag} + \alpha_{label} \cdot L_{label} + \cdots + \sum_{i \in D,X,C} (\beta_i \cdot L_{ba}^{i})$$

Trials follow practical constraints: learning rate $\gamma \in (10^{-6}, 10^{-3})$, beta's $\alpha_{left}, \alpha_{right} \in (10^{-3}, 10^{3})$ instead of $(0, +\infty)$, and $[0, 1]$ for the others.

PLM models also use general hyperparameters with learning rate 10^{-6} at **S1**. PLMs are frozen during the first 50 epochs to avoid noise pollution and then are fine-tuned with learning rate 3×10^{-6}. They inherit explored hyperparameters from PWE models at **S2**, except for learning rate 3×10^{-6}.

4.3.1 Overall Results

Table 4.8 shows F1 scores of recent neural discontinuous parsers under comparable conditions on test sets. We follow their reported number of significant digits and reduce the effects of random initialization with an average of five runs. The details are shown in Table 4.9.

DM models achieved state-of-the-art performances in terms of discontinuous F1 scores and parsing speeds. Although speed tests are reported on different platforms, my parsers lead by a significant margin. In terms of overall F1 score, my parsers outperform some chart parsers (Stanojevic and Steedman, 2020; Ruprecht and Mörbitz, 2021) and slightly underperform the overall best outline, as characterized in **boldface**.

Model			DPTB test set			TIGER test set		
without pre-trained language model	**Type**	**Complexity**	**F1**	**D.F1**	**Speed**	**F1**	**D.F1**	**Speed**
Coavoux, Crabbé, and Cohen (2019)	Trans-Gap	$O(n)$	91.0	71.3	80	82.7	55.9	126
Coavoux and Cohen (2019)	Stack-Free	$O(n^2)$	90.9	67.3	38	82.5	55.9	64
Pointer-based VG20 w/ Ling et al. (2015)	Seq-Labeling	$O(n^2)$	88.8	45.8	611	77.5	39.5	568
Pointer-based FG22 w/ Ling et al. (2015)	Multitask†	$O(n^2)$	-	-	-	**86.6**	**62.6**	-
Stanojevic and Steedman (2020)	Chart	$O(n^6)$	90.5	67.1	-	83.4	53.5	-
Corro (2020)	Chart	$O(n^3)$	**92.9**	64.9	355	85.2	51.2	474
Ruprecht and Mörbitz (2021) w/ flair	Chart	-	91.8	76.1	86	85.1	61.0	80
DB w/ fastText (en & de)	Combinator	$O(n^2)$	92.0	75.6	**940**	84.9	60.1	**1160**
DM w/ fastText (en & de)	Combinator	$O(n^3)$	92.1	**78.1**	**970**	85.1	62.0	**1300**
with pre-trained language model								
Pointer-based VG20 w/ $\text{BERT}_{\text{BASE}}$	Seq-Labeling	$O(n^2)$	91.9	50.8	80	84.6	51.1	80
Pointer-based FG22 w/ $\text{BERT}_{\text{BASE}}$	Multitask†	$O(n^2)$	-	-	-	89.8	**71.0**	-
Corro (2020) w/ BERT	Chart	$O(n^3)$	94.8	68.9	-	**90.0**	62.1	-
Ruprecht and Mörbitz (2021) w/ BERT	Chart	-	93.3	80.5	57	88.3	69.0	60
FG21 w/ XLNet (en) or $\text{BERT}_{\text{BASE}}$ (de)	Reorder-Chart	$O(n^3)$	95.1	74.1	179	88.5	63.0	238
FG21 w/ XLNet (en) or $\text{BERT}_{\text{BASE}}$ (de)	Reorder-Trans	$O(n^2)$	**95.5**	73.4	133	88.5	62.7	157
DB w/ XLNet (en) or $\text{BERT}_{\text{BASE}}$ (de)	Combinator	$O(n^2)$	94.8	76.6	**275**	89.5	69.7	**424**
DM w/ XLNet (en) or $\text{BERT}_{\text{BASE}}$ (de)	Combinator	$O(n^3)$	95.0	**83.0**	**375**	89.6	70.9	**535**

TABLE 4.8: Overall performances from recent works. Speeds in sentences per second are reported on incomparable hardware and software platforms. Ours and Vilares and Gómez-Rodríguez, 2020, VG20 are on GeForce GTX 1080 Ti with PyTorch implementation and Fernández-González and Gómez-Rodríguez, 2021, FG21 on GeForce RTX 3090. Fernández-González and Gómez-Rodríguez, 2022, FG22 involved lexical dependency information†.

DPTB (test)		F1	D.F1
PWE	**DB**	91.97±0.05	75.62±0.82
	DM	92.06±0.10	78.14±0.69
PLM	**DB**	94.84±0.24	76.62±2.07
	DM	95.04±0.06	83.04±0.79
TIGER (test)		**F1**	**D.F1**
PWE	**DB**	84.88±0.08	60.08±0.37
	DM	85.11±0.13	62.02±0.71
PLM	**DB**	89.48±0.16	69.68±0.55
	DM	89.61±0.09	70.93±0.63

TABLE 4.9: Means and standard deviations of five runs on test sets with four significant digits. **DM** ourperforms **DB**. Development sets reflect similar variability.

Model (Stage)		Development set DPTB F1	DPTB D.F1	TIGER F1	TIGER D.F1	Test set DPTB F1	DPTB D.F1	TIGER F1	TIGER D.F1
DB	(0,0,0)	90.93	63.28	87.73	56.49	90.82	62.48	82.29	48.90
	(0,1,1)	91.61	69.84	88.70	61.15	91.69	71.60	83.56	54.32
	(1,0,1)	91.62	**74.25**	87.93	59.85	91.26	69.12	82.86	54.33
	(1,1,0)	91.48	70.97	89.05	63.32	91.65	70.18	84.32	56.45
(**S1**)	‡(1,1,1)	**91.72**	66.82	**89.28**	**63.49**	**91.76**	**73.49**	**84.53**	**59.68**
(**S2**)	↪*optuna*	92.25	76.60	89.59	66.03	92.08	74.30	84.56	59.64
DM	(0,0,0)	91.62	79.37	88.30	62.41	91.26	71.52	83.15	55.81
	(0,1,1)	91.74	79.02	89.64	67.40	91.51	**78.86**	84.89	61.02
	(1,0,1)	91.44	78.70	88.61	65.10	91.16	74.38	83.79	58.30
	(1,1,0)	91.84	77.37	**89.78**	67.78	91.81	78.53	85.06	**62.15**
(**S1**)	‡(1,1,1)	**92.16**	**80.29**	89.77	**68.20**	**92.07**	77.39	**85.17**	62.02
(**S2**)	↪*optuna*	92.37	82.76	89.84	68.45	92.13	77.99	85.20	62.15

TABLE 4.10: Ablation in two-stage training with dev scores. Triplets in $\{0,1\}$ stand for turning on and off ($\rho_\varnothing$, $\rho^{\mathbf{B}} \sim \text{Beta}(1,1)$, *shuffle*) for **DB** and ($\rho_\varnothing$, β_c, β_x) for **DM**. Variants of "‡" are **S1** — the start of **S2**.

Model	Dev set	$\rho_\varnothing = 0$	0.1	‡0.25	0.5
DB	**DPTB**	91.61	91.79	91.72	**91.95**
	TIGER	88.70	89.04	**89.28**	89.25
DM	**DPTB**	91.44	91.80	**92.16**	89.86
	TIGER	88.61	89.45	**89.77**	88.61

TABLE 4.11: **DM** is sensitive to $\rho_\varnothing$ with dev F1 scores.

Test data	DM Medoid $\rho^{\mathbf{M}}$	F1	D.F1
DPTB	*uhead*	**95.05**	**83.58**
	leftmost	95.00	81.64
	rightmost	95.03	82.47
	random (min)	95.01	82.18
	random (max)	95.04	83.17
TIGER	*uhead*	**89.62**	**71.61**
	leftmost	89.56	71.43
	rightmost	89.56	70.92
	random (min)	89.55	71.26
	random (max)	89.61	71.52

TABLE 4.12: **DM** medoid factor $\rho^{\mathbf{DM}}$ = *uhead* offers stable gains even without head information during training. I tested $\rho^{\mathbf{DM}}$ = *random* five times.

4.3.2 Ablation Study

I ablate the PWE models in two-stage training, as shown in Table 4.10. Only one representative run with ablation is shown because of the similar low variability on development sets. **DB** has two data augmentation items $\rho_\varnothing$ and $\rho^{\mathbf{B}}$ as well as one model item *ply shuffle*. On $\rho_\varnothing$ refers to $\rho_\varnothing = 0.25$ and off $\rho_\varnothing = 0$. Off Beta(1,1) refers to a static $\rho^{\mathbf{DB}} = 0.5$ and (0,0,0) shows the performances of bare **DB** models.

On the flip side, **DM**'s (0,0,0) contains randomness because of $\rho^{\mathbf{M}}$ = *random*. We do not intend to examine a static $\rho^{\mathbf{DM}}$ as **DB** yields the negative results. Based on effective training tricks, the variants enter the BO process at **S2**. **DM** shows its sensitivity to $\rho_\varnothing$ in Table 4.11.

Model	Type	PTB	CTB	KTB
PWE **CB**	**Monolingual**	92.54	86.14	87.05
	Multilingual	92.32	86.05	87.57
PWE **CM**	**Monolingual**	92.11	85.33	87.46
	Multilingual	91.84	85.23	86.72
PWE **CM** (WS for CTB & KTB)	**Monolingual**	92.11	83.37	87.81
	Multilingual	91.86	83.94	87.63

TABLE 4.13: F1 scores of monolingual and multilingual continuous **NCCP** parsers. For CTB and KTB, **CM** extends its chunking function for word segmentation during training and **CM** is character-based for Chinese and Japanese but word-based for English.

		DPTB		TIGER	
Model	**Type**	**F1**	**D.F1**	**F1**	**D.F1**
PWE **DB**	**Monolingual**	91.76	73.49	84.53	59.68
	Multilingual	92.07	73.86	84.58	58.83
PWE **DM**	**Monolingual**	92.07	77.39	85.17	62.02
	Multilingual	92.09	79.45	84.77	59.89

TABLE 4.14: F1 scores of monolingual and multilingual discontinuous **NCCP** parsers.

4.3.3 Inference with Unsupervised Headedness

Both **CM** and **DM** provide unsupervised headedness $\bar{\lambda}$. Chen et al. (2021) were unable to test the benefits of **CM**'s unsupervised headedness because it is a final product that cannot affect parsing. However, **DM**'s medoid affects parsing performance. On PLM **DM**, we select different $\rho^{\mathbf{M}}$ categories, which affect the location of all discontinuous constituent, and examine their generalization on test sets, as shown in Table 4.12. All models are trained with $\rho^{\mathbf{M}}$ = *random* but inference with $\rho^{\mathbf{M}}$ = *uhead* exerts positive gains on accuracy.

4.4 Multilingualism and Word Segmentation for Chinese and Japanese

Finally, I merge all models into two multilingual parsers respectively for continuous parsing and discontinuous parsing. The method is described in Section 3.4.6. Thus, the continuous parser is trained on PTB, CTB, and KTB; the discontinuous parser is trained on DPTB and TIGER. I show their comparable results in Tables 4.13 & 4.14.

The only difference of a monolingual parser and a multilingual PWE parsers lies in $\texttt{EMBEDDING}^k$, $\texttt{FFNN}^k_{tag}$, and $\texttt{FFNN}^k_{label}$. Apart from well-studied fastText $\texttt{EMBEDDING}^k$, inspecting $\texttt{FFNN}^k_{tag}$ and $\texttt{FFNN}^k_{label}$ to different k is discussed in Section 5.1.4.

Because Chinese and Japanese do not use white space for word segmentation, a deliberate additional word segmentation process is very common for parsing the sentences in these two languages. Extending chunker-based parsing to word segmentation is a very natural option for my **CM** model. I assign the first ply to be a word segmentation layer and train **CM** for the sub-task. At inference phase, I use the gold word segmentation for F1 scores. The results are in the bottom of Table 4.13. See the discussion in Section 5.1.4.

Chapter 5

Discussion

5.1 Feature of NCCP Family

5.1.1 Compact Neural Combinator

NCCP parsers are neural combinators. Each comprises not only a neural encoder for scoring (i.e., Algorithm 1) but also a simple non-neural decoder for decision and removal of ∅ nodes. The decoder is a symbolic extension of the encoder in that both run in bottom-up manner, and the decoder interprets the scores as local-and-greedy decisions. Other neural parsers also fit a similar encoder–decoder framework. However, decoders with dynamic programming often include forward and backward processes heterogeneous to their forward encoders (Kitaev and Klein, 2018; Kitaev and Klein, 2020). The encoder and decoder in my model and (Shen et al., 2018b) are more homogeneous and can be easily merged. My parsers are bottom-up combinatory, while theirs was top-down splitting. Similar homogeneity can be found in an easy-first dependency parser (Goldberg and Elhadad, 2010).

These parsers are also beyond pure neural combinators, such as RNTN by Socher et al. (2013a) and Tree-LSTM by Tai, Socher, and Manning (2015). Their models do not include a structural prediction components and rely on external structure providers. The **CB** model in Figure 4.1 indicate that the structural orientation is a crucial factor for parsing accuracy.

$\texttt{BiLSTM}_{cxt}$	**+CB**	**+CM**	**+DB**	**+DM**
3.25M	0.36M	0.55M	1.32M	1.45M

TABLE 5.1: Parameter sizes of PWE **NCCP** parsers.

Although vector composition can be as complex as cubic tensor operation (Socher et al., 2013b), neural combinators often do not have a very large model dimension like 768 or 1024 of pre-trained languages models (Devlin et al., 2019; Yang et al., 2019). Each composition is limited to a fixed time and memory size by the model dimension. Thus, my parsers can be fast and efficient with vector compositionality using a medium 300 dimension. From the results of multilingualism in Section 4.4, I know that sharing the structural parsing backend in Algorithm 4 does not significantly affect parsing accuracy in each language. Thus, I suppose my models do not store too much language-specific knowledge in their parameters. Instead, I exploit external language-specific knowledge in pre-trained word embeddings and languages models by using vector compositionality. This makes my parsers concise. Apart from the external knowledge, **NCCP** has no more than 5M parameters, as shown in Table 5.1. Meanwhile, the model of Shen et al. (2018b) has more than 20M.

CNF	Left-factoring $\rho^{B} = 0$		Mid-factoring $\rho^{B} = 0.5$		Right-factoring $\rho^{B} = 1$	
Ori.	Left	Right	Left	Right	Left	Right
PTB	3.8M	4.4M	3.0M	5.3M	2.3M	6.5M
CTB	2.5M	1.7M	1.9M	2.2M	1.4M	2.8M
KTB	4.5M	0.9M	2.8M	1.7M	1.8M	2.1M

TABLE 5.2: Frequencies of orientation with different CNF (biased) and referential only non-CNF (more balanced) $\rho^{B} = 0.5$ in different stratified continuous treebanks for **CB**.

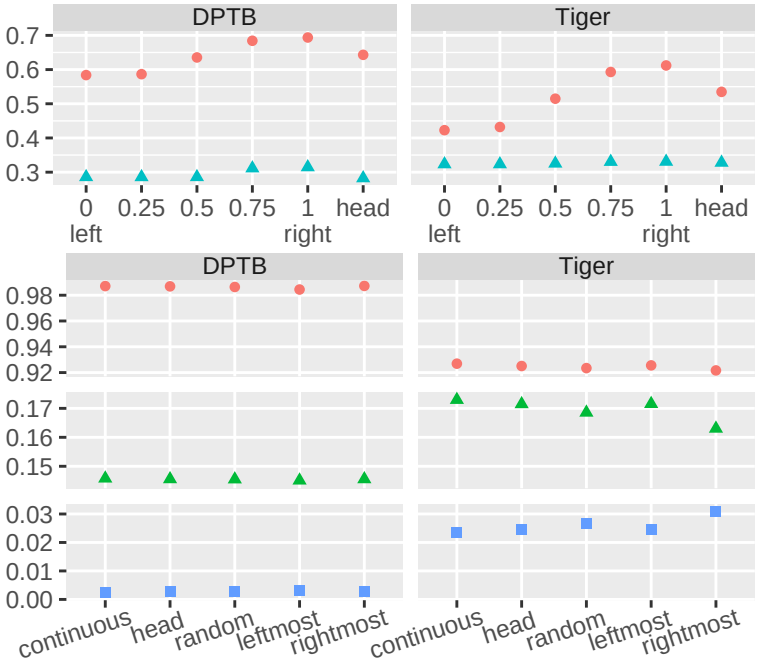


FIGURE 5.1: Top: **DB** signal polarity to ρ^{B} with *orientation right* "•" and *joint* "▲". Bottom: **DM** signal polarity to stratifying *medoid* factor ρ^{M} with *affinity* "•", *joint* "▲", and *discontinuity* "■". *Continuous* and *head* are referential only, looking for the least *discontinuity* and leveraging head information. (All $\rho_{\varnothing} = 0$.)

All models contain compact components without grammar restriction. By keeping the model simple and of small dimension, **NCCP** achieves the highest parsing speeds for both continuous and discontinuous constituency parsing tasks. Besides, it reflects linguistic properties with other features based on neural combinator.

Variability of discontinuous models are examined to be small except for the discontinuous F1 score of PLM **DB** on DPTB, as shown in Table 4.9. The main cause may not be the random initialization but the different training processes of PWE and PLM models: PLM models use the configuration of PWE models at **S2**. Those degraded PLM models adopted low $\rho_{\varnothing}$ configurations (e.g., $\rho_{\varnothing} = 0.078$). Because DPTB has less discontinuity and overall F1 scores is used for model evaluation, high variability in discontinuous F1 scores become more common without several BO trials, which are also reflected by Table 4.10. **DB** lacks explicit discontinuity and the selection of hyperparameter seems to be necessary on DPTB.

5.1.2 CB & DB Orientation: Branching Tendency

On one hand, I show orientation in binarized PTB for English, CTB for Chinese, and KTB for Japanese to present the syntactic branching tendencies in Table 5.2. As

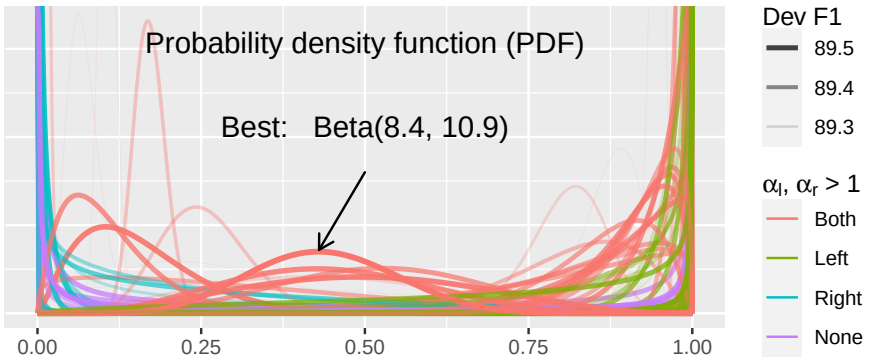


FIGURE 5.2: Beta distribution visualization for TIGER **DB** at **S2**. The line thickness corresponds to the development F1 score. See their other hyperparameters in Figure 5.6.

English is a right-branching language, its majority orientation is to the right. Even left-factoring cannot reverse the trend, but it should create a greater balance. The reverse tendency emerges in the KTB corpus as Japanese is a left-branching language. For Chinese, CTB does not exhibit a clear branching tendency.

As a result, both **CB** and **DB** are affected by the innate biases in the corpora. For **CB**, Bernoulli interpolation with two CNF-factored datasets is effective for the three languages studied, as shown in Figure 4.2. Dynamic sampling allows the model to cover a wider range of composed vectors to improve its robustness to ambiguous orientations. Furthermore, it seems counterintuitive for human learners to obtain the best model using left-biased interpolation for a right-branching language or vice versa. However, for a neural model, balancing the frequency seems to be the key factor for improving performance (Sennrich, Haddow, and Birch, 2016b; Zhao et al., 2018). The fact that the L50R50 dataset yielded the worst models also suggests that the balance should be based on a default orientation tendency.

Similarly, **DB**'s **S2** observes slightly leftward exploration with the beta distribution, as shown in Figure 5.2. DPTB has a similar situation. Yet, the optimized distributions are relatively uniform and symmetric, which qualifies my uniform randomness with $\rho^{\mathbf{B}}$ at **S1** and indicates a desired property for future language-agnostic practice.

5.1.3 CM & DM Unsupervised Weight: Headedness

On the other hand, the multi-branching models dynamically place close attention on what the syntactic head is supposed to be. Table 5.3 provides more statistical support. In the noun phrases, determiners receive the highest weight averages, and the nouns obtain the second. This phenomenon suggests that an English noun phrase's syntactic role is mainly projected from the determiners, as discussed by Zwicky (1985). **CM** selects `DT` as an `NP` head if it is available; otherwise, nouns and adjectives are prominent heads. Chinese and Japanese parsers work similarly for their headedness. Some multi-branching parsing instances with the weights are exhibited in Section 5.4.

DB's medoid $\rho^{\mathbf{M}}$ = *uhead* further stably benefits the parsing accuracy, as shown in Table 4.12. In other words, the place of a (discontinuous) phrase in a ply matters to some extends. I suppose that the reason lies in that $\texttt{BiLSTM}_{ply}$ can better encode the information for nodes in the ply. A head child that has a significant place can better inform other siblings or other relevant nodes.

Parent (#)	Head child by maximum weight
NP (14.4K)	DT (4.5K); *NP (4.3K); *NNP (1.6K); *JJ (922); *NN (751); *NNS (616); etc. (1.6K; 38 of 50 types with "*")
VP (6.8K)	VBD (1.5K); VB (1.4K); VBZ (1.0K); VBN (954); VBP (705); MD (523); VBG (387); VP (169); TO (81); etc.
PP (5.5K)	IN (5.0K); TO (397); etc.
S (3.8K)	VP (3.4K); S (194); NP (90); etc.
SBAR (1.2K)	IN (649); WHNP (395); WHPP (19); WHADVP (121); SBAR (15); etc.
ADVP (278)	RB (181); IN (30); RBR (25); etc.
QP (198)	CD (67); IN (65); RB (29); JJR (16);

TABLE 5.3: English headedness selection with my multi-branching model **CM** on PTB test set. **DM** also provide very similar statistics. "*" marks the absence of a DT child for its NP sisters. For quantifier phrases (QP), some non-quantifiers are more likely to be heads if they appear; e.g., adverbs (RB; e.g., "approximately"), prepositions (IN; e.g., "about"), and relative adjectives (JJR; e.g., "more than").

5.1.4 Multilingualism

As shown in Tables 4.13 & 4.14, training multilingual parsers slightly affects the performances comparing to their monolingual versions. The effect is not a global decrease across all F1 scores. It is observed that **CM** benefits from multilingualism and word segmentation extension on KTB. However, CTB does not enjoy multilingualism and word segmenation. I guess that the word segmentation and syntactic annotation schemes are quite different on CTB and KTB. Although CTB have fewer training samples than KTB, the shared parameters does not works the best for Chinese word segmentation and constituency parsing.

Both **DB** and **DM** benefits from multilingualism on DPTB instead of TIGER. It seems that, by sharing model parameters, the performance of DPTB borrowed something from TIGER, especially the discontinuous accuracy. As DPTB has less discontinuity than TIGER does, the multilingual **DB** and **DM** "average" the performances of individual corpora.

I further check the relationship between different $\texttt{FFNN}^{k}_{label}$ vector pairs with $k \in \{\text{PTB}, \text{CTB}, \text{KTB}\}$ for the continuous multilingual parser or $k \in \{\text{DPTB}, \text{TIGER}\}$ for the discontinuous multilingual parser. I use each parameter vector for label l in the final layer $\texttt{FFNN}^{k}_{label:l}$ of k to calculate Euclidean distance,

$$\text{distance}^{k_{lhs}:l_{rhs}}_{k_{lhs}:l_{rhs}} = \left\| \texttt{FFNN}^{k_{lhs}}_{label:l_{lhs}} - \texttt{FFNN}^{k_{rhs}}_{label:l_{rhs}} \right\| .$$

For example, the distance of two NPs in PTB and CTB is $\left\| \texttt{FFNN}^{\text{PTB}}_{label,\,\texttt{NP}} - \texttt{FFNN}^{\text{CTB}}_{label:\texttt{NP}} \right\|$. Then, I sort the the labels in treebank k_{rhs} to a label l_{lhs} in treebank k_{lhs} and create Table 5.4.

As can be seen, two treebanks works well for pairs PTB × CTB and TIGER × DPTB. KTB labels do not align well with the other treebanks probably because of its annotation scheme or linguistic category. Furthermore, Japanese is a head-final agglutinative language, different from other language we used. Meanwhile, English and German are inflectional languages and share a part of vocabulary and grammars. Although Chinese is an isolating language, CTB's annotating scheme is very similar to PTB's.

CTB and KTB do not have an S symbol. Instead they have an IP (inflectional phrase) for a complete sentence. I can find **IP** and IP+VP in CTB are close to **S** and

$k_{lhs} \times k_{rhs}$	l_{lhs}	l_{rhs} (sorted by increasing distance from left to right.)
PTB × CTB	NP	NP, NP+CP+IP, PP+NP, VP+LCP, NP+CP+IP+VP, NP+QP, ...
	VP	VP, VP+NP, VP+LCP, VP+VNV, PRN+IP+VP, VP+ADJP, ...
	PP	PP, IP+VP+PP, PRN+VP, NP+CP+IP+VP, IP+VP+NP, ...
	S	IP, NP+CP+IP, CP+IP, PRN+IP, IP+VP+NP, VP+IP, ...
	S+VP	IP+VP NP+CP+IP+VP CP+IP+VP IP+VP+VSB PRN+VP, ...
PTB × KTB	NP	PRN+IP, PNLP, PP+NP+NUMCLP, LST+NUMCLP, FRAG+ADVP, ...
	VP	PP+NP, IP+NP+NUMCLP, CONJP+NP+NUMCLP, PRN+IP, ...
	PP	NP+PP, CONJP+CP, PP+NP+NUMCLP, IP+PP, P, ...
	ADVP	NP+PP, IP+NP+NUMCLP, PP+NP+NUMCLP, CONJP+CP, ...
	S	PP+NP, CP+IP, NP+IP, IP+NP+NUMCLP, PRN+IP, FRAG+IP
CTB × KTB	NP	FRAG+ADVP, PRN+IP, FS+NP+NUMCLP, PP+NP+NUMCLP, ...
	VP	IP+NP+NUMCLP, CONJP+NP+NUMCLP, PRN+IP, ...
	ADVP	CONJP+CP, FS+NP+NUMCLP, CP+INTJP, FRAG+ADVP, ...
	IP	CP+IP, FRAG+IP, NP+IP, PRN+IP, multi-sentence, ...
	PP	CONJP+CP, FS+NP, IP+CP, META, FS+IP, FS+NUMCLP, ...
TIGER × DPTB	NP	NP, S+ADVP, X+NP, VP+ADVP, NP+S+VP, NP+ADJP, ...
	PP	PP, X, FRAG+PP, S+UCP, S+PP, PP+PP, X+NP, ...
	S	S, FRAG, RRC, FRAG+ADJP, S+PP, FRAG+ADVP, FRAG+S, ...
	VP	RRC, FRAG+PP, S+ADVP, LST, S+UCP, PRN+S, VP+ADVP, ...
	VROOT	FRAG, FRAG+NP, FRAG+ADJP, SBARQ, FRAG+ADVP, S+UCP, ...

TABLE 5.4: Similar labels in different treebanks by their Euclidean distances in multilingual parsers' $\texttt{FFNN}^{k}_{label}$.

S+VP in PTB. Further, KTB does not have a VP symbol (Hinds, 1973) and I did not observe strong alignment with labels in other treebanks.

TIGER's VP is close to PTB's RRC (reduced relative clause) and FRAG+PP (fragment which yields a preposition phrase). According to the PTB's guidelines (Bies et al., 1995), *the RRC label is used only in cases where there is no VP and an extra level is needed for proper attachment*. TIGER's VROOT is a dummy root label which can include incomplete fragments. In these subtle senses, TIGER and PTB's labels are aligned.

5.1.5 Beyond Constituency Parsing

Beside multilingual constituency parsing, I exhibited the versatility of **NCCP** through sentiment analysis by utilizing its neural combinator. There are three observations: 1) **NCCP** can be a strong model for semantic tasks. Its performance is near state-of-the-art. The speed advantage is still a bright side. However, 2) imposing the syntactic loss decreases the semantic performance. 3) joint training with syntactic treebank also leads to degraded semantic performance.

In fact, I also tested **NCCP** to another task: named entity recognition. However, unlike the competitive results on SST, the results fell far behind the state-of-the-art models on CoNLL 2003 English NER. Specifically, I used an 1-ply **CM** as a shallow parsing chunker with $\texttt{FFNN}_{label}$ for NER labels and the F1 score was around 83 with PWE. Recent NER models' performance frequently show F1 scores above 92.

For these results, I suppose two reasons account the gap: 1) different syntactic annotation schemes and 2) the natural difference between syntactic and semantic tasks. To be concrete, the syntactic structures in SST annotated automatically by a syntactic parser and are imposed in a binary tree format. (See the example in Section 5.4.3.) On the other hand, CoNLL provides both NER chunks and syntactic

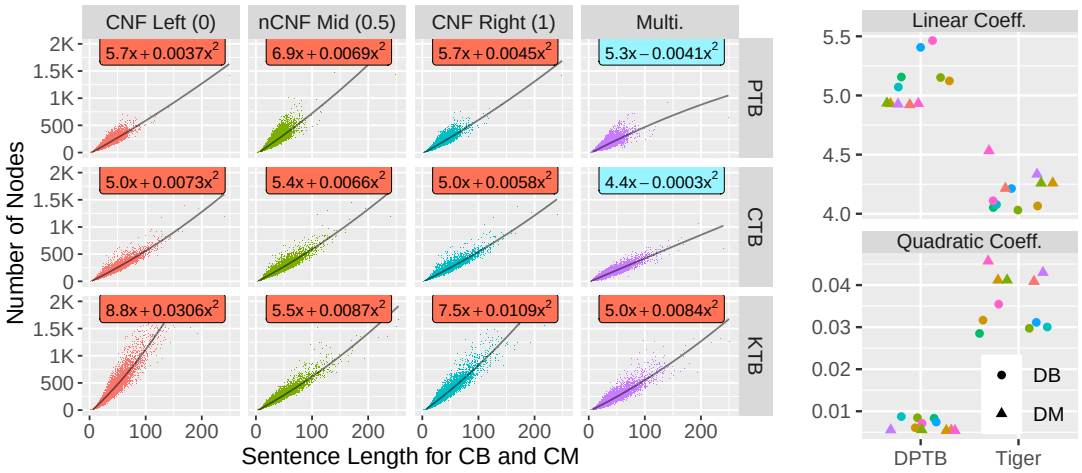


FIGURE 5.3: Empirical complexities of **NCCP** parsers for each corpus with linear regressions (LR) are shown on a light blue background when the quadratic terms are negative for **CB** and **CM** (left) or direct linear and quadratic coefficient scatter for **DB** and **DM** (right). For DPTB and TIGER, $\rho^{\mathrm{B}} \in \{0, 0.25, 0.5, 0.75, 1\}$ and $\rho^{\mathrm{M}} \in \{\textit{continuous}, \textit{head}, \textit{random}, \textit{leftmost}, \textit{rightmost}\}$ are used for the scatter. Cubic LR gives all negative cubic terms highly close to zero.

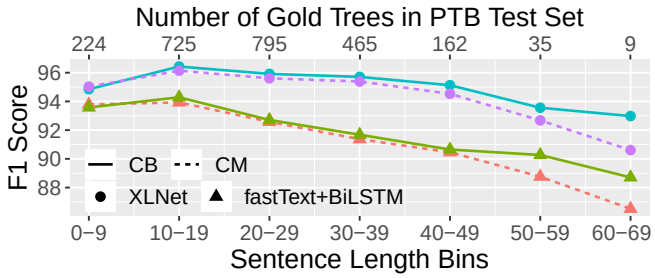


FIGURE 5.4: XLNet provides an overall improvement for **CB** and **CM** on length bins. All models find it challenging to handle long sentences.

spans. However, NER chunks and phrase spans are not always in relations of either containing or being contained, creating some ambiguities.

5.2 Empirical Complexity and Speed

Extreme cases indicates that **NCCP**'s theoretical complexities are either $O(n^2)$ or $O(n^3)$. Specifically, **CB** and **CM** is bounded by **DB** and **DM** because *swap* and *biaffine attention* create additional computation. However, **NCCP** has an empirical $O(n^2)$ complexity with strong linearity, as shown in Figure 5.3. **DB** has higher linear coefficients because of its slow binary combination. Meanwhile, **DM** shows stronger quadratic tendency because of *biaffine attention*. On the flip side, **CM** has some negative quadratic coefficients for PTB and CTB. Yet, their coefficient magnitudes are on par with one another.

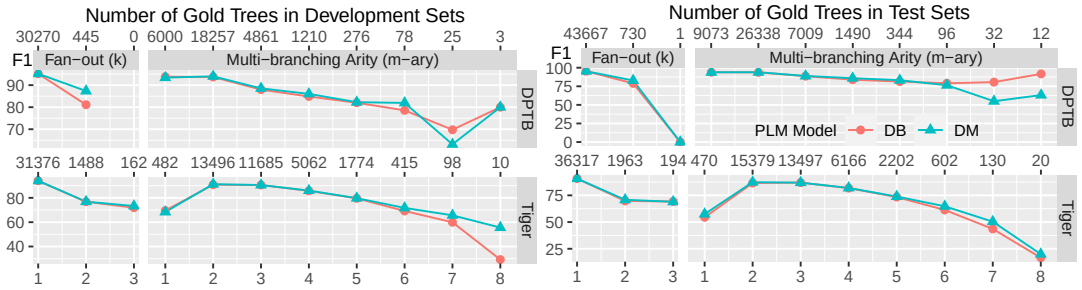


FIGURE 5.5: **DB** and **DM**'s discontinuity and multi-branching performances. **DM** keeps some points ahead in terms of F1 scores on TIGER richer in discontinuity.

Prop.	Gold	PWE		$\rho_\varnothing = 0$		$\rho_\varnothing > 0$		PLM		$\rho_\varnothing = 0$		$\rho_\varnothing > 0$	
M-ary	Tree	CB	CM	DB	DM	DB	DM	CB	CM	DB	DM	DB	DM
1	9,073	**92.25**	92.02	**91.43**	91.35	**91.68**	91.53	93.80	**94.33**	93.36	**93.41**	**93.91**	93.82
2	26,338	**90.41**	89.94	**89.87**	89.68	89.95	**90.02**	**94.41**	94.33	93.47	**93.65**	93.77	**93.92**
3	7,009	**84.17**	83.56	**83.50**	83.34	83.60	**83.79**	**90.27**	89.81	88.31	**88.60**	88.57	**88.87**
4	1,490	77.87	**78.95**	78.19	**79.82**	78.50	**78.86**	**87.42**	86.51	83.98	**86.50**	83.88	**85.90**
5	344	74.19	**77.29**	76.14	**78.46**	74.97	**78.87**	81.42	**84.06**	78.61	**85.15**	81.38	**83.17**
6	96	70.05	**78.35**	72.90	**80.63**	**77.39**	76.29	78.64	**80.00**	79.23	**83.50**	**79.02**	76.44
7	32	64.71	**86.15**	73.53	**87.10**	**76.47**	71.43	**76.47**	70.18	75.76	**77.42**	**80.60**	54.90
8	12	64.00	**72.73**	81.82	**85.71**	75.00	**80.00**	78.26	**86.96**	78.57	**83.33**	**91.67**	63.16
9	3	**100.00**	75.00	**100.00**	75.00	**100.00**	50.00	**100.00**	75.00	**100.00**	85.71	85.71	**100.00**
$k > 1$	731	-	-	73.95	**77.60**	75.68	**78.94**	-	-	78.62	**83.04**	78.65	**82.71**
All	44,397	**92.54**	92.08	91.99	**92.02**	92.00	**92.00**	**95.71**	95.44	94.70	**94.79**	95.08	**95.09**

TABLE 5.5: Multi-branching and discontinuous F1 scores of **DB** and **DM** on (D)PTB test sets. We grouped $k > 1$ because only one tree has fan-out $k = 2$ in the test set. The scores of **CB** and **CM** are from Chen et al. (2021).

5.3 Accuracy and Robustness

5.3.1 Contextualization and Length

For the continuous models **CB** & **CM**, better contextualization with PLM offers great improvement for most sentences, especially for longer sentences, as suggested in Figure 5.4. $\texttt{BiLSTM}_{cxt}$ also plays an important role for increasing my PWE parsers' accuracy as demonstrated in 4.3. Beside the contextualization for the input terminals, contextualization for the iterative plies are also crucial.

In Table 4.4, I examined that using the performance of $\texttt{FFNN}_{ply}$ is significantly inferior to that of its RNN counterparts, suggesting that some information might not be encoded locally. Thus, the `COMPOSE` function should remain somehow in a contextualized form to collaboratively leverage the whole ply. However, using BiRNN (e.g., $\texttt{BiQRNN}_{ply}$ and $\texttt{BiLSTM}_{ply}$) might still be a bottleneck for long-range dependency in a ply.

DB & **DM** show similar trends for the relationship between sentence length and parsing accuracy. In the next subsection, I show other perspectives of **DB** & **DM**'s accuracies.

Prop.	Gold	PWE	$\rho_\varnothing = 0$	$\rho_\varnothing > 0$		PLM	$\rho_\varnothing = 0$	$\rho_\varnothing > 0$	
M-ary	Tree	DB	DM	DB	DM	DB	DM	DB	DM
1	470	45.37	**49.14**	51.64	**55.32**	55.16	**56.84**	54.45	**57.48**
2	15,379	81.83	**82.57**	82.25	**83.36**	85.91	**86.34**	86.50	**87.31**
3	13,497	**80.58**	80.41	80.95	**81.05**	**85.96**	85.35	**86.93**	86.89
4	6,166	**73.43**	73.34	74.04	**74.36**	**80.71**	79.93	81.76	**81.92**
5	2,202	63.66	**64.14**	64.27	**65.15**	71.09	**72.40**	73.37	**73.88**
6	602	50.72	**52.85**	51.62	**55.13**	59.30	**63.61**	61.32	**64.79**
7	130	36.25	**43.38**	40.53	**44.91**	45.51	**55.02**	43.59	**50.37**
8	20	11.24	**24.39**	16.44	**19.51**	**24.32**	24.24	16.67	**20.00**
9	6	12.90	**66.60**	21.43	**28.57**	16.67	**33.33**	12.90	**54.55**
$k = 1$	36,317	**85.95**	85.92	**86.41**	86.39	**89.85**	89.75	**90.69**	90.66
$k = 2$	1,963	**59.88**	59.64	59.95	**62.05**	**71.15**	67.53	69.78	**70.85**
$k = 3$	194	57.46	**59.83**	58.89	**60.61**	**68.23**	63.91	**69.21**	68.95
All	38,474	**84.56**	84.50	84.99	**85.08**	**88.82**	88.57	89.55	**89.58**

TABLE 5.6: Multi-branching and discontinuous test F1 scores of **DB** and **DM** on TIGER. Fan-out is detailed in k.

5.3.2 Multi-branching Arity and Fan-out Degree

I present the F1 scores by discontinuity and multi-branching arity in Figure 5.5. Like sentence length, discontinuity and multi-branching arity generally create more difficulties for constituency parsing. Both **DB** and **DM** exhibit declining trends of F1 scores with the increasement of the two aspects.

DM exhibits persistent advantages over **DB** when these properties are frequent. I further examined that **CM** has the same gains over **CB** starting identically from 4-ary nodes with minor score differences on (D)PTB under the same $\rho_\varnothing = 0$ condition, as shown in Table 5.5. The result supports the argument of Xin, Li, and Tan (2021) that n-ary constituency parsing without binarization preserves some natural advantages, e.g., predicate-argument structure. Specifically, $\rho_\varnothing > 0$ shifts **DM**'s multi-branching advantage to frequent low-arity trees, favoring the overall scores on DPTB, while it enhances both discontinuity and multi-branching advantages on TIGER, as shown in Table 5.6, in agreement with Table 4.11.

5.3.3 Hyperparameter Tuning and Robustness

On one hand, **CB** and **CM** have fewer hyperparameters and the optimal loss weights can be obtained with grid search, as demonstrated in Figure 4.1. The result show putting more weight on the structural orientation part brings more marginal accuracy gain.

One the other hand, **DB** and **DM** has more hyperparameters which necessitate the automatic BO hyperparameter tuning process. Likewise, I can obtain some interpretation from Figure 5.6. For example, on TIGER, higher α_{ori}^{shfl}, β_C, and β_X are preferable.

From Figure 5.5 and Table 4.10, we learnt that TIGER is more challenging in discontinuity. Bad discontinuity prediction seems to cascade to multi-branching prediction, resulting in degradation of both properties. Meanwhile, DPTB is largely transformed from PTB by typed traces and automatic rules, where the multi-branching accuracy stays more stable. Higher α_{ori}^{shfl}, β_C, and β_X help **DB** and **DM** achieve better robustness for discontinuity.

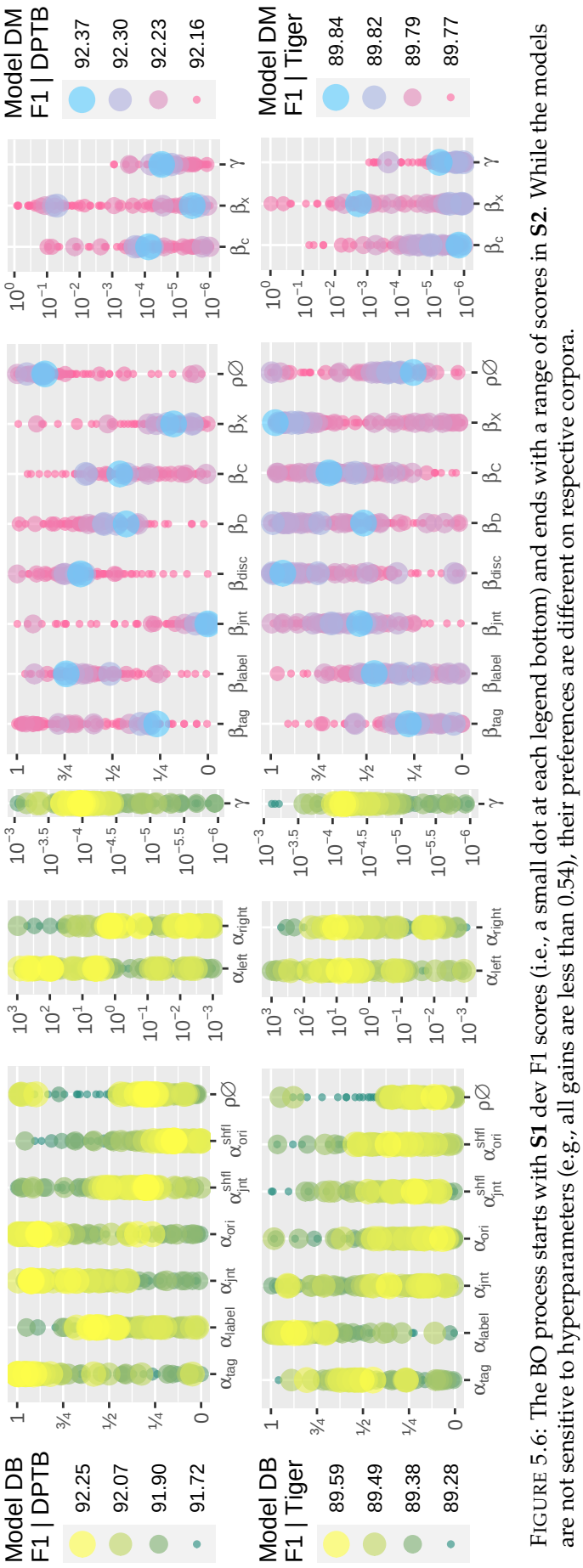


FIGURE 5.6: The BO process starts with **S1** dev F1 scores (i.e., a small dot at each legend bottom) and ends with a range of scores in **S2**. While the models are not sensitive to hyperparameters (e.g., all gains are less than 0.54), their preferences are different on respective corpora.

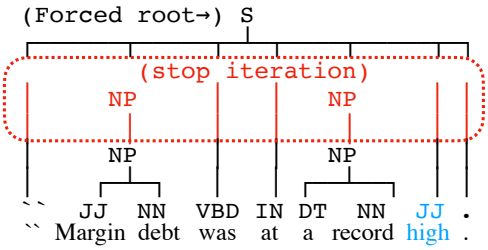


FIGURE 5.7: Failed parse from the multi-branching model. The model stops parsing and saves computations when it repeats the same chunking positions.

Test set	DPTB	TIGER
Total trees	2,416	4,998
DB's ill-formed parses	1	2
DM's ill-formed parses	15	47
Biaffine attention matrices	594	7,278
$\theta = 0.5$ solutions	587	7,114
Average of tries if $\theta \neq 0.5$	12.4	42.9
`FAIL` + identity matrices	0+4	0+81

TABLE 5.7: Errors in discontinuous **NCCP** PLM models with $\rho_\varnothing > 0$. A `FAIL` causes a matrix of ones, whereas a θ close to one gives an identity matrix — an expensive null action.

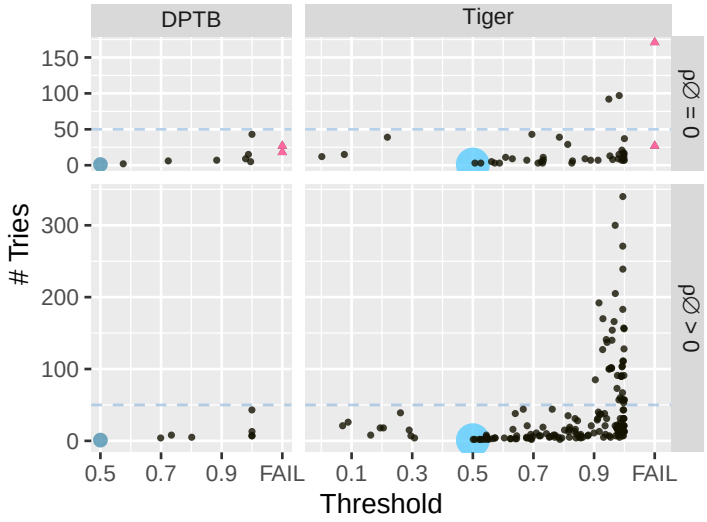


FIGURE 5.8: The numbers of tries to decompose *biaffine attention* matrices. `FAIL`s are marked with "▲".

5.3.4 Inference Error Rate

As greedy parsers allow ill-formed outputs without a single root, especially in case of single-model inference, the rate of an invalid parse is a focused topic for my greedy parsers. During the training process, fatal errors, such as frame-breaking orientations, appear at an early stage of training. However, the late 90% training time contains very few errors. **CB** on PTB is free from invalid parsing on the test set. For **CM**, it is observed that 11 out of 2,416 test parses are forests rather than parse trees when they are trained with fastText. However, the multi-branching parser with fine-tuned XLNet reduces the error count on the test set to 1.

I present a failed **CM** parse, as shown in Figure 5.7. The postnominal adjective "high" is uncommon for English. Because the model did not group it with the adjacent "a record" to form an `NP`, the error propagated to higher layers (e.g., no `PP` as an adjunct to form a `VP`), causing the bad parse. It implies that the multi-branching models require an appropriate predict-argument configuration to chunk.

In the same vein, discontinuous models **DB** & **DM** yielded a few invalid parses, as demonstrated in Table 5.7. **DM** models produce more errors. However, unsuccessful decomposition of *biaffine attention* matrices might not be the direct cause, as also shown in Figure 5.8. $\rho_\varnothing > 0$ variants cleared the matrices which cannot be

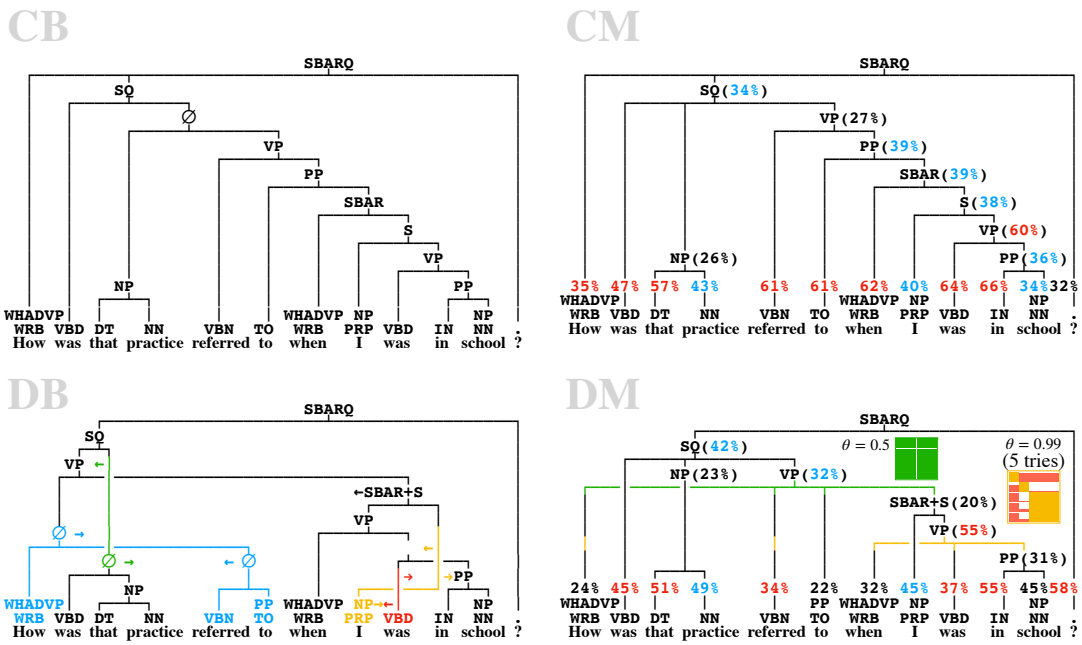


FIGURE 5.9: An exact matched DPTB sample from PLM **DB** and **DM** models versus **CB** and **CM** on PTB. The parse contains complex nested clauses which **CM** must fail to capture, and it becomes ungrammatical in the continuous scenario. **DB**'s outputs include orientations depicted as arrows and their traveling traces colored for groups. Meanwhile, **DM** produces two *biaffine attention* matrices, one of which has a highly biased but correct threshold $\theta = 0.99$. Bar heights indicate values in matrices and their colors indicate the relationship to θ.

decomposed with any θ (FAIL). Similar to **CM**, this genre suffers from more failures. Specifically, the ply size cannot be reduced to one during the iteration. Greedy parsers must suffer the defect because of their simplicity. However, invalid parses can contribute positive F1 scores and global parsers can yield inaccurate parses.

I applied methods such as Boolean matrix factorization and singular value decomposition. However, they did not show any improvement but significantly slowed down the speed. This is because $\theta \neq 0.5$ cases are few. My sequential tries to decompose might be naïve, but it is effective. In Figure 5.8, the number of tries does not significantly increase under $\theta < 0.9$ within 50 tries. For $\theta \geq 0.9$, although some tries are expensive, I will see that they are worthy in the next section. The imbalanced signals from both datasets account for the bias of θ — more than 92% *biaffine attention* signals are ones, as shown in Figure 5.1.

5.4 Sample Analysis

5.4.1 Continuous vs. Discontinuous Parsing

Figure 5.9 highlights the value of discontinuous parsing by demonstrating the respective **CM** parse. Conspicuously, the branching tendency of the continuous parse is to the right, while it is not obvious for the discontinuous parsing. Meanwhile, I observed instances of similar unsupervised headedness weights. This sample is not trivial, which challenges both PLM **DB** and **DM** models.

In Figure 5.9, **DB** shows sinuous travel traces of "I", "was", and larger ∅-subtree nodes which involve the turning of orientations. The varying context leads them to

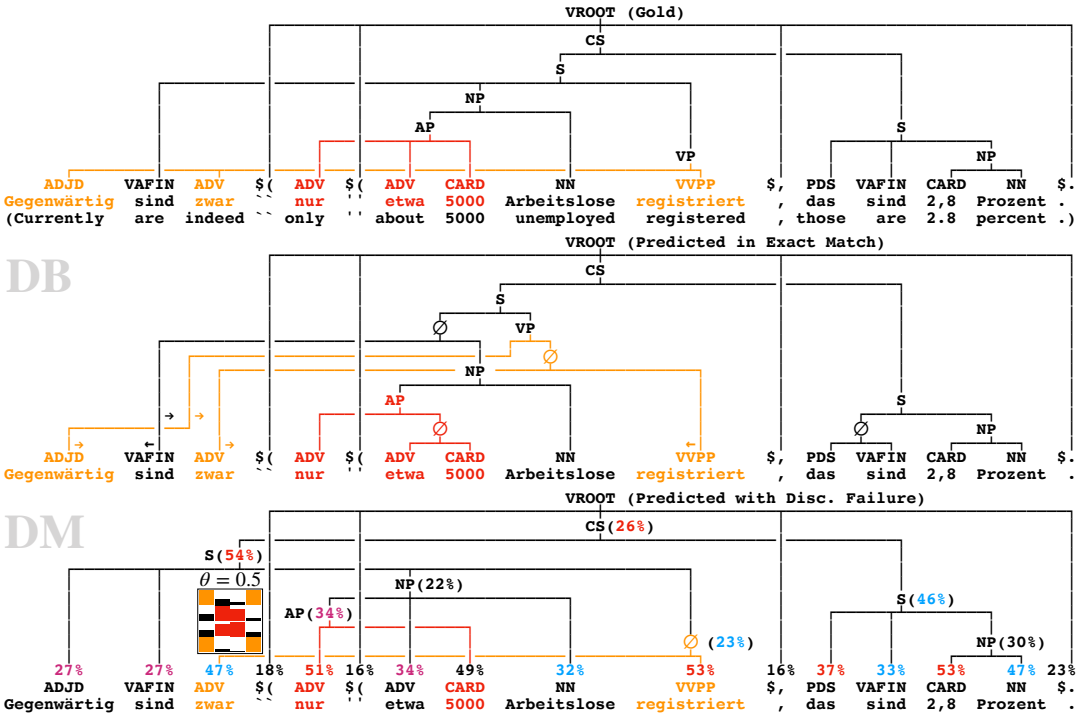


FIGURE 5.10: A TIGER parse. **DB** natively with ∅-subtrees achieved the exact match but **DM** erred with $\rho_{\varnothing} = 0$.

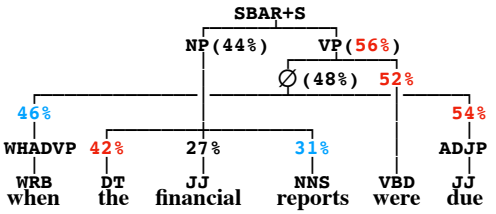


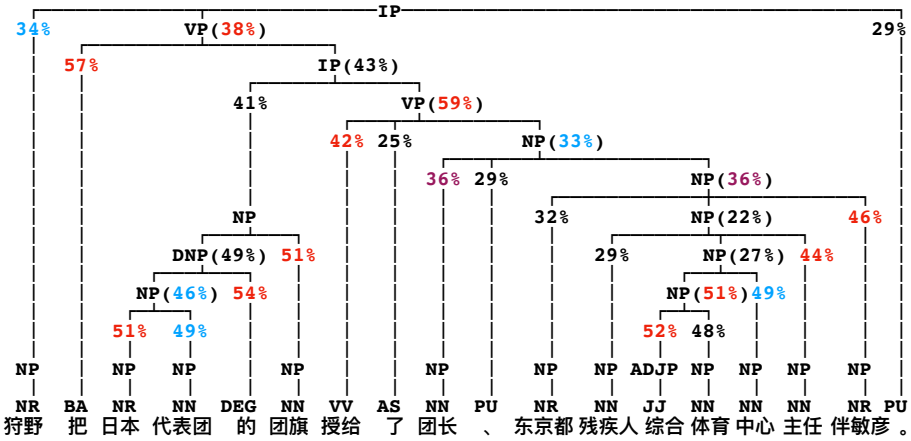
FIGURE 5.11: A semantic ∅-subtree by **DM** with $\rho_{\varnothing} > 0$. Copula "were" has less *affinity* than "when" and "due".

achieve complex movement. **DB** also created some grammatical substructures for "How", "referred", "to", "was", "in", and "school".

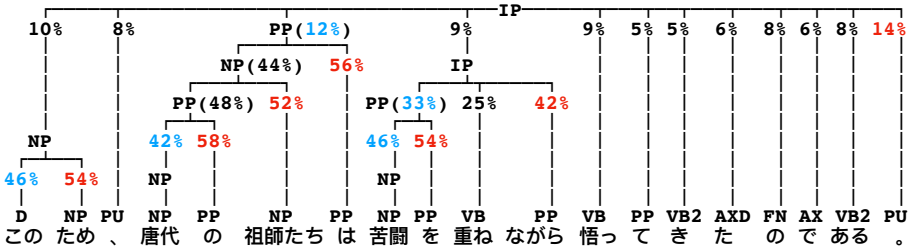
Meanwhile, the **DM** parse is more dramatic. The formation of the lower discontinuous VP involves five nodes, two of which are irrelevant words "How" and "referred" triggered by wrong *discontinuity* signals. They are discontinuous but for higher VP. The two nodes create a noisy *biaffine attention* matrix because their grammatical roles are compatible with the lower VP. Trained for extra robustness, the matrix decomposition with 5 tries found the right θ to identify the correct VP members excluding "How" and "referred". The interply loss and the decoding process gave this parse a chance for perfection.

In Figure 5.10 for German, **DB** achieved a long distant constituent in a more subtle way. The word "zwar" joins "registriert" as an ∅-subtree when the formation of intermediate NP shortens their distance instead of a travel through. "Gegenwärtig" follows and forms a VP. However, **DM** failed.

The $\rho_{\varnothing} > 0$ matters. The above failure explains why **DM** is inferior to **DB** with $\rho_{\varnothing} = 0$. **DB**'s orientation system allows some free travel before joining with correct mates. The constituent formation through steps of accumulation creates a more



Translation: Kano awarded (授给/了) the Japanese delegation's flag to the head of the delegation (团长) and director (主任) of the Tokyo Comprehensive Sports Center for the Disabled, Toshihiko Ban.



Translation: For this reason, the Tang dynasty's (の) ancestors (祖師たち/は-subject) have come to realize (悟っ) it through struggles (苦闘/を-object).

FIGURE 5.12: Chinese (top) and Japanese (bottom) parses from **CM** model.

stable context. However, **DM**'s group action happens all at once. A wrong composition might create a quite different context, which leads to unseen reaction chains. The strange unsupervised headedness weights reflect the issue. On the other side, with $\rho_{\varnothing} > 0$, **DM** can also gradually build and discover some semantic substructures, as shown in Figure 5.11. In contrast, **DB** is not sensitive to $\rho_{\varnothing}$ because of its nature, in agreement with Table 4.11.

5.4.2 Unsupervised Headedness in Japanese and Chinese

Figure 5.12 presents two non-English parses from the **CM** model. Both the Chinese and Japanese languages possess functional markers that receive high attention (percentage and words in red), such as the second character tagged with `BA` in Chinese, and Japanese case markers tagged with `PP`. Interestingly, the Chinese verb (i.e., the word meaning "awarded") received the highest attention, whereas Japanese verbs (i.e., two sub-words tagged with `VB`) did not. I supposed the reason behind this is that Japanese sentences drop the `VB`s and other heads more often than Chinese. The coordinated `NP`s in the Chinese parse (i.e., two words meaning "head" and "director") received equal attention weights. Moreover, two trees show their branching tendencies: Chinese is *middle*-factoring-alike; Japanese is a *left*-branching language, and KTB has a large flat structure on the right.

5.4.3 Structured Sentiment Analysis

As shown in Figure 5.13, sample (a) has an obscurely hard-to-capture label 3 for "*is a gem*". The model failed to label it with 3 but "*is a gem .*". This sample is not a rare case

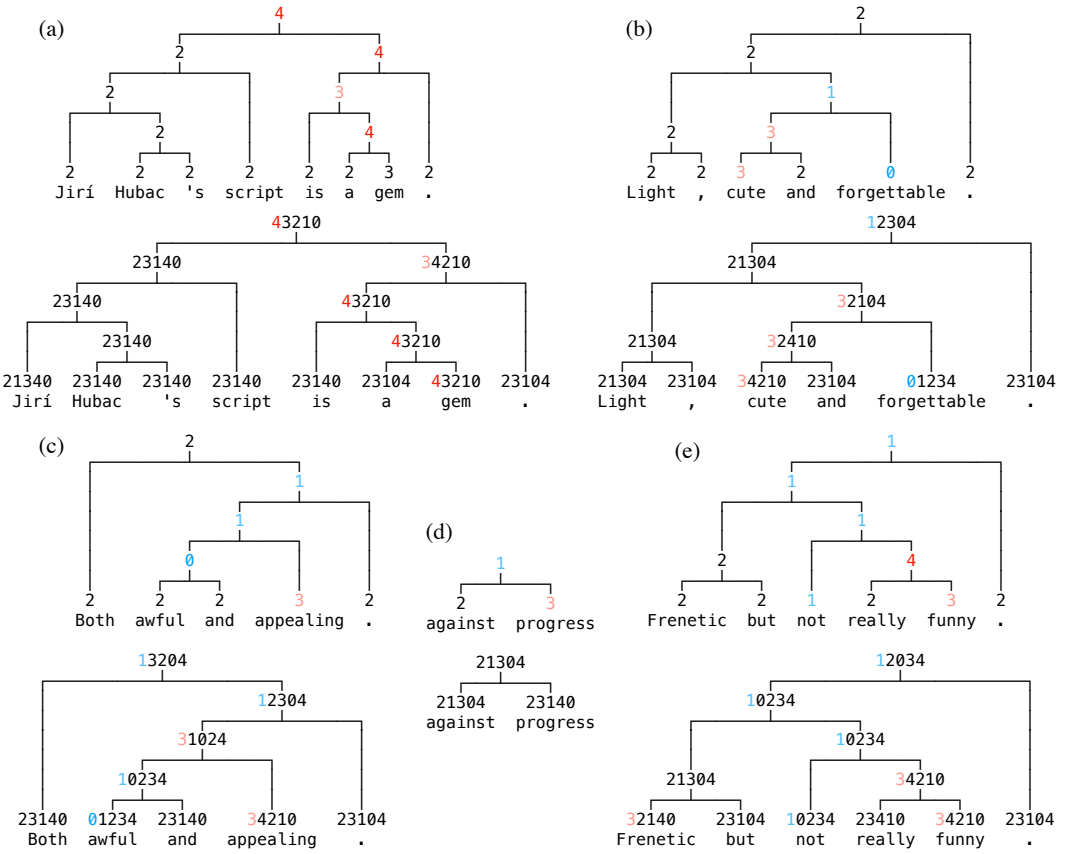


FIGURE 5.13: Samples from SST (upper parts) and its PWE**CB** prediction (bottom parts). Sentiment labels are color in red for positive and blue for negative. Each **CB** prediction is a string of five labels sorted by their scores and the leftmost label is the result.

for the obscurity in SST, which makes the task challenging. Samples (b) and (c) are binarized coordination where **CB** also failed due to the semantic reasons. Sample (b) relays the right-hand side sentiment but (c) relays the left-hand side. Sample (d) contains a semantic negation "*against*" that reverts the polarity of *progress*. **CB** capture all labels as neutral. Finally, sample (e) has an common negation "*not*" which **CB** managed to predict. However, the reinforcement from "*really*" to "*funny*" is not captured.

Chapter 6

DAG Conversion for PTB and CTB

Natural languages contain complicated structures beyond tree structures, as introduced in Section 1.2.2. PTB and CTB as two of the most frequently studied large-scale annotated phrase structure corpora has co-indexing systems (Bies et al., 1995; Xue et al., 2000) to recover discontinuity and multi-attachment for DAG structure.

The recovery of discontinuity is pioneered by Evang (2011), which then facilitates the research of discontinuous constituency parsing in the recent decade. It is done by looking for each pseudo-attachment with a co-indexing number to its true syntactic parent. The pseudo-attachment is a phrase marked by an identity index (i.e., an unique integer suffixed with the constituency label) and the other null element(s), which are one or more terminal(s) marked by a corresponding reference index (i.e., co-indexing). The pseudo-attachment is detached from its original parent in the continuous bracketing format and is reattached to a phrase with a corresponding reference index to replace its null element. The recovery by Evang (2011) considered only one null elements and such detach-reattach operations create the discontinuity within the tree structure.

However, there are cases where multiple co-indexing null elements exist, which means the pseudo-attachment should be shared by multiple phrases with the identical indices. Multi-attachment leads to the DAG structure. Evang (2011) avoided multi-attachment by resolving a closest phrase for reattachment, while discarding other phrases and creating ungrammatical structures. Instead, I provide method with manual check for the full reattachment in this thesis. Moreover, I also address and include intra-sentential gapping previously avoided by him. (PTB co-indexing does not support inter-sentential phenomena.) I also investigate CTB which has a close annotation scheme to PTB. I provide sample analysis and statistics after conversion. Please find my code more details.

Most research for constituency parsing is supervised, where parsers follow the annotation schemes of corpora. Many other constituency corpora, such as KTB, NINJAL Parsed Corpus of Modern Japanese (NPCMJ) Japanese Language and Linguistics (2016) and shared task SPMRL 2013 Seddah et al. (2013), are annotated in bracketing format that limits the parsing structure to continuous trees. Those treebanks require special knowledge for their respective DAG conversion. Finally, I think providing a highly specialized parser only for constituency parsing has somehow very limited range of usage. Thus, I discuss a new pair of NCCP parser for the DAG structure, which potentially be applied beyond constituency parsing.

Algorithm 5: DAG conversion with null element.

```
1  T ← {*T*, *ICH*, *EXP*, *RNR*, *};
2  foreach sentence s do
3      foreach node c from s co-indexed with null elements N of types in T do
4          foreach n in N do
5              attach c to the grandparent of n (i.e., destination d);
6              if c properly dominates d then
7                  find the node labeled with PRN on the path from c to d, call it e;
8                  detach e from its current parent;
9                  if c is not the root of s then
10                     attach e to the parent of c;
11         if none of N belongs to type * then
12             detach c from its original parent;
13     go to Algorithm 6 for intra-sentential gapping;
14     delete all null elements;
15     delete all non-terminals without children;
16     remove all remaining indices from node labels;
```

6.1 Conversion with Co-indexing

6.1.1 Trace with Null Element

Following Evang (2011), I consider five types of null elements for reattachment in PTB. For types already exploited by Evang (2011) to create discontinuous trees or not, I use ✔ for his fully exploitation, ▲ for partially exploitation, and ✘ for no exploitation.

✔	***T***	*A-bar movement*.
✔	***ICH***	*Interpret constituent here*.
✔	***EXP***	*Expletive*.
▲	***RNR***	*Right node raising*. (Evang (2011) picks the closest reattachment.)
✘	*	*A-movement*.

Among those types, types ***RNR*** and * are special because they lead to the DAG structure with a fully exploitation in Algorithm 5, The loop in line 4 enables multi-attachment. Specifically, type ***RNR*** and type * are different. Type ***RNR*** requires detachment from the pseudo-attachment, while type * keep the original parent attached during reattachment. The reason for detachment is that their parent is a pseudo-attachment that does not have any syntactic role. In contrast, non-* nodes do have (Marcus et al., 1994). Besides the above differences, The improvement also includes lines 11–13 for gapping.

An instance in Figure 6.1 exemplifies Algorithm 5. The A-movement of *programs like this* has two parents and two roles for them; One is as the original subject for `SQ` and the other is as an unlabeled object for `VP` of the verb *eliminated*. Because two roles are grammatical for their parents, there is no need to remove the original attachenment like *why* originally at `SBARQ`. Noticeably, my conversion keeps the function tags for grammatical roles as Evang (2011) does, such as `-SBJ` and `-PRP`.

There are a few cases where different trace types share the same reference index, where I need to decide whether to detach the subtree with the identity index. I will cover those minority in Section 6.2.

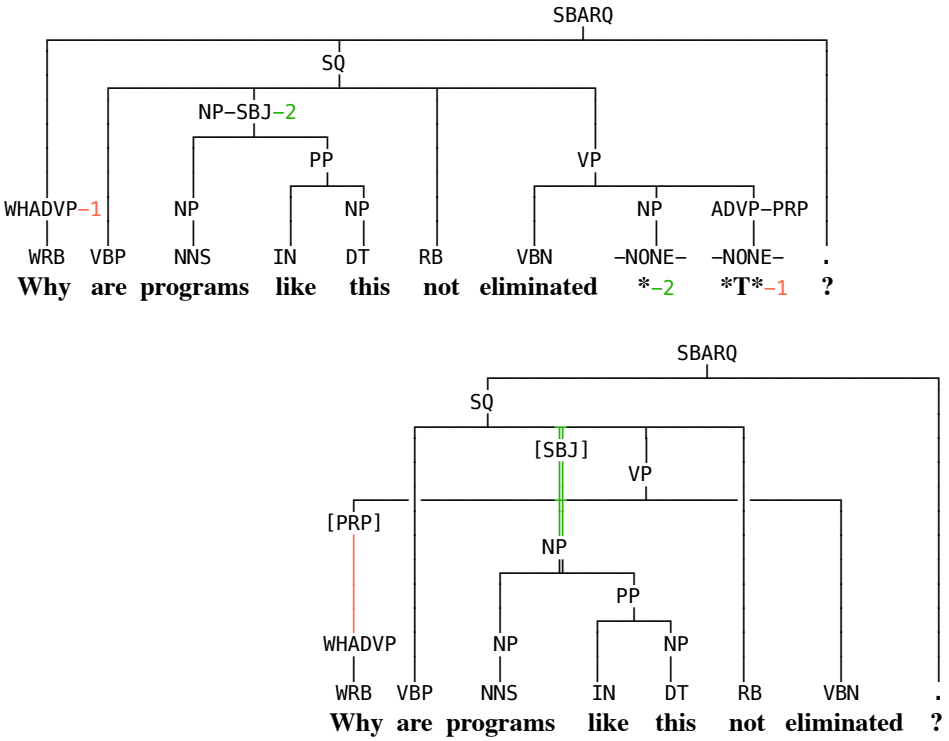


FIGURE 6.1: DAG conversion with both A-movement (*) and A-bar movement (*T*).

6.1.2 Intra-sentential Gapping

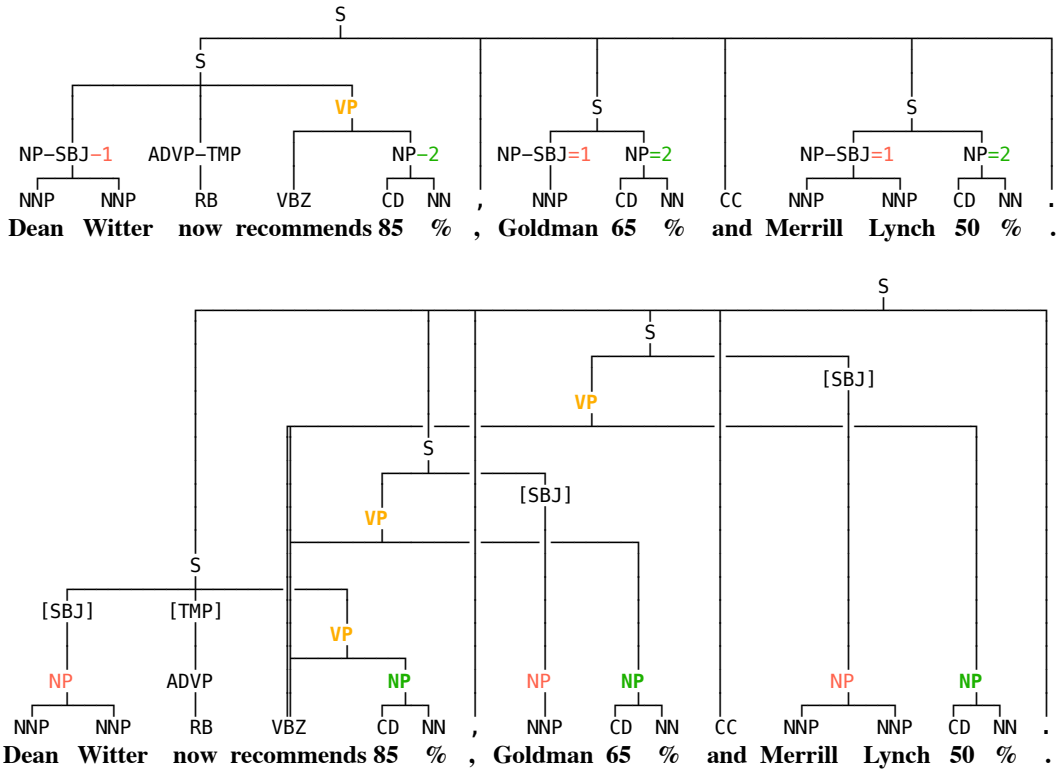


FIGURE 6.2: DAG conversion with gapping coindexing marks.

PTB co-indexes each coordination with either "-" for a complete phrase (i.e., a template usually at the initial place) or "=" for one or more incomplete phrases (i.e.,

Algorithm 6: DAG conversion with intrasentential gapping.

```
1  Function SHARE-COMPLEMENT(template phrase t, gapped phrase g):
2      foreach not co-indexed direct child r of t do
3          if r is a complement of t and r is not a child of g then
4              attach r to g;
5  Function IMITATE(template phrase t, gapped phrase g):
6      SHARE-COMPLEMENT(t, g);
7      foreach common co-index c do
8          c co-indexes (i, j)-th descendants (t^i, g^j) of (t, g);
9          if i = j then
10             if i = j = 1 then
11                 continue;
12             else
13                 IMITATE(t^1, g^1);
14         else
15             create a new child ḡ for g with the label of t^1;
16             remove g^1 from g and attach g^1 to ḡ;
17             IMITATE(t^1, ḡ);
18 if the only template phrase t in sentence s exists then
19     foreach corresponding gapping phrase g in s do
20         IMITATE(t, g);
```

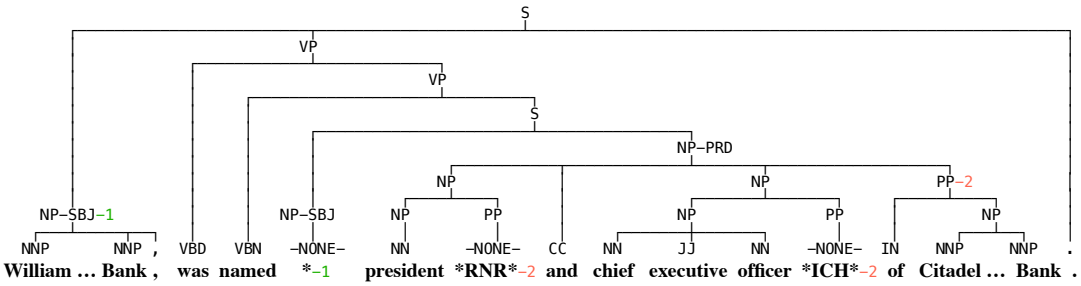
gapping with remnant components) to imitate the template. The template and gapping phrases are commonly conjunct in no more than one coordination structure in a sentence.

Algorithm 6 has a recursive `IMITATE` function making each gapping phrase resemble its template. In PTB, gapping phrases must have incomplete structures of small heights. Each `IMITATE` entrance lets a gapping phrase regain one level of missing complements from the template via the `SHARE` function. The missing structures are recovered by lines 15 & 16 by forcing co-indexed nodes to grow to the same height with the same structure.

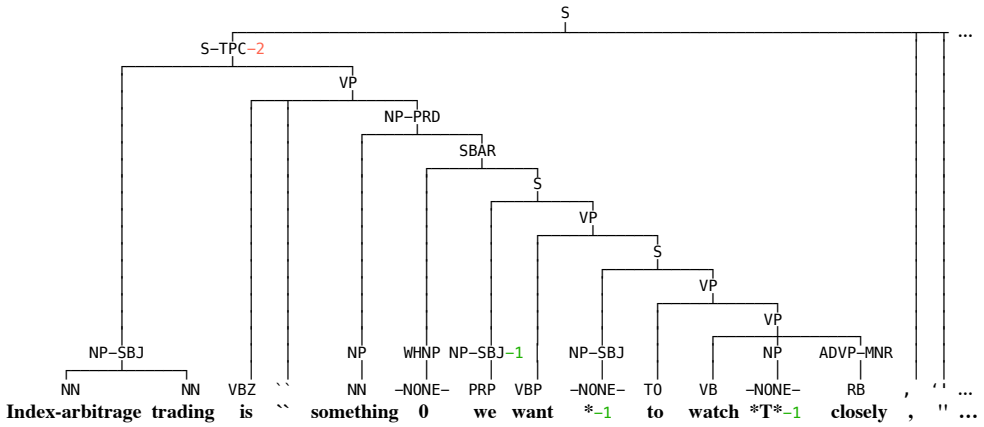
I exemplify the process in Figure 6.2. At first, `S` for *Dean Writer new recommends 80 %* is found to be the template t with `S` for *Goldman 65 %* being an imitator phrase. The modifier `ADVP` is not shared to the imitator because it is not a complement for the formation of `S`. Then, a new substructure `VP` (in yellow) is created by lines 14 & 15 and the second `IMITATE` entrance creates two `VP`s in two respective `S` nodes. In contrast, `VBZ` is shared by `VP`s because it is the complement for them. The process works in a top-down style for every substructure paired with the template and gapping phrases.

6.2 Exception and Error Correction

The annotation of PTB involved a large amount of human labor. Thus, a few errors and exceptions are inevitable. Evang (2011) adjusted annotation for 26 sentences in DPTB via my implementation to recover the exact DPTB corpus. In addition, I find 10 sentences during my conversion.



(a) Wrong trace type of ***ICH***.



(b) Wrong trace type of ***T*** and missing index of null complementizer.

FIGURE 6.3: Two annotating errors in PTB causing the same index for different trace types. Top: `*ICH*` is supposed to be `*RNR*`. Bottom: `*T*-1` should be a null element for null complementizer *0* instead of *I*.

6.2.1 Wrong Trace Type

I show annotating errors in PTB which causes the same index for different trace types in Figure 6.3. The top example (a) does not trigger an error in Algorithm 5 because the algorithm is not sensitive to the difference between ***RNR*** and ***ICH***. The result of the conversion is a DAG with the two different reattachment and the detachment of `PP-2` from `NP-PRD`.

Meanwhile, the bottom example (b) triggers the an error because `*T*` requires detaching *I* from `S` but `*` does not. My reannotation is to put a suffix `-2` to `WHNP` and replace `*T*-1` with `*T*-2`. The total samples of wrong trace type is four, two of which are already addressed by Evang (2011).

6.2.2 Circle in Gapping

Although there seems no limitation that the syntactic can go beyond DAG for cyclic graph, cyclic graph is the very minority which makes parsing very difficult to handle. To my best knowledge, no parser support cyclic graph. In case of my `NCCP` parsers, they cannot make a decision to change the structure it already produced, even for the concept for DAG parsers that will be discussed in Section 7.4.

Figure 6.4 shows (c) a sample that contain mutual referred indices that create cyclic graph. There are only a few such samples and I manually reduce the cyclic graphs back into DAGs by removing some indices. In the case of (c), I discard ***ICH***`-2` because A-movement is more common in statistics. (However, I provide

an option in implementation that allows the alternatives.) Some other A-movements refer to its descendants and I also discard such indices.

6.2.3 Ill-formed Coordination

My DAG conversion features in intrasentential gapping from coordination. However, some of them are ill-formed that lack basic structures that makes them symmetric to the template phrase. Figure 6.4 (d) shows such a case. Those sample are very easy to detect because the labels of the gapped phrases are not the same with the label of the template phrase and the tree heights are not matched. Manual reannotation ensures the samples are regular coordintation. There is only one exception sample that the gapped phrase is not in the coordination. I manually indicate the roots of template phrase and the gapped phrase for Algorithm 6.

6.2.4 Involving External Phrase

Algorithm 6 executes after Algorithm 5 so that gapped phrase can immitate the structure in the trace-modified template prhase. However, there are three samples with both expletive (***EXP***) and coordination, which the gapped phrase cannot catch the structure removed by ***EXP***. Besides, the label of coordination is not grammatical after the process.

Thus, I manually changed the coordination label and the structure to fit it into my algorithms. Figure 6.4 (e) shows such a case.

6.2.5 Difference of PTB and CTB

CTB uses similar annotation guildlines to PTB. However, there are some certain differences:

- Fewer trace type with coindexing: ***T***, ***RNR***, and *.
- The coordination template phrase is also marked with =.

Thus, Algorithm 5 is applicable for CTB but Algorithm 6 needs to first decide which phrase is the template. The decision is the phrase that contains the most unindexed descendant phrases because some templates are the last phrase of the coordination.

6.3 Summary and Discussion

6.3.1 Data Statistics

As a new feature of DAG corpora, I show the statistics of number of parents and their conversion types in Tables 6.1 & 6.2. DPTB can be seen as a subset which only has the 1-ary column of each table. For both DAG conversions of PTB and CTB, * and ***T*** are the major sources of reattachment, while *, **gapping**, and ***RNR*** are the top three DAG provider. However, both reattachment and DAG are very sparse, especially for CTB, which makes the task very challenging.

6.3.2 Comparison with Combinatory Categorical Grammar

As a parsing formalism, CCG also addressed gapping phenomena in coordination (Little, 2010). The solution of CCG is through compound lexical tags (i.e., supertagging) operated by combinatory rules, whereas ours is not a parsing formalism but a graphbank in DAG with each node being an atomic constituency label.

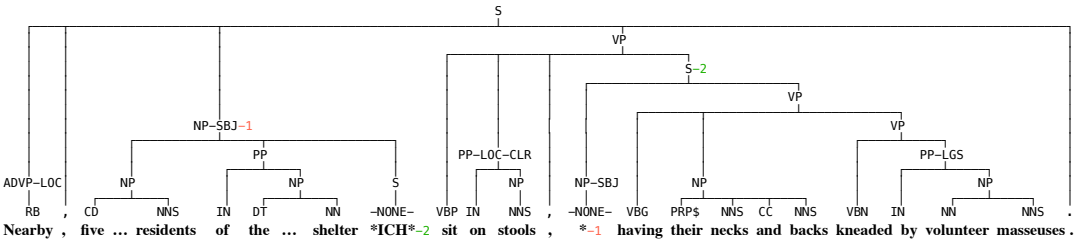
Type	Phrase Level: Number of Parents						Sentence Level	
	1-ary	**2**-ary	**3**-ary	**4**-ary	**5**-ary	**6**-ary	Reatt.	DAG
T	15,837	71	3	-	-	-	13,584	82
***T*-PRN**	840	2	-	-	-	-	842	4
ICH	1,265	2	-	-	-	-	1,239	3
RNR	2	203	5	1	-	-	209	210
EXP	657	1	-	-	-	-	651	1
*	13	15,590	1,906	217	22	8	15,992	17,756
Gapping	372	457	36	8	-	4	311	534
Total	55,226 reatt. of 918,730						26,164	16,324

TABLE 6.1: Statistics of PTB after my DAG conversion. PTB (2.0) has 49,208 sentences.

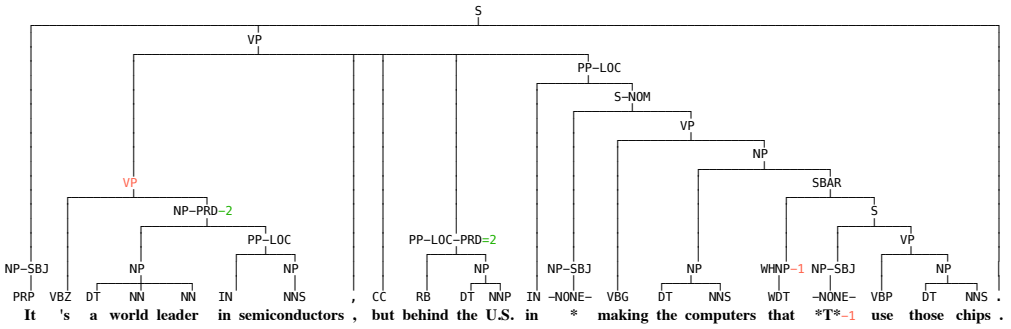
Type	Phrase Level: Number of Parents							Sentence Level	
	1-ary	**2**-ary	**3**-ary	**4**-ary	**5**-ary	**6**-ary	**12**-ary	Reatt.	DAG
T	3,214	36	6	-	-	-	-	3,132	43
ICH	26	-	-	-	-	-	-	24	0
RNR	999	48	7	-	-	1	-	989	58
*	2	2,971	55	4	1	-	-	2,877	3,031
Gapping	-	16	7	1	-	-	1	25	25
Total	10,426 reatt. of 2,476,071							6,883	2,992

TABLE 6.2: Statistics of CTB after my DAG conversion. CTB (9.0) has 132,080 sentences.

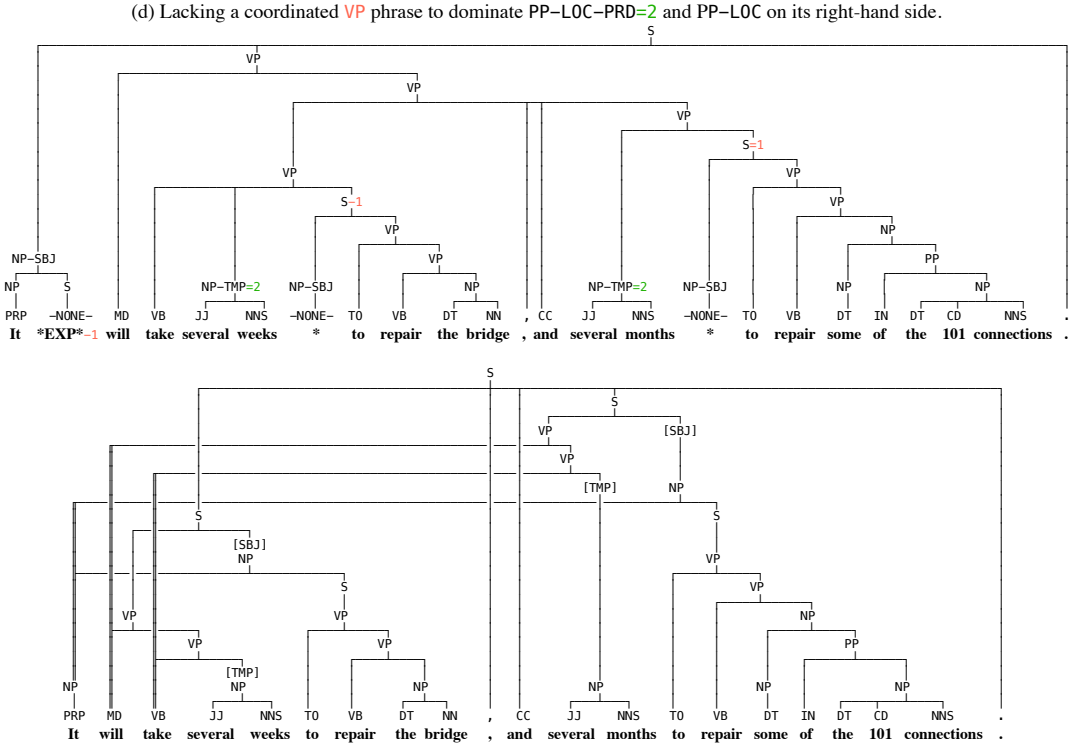
Specifically, CCG assigns each word with a string of constituency labels and operating slashes that define how the word combines with adjacent words to form a parse. Apart from CCG's ability for gapping, its parsing formalism is context-free by assuming that necessary contextual information is locally encoded into each lexical string. Our graphbank simply provides the DAG structure without assumption. Moreover, operating slashes can only handle binary combinatory operations, in contrast to our conversion's flexible number of child phrases.



(c) Circle that beyond the capacity of DAG structure



(d) Lacking a coordinated VP phrase to dominate PP-LOC-PRD=2 and PP-LOC on its right-hand side.



(e) Coordination gapping that involves expletive.

FIGURE 6.4: Another three annotating error or exceptions in PTB. Top (c): the original annotation leads to a directed cyclic graph instead of DAG. This is a rare case in PTB and I choose to reduce it to DAG by manually removing one or more paths. Middle (d): some samples do not follow PTB guidelines and miss some phrases structure for coordination. I manually add such phrases by referring to the template phrase. Bottom (e): three samples involve both expletive (***EXP***) and coordination, which make Algorithms 5 & 6 miss key components. I manually change the coordination of VP into S and the construction for grammatical and symmetric coordination.

Chapter 7

Conclusion

7.1 Conclusion Remarks

In this thesis, I proposed a *neural combinatory constituency parsing* family or **NCCP** that utilizes neural combinators for fast accurate constituency parsing and versatile non-parsing tasks. I generalize **NCCP** models into that transition-based genre, which share common features, such as state and iterative ply of finite automata and chunker-based parsers. **NCCP** is also grammar-less but reflects linguistic phenomenon such as branching tendency and headedness.

I implemented four parsers, **CB**, **CM**, **DB**, and **DM**, from the combination of *{continuous, discontinuous}* tree-based constituency parsing in *{binary, multi-branching}* styles. The *orientation*-based binary parsers **CB** and **DB** are the special cases of the multi-branching **CM** and **DM** that utilize *chunking* function. Specifically, the discontinuous models **DB** and **DM** are extension of the continuous models. For discontinuity, I equip **DB** with a *swap* operation and generalize **DM**'s continuous chunking function into discontinuous *affinity*. To further expand NCCP family, I provide two conceptual DAG parsers: **GB** and **DM** (See Section 7.4). As a continuous work, **NCCP** family cover different grammar levels and parsing complexities in the Chomsky Hierarchy: context-free continuous tree (type-2, e.g., CFG), mildly context-sensitive discontinuous tree (type-1, e.g., LCFRS), and DAG from unrestricted level (type-0).

Through experiments, new state-of-the-art parsing speeds on all five corpora are oberserved, which exhibit significant margins over recent approaches. Meanwhile, accuracy of **NCCP** parsers is close to the best models. There are still rooms for further accuracy improvements, given my parsers are greedy single models without decoding enhancement such as beam search. New state-of-the art discontinuous F1 scores are observed for my discontinuous parsers, suggesting that `NCCP` parsers can be good at handling new features if the models are well designed. The models achieved a nice balance among efficiency, effectiveness, and reflection of interesting linguistic phenomena.

Specifically, the lineup of binary combinators are **CB**, **DB**, and conceptional **GB**. **CB** and **DB** reflect the branching tendency of a given language during training, which is driven by signal bias in the corpus. Even though language-specific features can add extra human labor to the training process, the effects of which are relatively minor and knowledge about a language can guide our practice to counteract the effects. Finally, **GB** has an explicit replicate function to create DAG structure. Although there can be better solution, I expect the implemented **GB** model still be fast and influenced by the orientation and duplicate bias in a given corpus.

Meanwhile, the lineup of multi-branching combinators **CM**, **DM**, and conceptional **GM** exhibit different characteristics. **CM** and **DM** show multi-branching parsing advantage over the binary parsing, which support the observation of Xin,

Li, and Tan (2021), and the models provide statistically meaningful unsupervised headedness as a grammatical reflection and a useful by-product. **CM** needs one few hyperparameter (i.e., binarization) than its binary counterpart, whereas adding extra substructures as data augmentation do improve its accuracy. **DM** has an additional hyperparameter (i.e., medoid) but the unsupervised headedness can be a good choice for medoid. Finally, **GM** leverages biaffine attention for DAG and adds a new component for standalone medoids. Again, I expect the implemented **GM** model to reflect more grammatical headedness and to continue the advantage over its binary counterpart.

7.2 Potential Social Impact

The interaction between human and machine is a central topic since the born of ideas for automatic computation. Accurately interpreting the surface syntactic and deep semantic structures embedded in natural languages are important milestones toward the goal of human-machine interaction. Syntactic and semantic parsers are the efforts as those important milestones.

The trade-off between computational cost and performance is an unavoidable issues for parsing. **NCCP** family offers the fastest solutions (i.e., the advantage of linear empirical complexity and small memory footprint) to the community of parsing or human-machine interaction research with sufficient accuracy. As a prospective application, **NCCP** parsers can be transplanted onto portable devices for semantic parsing jobs, which are an essential part of intelligent virtual assistant (IVA) or intelligent personal assistant (IPA), such as Google Assistant, Amazon's Alexa, and Apple's Siri.

Beside IVA or IPA, some NLP applications involve hierarchical structures, such as event detection of information extraction. One of current efficient solution utilizes beam search with a transition-based parser for DAG in the domain of biomedical instruction (Espinosa, Miwa, and Ananiadou, 2019). Although this solution has low complexity of transition-based parsing, the beam search linearly slows down the parsing speed. In contrast, my future DAG parsers are expected to offer the results in one forward pass without beam search. For the overwhelming document being produced every day from different domains, an efficient solution can speed up many potential research of fundamental science.

7.3 Weakness and Limitation

A well-defined grammar is the standard to qualify whether a given sentence is grammatical or not. A generative parser can generate a correct sentence from scratch in a top-down style. Being grammar-less (i.e., not being generative), **NCCP** as well as many neural supervised discriminative parsers can neither generate sentences nor grammatically discern correct sentences from incorrect ones. However, sometimes a line between grammatical or ungrammatical is obscure, especially for nature languages. And the capacity of discerning incorrect sentence can be indirectly induced by introducing extra training processes or modeling uncertainty at inference.

While the multi-branching **NCCP** models show their natural intimacy with headedness, they do not support other properties as with HPSG AVM representation, which involves very detailed lexical and phrasal information. Even though one can assume that such information exists in embedding vector, the extraction can be very challenging, especially when supervision is lacking.

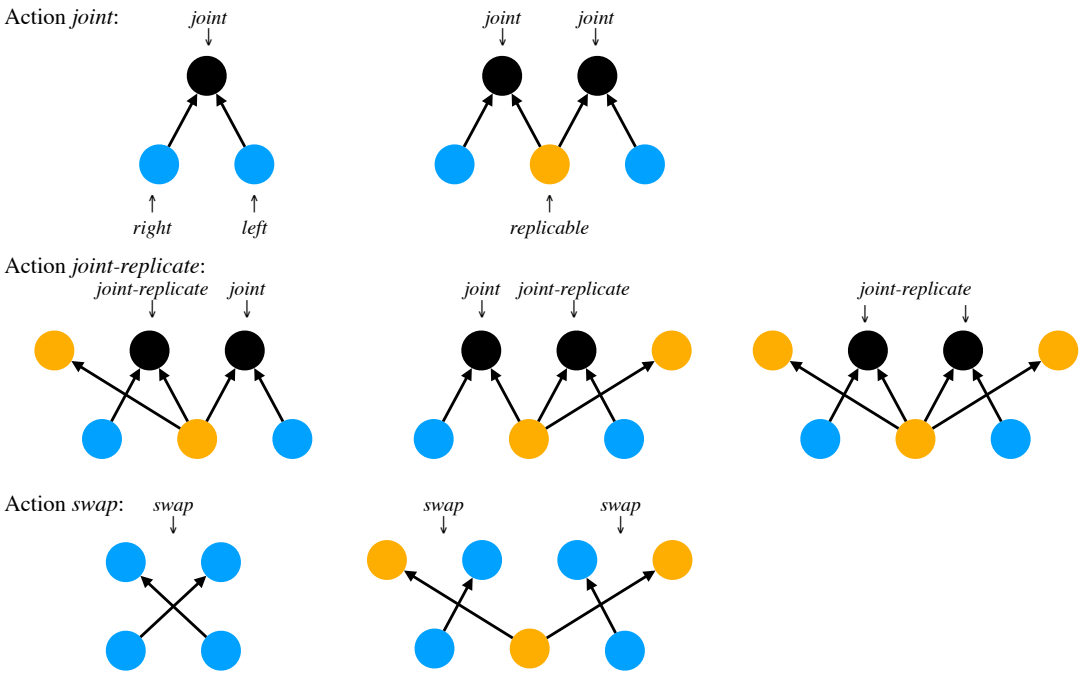


FIGURE 7.1: **GB** action cases.

Moreover, constituency is only one facet of syntactic parsing or general parsing. Lexical dependency and semantics should be included to make **NCCP** truly versatile. However, based on our current observation of joint training with sentiment analysis, it seems that syntactic and semantic tasks are not very compatible within **NCCP**. Further investigation is necessary to understand why such conflict occurs in terms of mechanism or nature of different tasks.

7.4 Future Work

DAG Parser in Concept I provide suggestions and naïve ideas for future DAG **NCCP** extension. It may still come with two variants: binary (**GB**) and multi-branching (**GM**). In the same vein, they build DAGs based on a bottom-up combinatory ply with respective atomic action sets. Because the major difference of discontinuous tree and DAG parsing lies in the multi-attachment capacity, the capacity to replicate the ply nodes could lead to those DAG models.

7.4.1 GB: Replicate Node

Current **DB** has a set of two signals {*orientation*, *joint*}. Each of them is a binary signal and the combination gives two types of actions: a *joint* action by *joint* signal on a pair of agreeing orientations and a *swap* action by *swap* signal on a pair of agreeing orientations. Could the remaining cases of combination be leveraged to house the capacity of replicating nodes?

Unfortunately, the remaining three pairs of disagreeing orientations (i.e., {(*left*, *left*), (*left*, *right*), (*right*, *right*)}) combined with the interstice *joint* (i.e., {*joit*, *swap*}) may have been intervened with other agreeing orientation pairs. Even it is possible to include replication, it makes the system hard to interpret.

I suggest to expand both orientation and joint-swap action for **GB**. Specifically, orientation function becomes

$$ori(x_i) \in \{-1, 0, 1\}, \tag{7.1}$$

where {-1, 0, 1} stands for *left*, *replicable*, and *right*. Thus, the agreeing orientation pair must be expressed as

$$ori(x_i) - ori(x_i) > 0. \tag{7.2}$$

The corresponding action conditioned on Formula 7.2 gets expanded to

$$\begin{aligned}
&action(x_i \oplus x_{i+1}) \in \{joint,\ swap,\ joint\text{-}replicate\} \\
&\qquad joint : (x_i,\ x_{i+1}) \to compose(x_i,\ x_{i+1}) \\
&joint\text{-}replicate : (x_i,\ x_{i+1}) \to (compose(x_i,\ x_{i+1}), x_i) && \textbf{if } ori(x_i) = 0 \textbf{ and } ori(x_{i+1}) \neq 0 \\
&\qquad\qquad (x_i,\ x_{i+1}) \to (x_{i+1},\ compose(x_i,\ x_{i+1})) && \textbf{if } ori(x_i) \neq 0 \textbf{ and } ori(x_{i+1}) = 0 \\
&\qquad\qquad (x_i,\ x_{i+1}) \to (x_{i+1},\ compose(x_i,\ x_{i+1}),\ x_i) && \textbf{if } ori(x_i) = ori(x_{i+1}) = 0 \\
&\qquad swap : (x_i,\ x_{i+1}) \to (x_{i+1},\ x_i).
\end{aligned} \tag{7.3}$$

Formula 7.3 keeps the **DB** basic orientation and joint-swap actions. It features in an additional *joint-replicate* action specialized for multi-attachment in DAG. Note that *joint-replicate* cooperates with the *replicable orientation* (0).

A node in **GB** can be bidirectional and duplicate split itself with the old *swap* action. When it needs to both joint and replicate, it first signals *replicable* and gets captured by *replicable orientation* at the interstice action. It can also create two joints at once by signaling *replicable* through the old *joint* action.

7.4.2 GM: Multi-medoid Biaffine Attention

To allow component-sharing in biaffine attention of *affinity*, I reduce the constrained on the matrix to be

- B symmetric and
- $B \cdot I = I$, where I is the identity matrix,

so that the following matrices can contain shared nodes for different groups:

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

The first matrix indicates that the second node is shared by two groups, where as the second matrix indicates the third node is shared by three groups.

The extension for **GM** need no extra signal except for a function to judge whether a *non-context-free* node is standalone or not. As shown in Formula 7.4, I change the action of **DM** from {*continuous*, *discontinuous*} into {*context-free*, *non-context-free*} in order to indicate the different processing subjects.

$$
\begin{aligned}
&\textit{action}(x_i) \in \{\textit{context-free},\ \textit{non-context-free}\} \\
&\textit{context-free}: \\
&\mathcal{G} = \{x_j \mid lb < i \leq rb,\ lb < j \leq rb,\ \text{and } \textit{affinity}(x_j,\ x_{j+1}) = \mathbb{1}(j \notin \{lb,\ rb\})\} \\
&\qquad\qquad \textit{compose}(\mathcal{G}) \rightarrow x_{lb+1} \\
&\textit{non-context-free}: \\
&\quad \mathcal{K} \leftarrow \text{ an empty set } \{\} \\
&\quad \text{For } j \text{ in } \{j \mid \textit{affinity}(x_i,\ x_j) = 1\}: \\
&\qquad\quad \text{append } \min(\{k \mid \textit{affinity}(x_k, x_j) = 1\}) \text{ to } \mathcal{K} \\
&\quad \text{For } k \text{ in sorted } \mathcal{K} \text{ by ascending order:} \\
&\qquad\qquad \mathcal{G} \leftarrow \{x_j \mid \textit{affinity}(x_g, x_j)\} \\
&\qquad\qquad \textit{medoid} \in \{j \mid x_j \in \mathcal{G}\} \\
&\qquad\qquad \textit{compose}(\mathcal{G}) \rightarrow x_{\textit{medoid}} \\
&\qquad \text{add any standalone } x_j \in \mathcal{G} \text{ to the ply}.
\end{aligned}
\tag{7.4}
$$

The *context-free* part remains exactly the same as those of **CM** and **DM**, which is the continuous chunking process. Meanwhile, from the line of "*non-context-free*:" to the last second line, the identification proceeds for each group $\mathcal{G}$ what may share nodes with other groups. The different groups are identified by the index of the left-most node. The last line is also special to **GM**. It keeps copies of some components for their siblings in higher plies.

7.4.3 Prospects of NCCP family.

From the viewpoint of generative parsing, **NCCP** lacks the capability to generate and rank sentences. Moreover, parsing annotation is quite expensive and exhausting. New improvements may considering make the models generative or unsupervised.

Comparing to information-rich HPSG constituency, the implemented **NCCP** models are very basic constituency parsers. To fully leverage the common information in current treebanks to be a useful parsing toolbox, **NCCP** need to create more wiring for semantic frames (e.g., predicate-argument structure). Both PTB-style and TIGER-style treebanks provide such information. Beyond tree-based constituency, I aim to implement the DAG parsers (i.e., **GB** and **GM**) for more parsing tasks. Finally, generalizing the models could be an interest research direction.

Bibliography

Akiba, Takuya et al. (2019). "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631. DOI: 10.1145/3292500.3330701. URL: https://doi.org/10.1145/3292500.3330701.

Backus, John W. (1959). "The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference". In: *Information Processing, Proceedings of the 1st International Conference on Information Processing, UNESCO, Paris 15-20 June 1959*. UNESCO (Paris), pp. 125–131.

Baker, James K (1979). "Trainable Grammars for Speech Recognition". In: *The Journal of the Acoustical Society of America* 65.S1, S132–S132.

Baldridge, Jason and Geert-Jan M. Kruijff (2002). "Coupling CCG and Hybrid Logic Dependency Semantics". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 319–326. DOI: 10.3115/1073083.1073137. URL: https://aclanthology.org/P02-1041/.

Barnes, Jeremy, Roman Klinger, and Sabine Schulte im Walde (2017). "Assessing State-of-the-Art Sentiment Models on State-of-the-Art Sentiment Datasets". In: *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pp. 2–12. DOI: 10.18653/v1/w17-5202. URL: https://doi.org/10.18653/v1/w17-5202.

Bies, Ann et al. (1995). "Bracketing guidelines for Treebank II style Penn Treebank project". In: *University of Pennsylvania* 97, p. 100.

Bojanowski, Piotr et al. (2017). "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5, pp. 135–146. URL: https://transacl.org/ojs/index.php/tacl/article/view/999.

Bradbury, James et al. (2017). "Quasi-Recurrent Neural Networks". In: *Proceedings of the 5th International Conference on Learning Representations*. URL: https://openreview.net/forum?id=H1zJ-v5xl.

Brants, Sabine et al. (2004). "TIGER: Linguistic Interpretation of a German Corpus". In: *Research on language and computation* 2.4, pp. 597–620.

Carl Pollard, Ivan A. Sag (1988). *Information-Based Syntax and Semantics*. Vol. Volume 1: Fundamentals. Stanford: Center for the Study of Language and Information Publications.

Carpenter, Brian E. and Robert W. Doran (1977). "The Other Turing Machine". In: *The Computer Journal* 20.3, pp. 269–279. DOI: 10.1093/comjnl/20.3.269. URL: https://doi.org/10.1093/comjnl/20.3.269.

Charniak, Eugene and Mark Johnson (2005). "Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking". In: *Proceedings of the Conference of the 43rd Annual Meeting of the Association for Computational Linguistics*, pp. 173–180. URL: https://www.aclweb.org/anthology/P05-1022/.

Chen, Zhousi et al. (2021). "Neural Combinatory Constituency Parsing". In: *Findings of the Association for Computational Linguistics: ACL/IJCNLP*, pp. 2199–2213. DOI: 10.18653/v1/2021.findings-acl.194. URL: https://doi.org/10.18653/v1/2021.findings-acl.194.

Chomsky, Noam (1956). "Three models for the description of language". In: *IRE Transactions on Information Theory* 2.3, pp. 113–124. DOI: 10.1109/TIT.1956.1056813. URL: https://doi.org/10.1109/TIT.1956.1056813.

Coavoux, Maximin and Shay B. Cohen (2019). "Discontinuous Constituency Parsing with a Stack-Free Transition System and a Dynamic Oracle". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 204–217. DOI: 10.18653/v1/n19-1018. URL: https://doi.org/10.18653/v1/n19-1018.

Coavoux, Maximin, Benoît Crabbé, and Shay B. Cohen (2019). "Unlexicalized Transition-based Discontinuous Constituency Parsing". In: *Transactions of the Association for Computational Linguistics* 7, pp. 73–89. URL: https://transacl.org/ojs/index.php/tacl/article/view/1544.

Cocke, John (1969). *Programming Languages and Their Compilers: Preliminary Notes*. New York University.

Collobert, Ronan (2011). "Deep Learning for Efficient Discriminative Parsing". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Vol. 15. JMLR Proceedings, pp. 224–232. URL: http://proceedings.mlr.press/v15/collobert11a/collobert11a.pdf.

Corro, Caio (2020). "Span-based discontinuous constituency parsing: a family of exact chart-based algorithms with time complexities from O(n\^6) down to O(n\^3)". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 2753–2764. DOI: 10.18653/v1/2020.emnlp-main.219. URL: https://doi.org/10.18653/v1/2020.emnlp-main.219.

Cross, James and Liang Huang (2016). "Span-Based Constituency Parsing with a Structure-Label System and Provably Optimal Dynamic Oracles". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1–11. URL: http://aclweb.org/anthology/D/D16/D16-1001.pdf.

David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams (1986). "Learning Representations by Back-propagating Errors". In: *nature* 323.6088, pp. 533–536.

Devlin, Jacob et al. (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*, pp. 4171–4186. URL: https://aclweb.org/anthology/papers/N/N19/N19-1423/.

Dozat, Timothy and Christopher D. Manning (2017). "Deep Biaffine Attention for Neural Dependency Parsing". In: *5th International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Hk95PK9le.

Drozdov, Andrew et al. (2019). "Unsupervised Labeled Parsing with Deep Inside-Outside Recursive Autoencoders". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 1507–1512. DOI: 10.18653/v1/D19-1161. URL: https://doi.org/10.18653/v1/D19-1161.

Durrett, Greg and Dan Klein (2015). "Neural CRF Parsing". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pp. 302–312. DOI: 10.3115/v1/p15-1030. URL: https://doi.org/10.3115/v1/p15-1030.

Dyer, Chris et al. (2016). "Recurrent Neural Network Grammars". In: *The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 199–209. DOI: 10.18653/v1/n16-1024. URL: https://doi.org/10.18653/v1/n16-1024.

Earley, Jay (1970). "An Efficient Context-Free Parsing Algorithm". In: *Commun. ACM* 13.2, pp. 94–102. DOI: 10.1145/362007.362035. URL: https://doi.org/10.1145/362007.362035.

Espinosa, Kurt J. P. et al. (2022). "Comparing neural models for nested and overlapping biomedical event detection". In: *BMC Bioinform.* 23.1, p. 211. DOI: 10.1186/s12859-022-04746-3. URL: https://doi.org/10.1186/s12859-022-04746-3.

Espinosa, Kurt Junshean, Makoto Miwa, and Sophia Ananiadou (2019). "A Search-based Neural Model for Biomedical Nested and Overlapping Event Detection". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 3677–3684. DOI: 10.18653/v1/D19-1381. URL: https://doi.org/10.18653/v1/D19-1381.

Evang, Kilian (2011). "Parsing Discontinuous Constituents in English". In: *Mémoire de master, University of Tübingen*.

Evang, Kilian and Laura Kallmeyer (2011). "PLCFRS Parsing of English Discontinuous Constituents". In: *Proceedings of the 12th International Conference on Parsing Technologies*, pp. 104–116. URL: https://www.aclweb.org/anthology/W11-2913/.

Fancellu, Federico et al. (2019). "Semantic graph parsing with recurrent neural network DAG grammars". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 2769–2778. DOI: 10.18653/v1/D19-1278. URL: https://doi.org/10.18653/v1/D19-1278.

Fancellu, Federico et al. (2020). "Accurate Polyglot Semantic Parsing With DAG Grammars". In: *Findings of the Association for Computational Linguistics: EMNLP*. Vol. EMNLP 2020. Findings of ACL, pp. 3567–3580. DOI: 10.18653/v1/2020.findings-emnlp.320. URL: https://doi.org/10.18653/v1/2020.findings-emnlp.320.

Fernández-González, Daniel and Carlos Gómez-Rodríguez (2019). "Faster shift-reduce constituent parsing with a non-binary, bottom-up strategy". In: *Artificial Intelligence* 275, pp. 559–574. DOI: 10.1016/j.artint.2019.07.006. URL: https://doi.org/10.1016/j.artint.2019.07.006.

— (2021). "Reducing Discontinuous to Continuous Parsing with Pointer Network Reordering". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 10570–10578. DOI: 10.18653/v1/2021.emnlp-main.825. URL: https://doi.org/10.18653/v1/2021.emnlp-main.825.

— (2022). "Multitask Pointer Network for multi-representational parsing". In: *Knowledge-Based Systems* 236, p. 107760. DOI: 10.1016/j.knosys.2021.107760. URL: https://doi.org/10.1016/j.knosys.2021.107760.

Ficler, Jessica and Yoav Goldberg (2016). "Improved Parsing for Argument-Clusters Coordination". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. DOI: 10.18653/v1/p16-2012. URL: https://doi.org/10.18653/v1/p16-2012.

Goldberg, Yoav and Michael Elhadad (2010). "An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing". In: *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics*, pp. 742–750. URL: https://aclanthology.org/N10-1115/.

Gómez-Rodríguez, Carlos and David Vilares (2018). "Constituent Parsing as Sequence Labeling". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1314–1324. DOI: 10.18653/v1/d18-1162. URL: https://doi.org/10.18653/v1/d18-1162.

Hershcovich, Daniel, Omri Abend, and Ari Rappoport (2017). "A Transition-Based Directed Acyclic Graph Parser for UCCA". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1127–1138. DOI: 10.18653/v1/P17-1104. URL: https://doi.org/10.18653/v1/P17-1104.

Hinds, John V (1973). "On the Status of the VP Node in Japanese". In: ☒☒☒☒.

Htut, Phu Mon, Kyunghyun Cho, and Samuel R. Bowman (2018). "Grammar Induction with Neural Language Models: An Unusual Replication". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 4998–5003. URL: https://aclanthology.org/D18-1544/.

Hudson, Richard (2021). "Word Grammar". In: *The Routledge Handbook of Cognitive Linguistics*. Routledge, pp. 111–126.

Japanese Language, National Institute for and Linguistics (2016). *NINJAL Parsed Corpus of Modern Japanese*. http://npcmj.ninjal.ac.jp/.

Joshi, Aravind K. (1983). "Factoring Recursion and Dependencies: an Aspect of Tree Adjoining Grammars (Tag) and a Comparison of Some Formal Properties of Tags, GPSGs, Plgs, and LPGS". In: *21st Annual Meeting of the Association for Computational Linguistics*, pp. 7–15. DOI: 10.3115/981311.981314. URL: https://aclanthology.org/P83-1002/.

Kamimura, Tsutomu and Giora Slutzki (1979). "DAGs and Chomsky Hierarchy (Extended Abstract)". In: *Automata, Languages and Programming, 6th Colloquium*. Vol. 71. Lecture Notes in Computer Science. Springer, pp. 331–337. DOI: 10.1007/3-540-09510-1_26. URL: https://doi.org/10.1007/3-540-09510-1_26.

Kasami, Tadao (1966). "An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages". In: *Coordinated Science Laboratory Report no. R-257*.

Keung, Phillip et al. (2020). "The Multilingual Amazon Reviews Corpus". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pp. 4563–4568. DOI: 10.18653/v1/2020.emnlp-main.369. URL: https://doi.org/10.18653/v1/2020.emnlp-main.369.

Kim, Yoon et al. (2019). "Unsupervised Recurrent Neural Network Grammars". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1105–1117. DOI: 10.18653/v1/n19-1114. URL: https://doi.org/10.18653/v1/n19-1114.

Kitaev, Nikita, Steven Cao, and Dan Klein (2019). "Multilingual Constituency Parsing with Self-Attention and Pre-Training". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pp. 3499–3505. DOI: 10.18653/v1/p19-1340. URL: https://doi.org/10.18653/v1/p19-1340.

Kitaev, Nikita and Dan Klein (2018). "Constituency Parsing with a Self-Attentive Encoder". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 2676–2686. DOI: 10.18653/v1/P18-1249. URL: https://www.aclweb.org/anthology/P18-1249/.

— (2020). "Tetra-Tagging: Word-Synchronous Parsing with Linear-Time Inference". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6255–6261. URL: https://www.aclweb.org/anthology/2020.acl-main.557/.

Klein, Dan and Christopher D. Manning (2003). "Accurate Unlexicalized Parsing". In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, 7-12 July 2003, Sapporo Convention Center, Sapporo, Japan*, pp. 423–430. DOI: 10.3115/1075096.1075150. URL: https://aclanthology.org/P03-1054/.

Knuth, Donald and Luis Trabb Pardo (1980). "The Early Development of Programming Languages". In: *A History of Computing in the Twentieth Century*, pp. 197–273.

Knuth, Donald E. (1965). "On the Translation of Languages from Left to Right". In: *Information and Control* 8.6, pp. 607–639. DOI: 10.1016/S0019-9958(65)90426-2. URL: https://doi.org/10.1016/S0019-9958(65)90426-2.

Lewis, Philip M and Richard Edwin Stearns (1968). "Syntax-directed Transduction". In: *Journal of the ACM (JACM)* 15.3, pp. 465–488.

Li, Jun et al. (2020). "An Empirical Comparison of Unsupervised Constituency Parsing Methods". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 3278–3283. DOI: 10.18653/v1/2020.acl-main.300. URL: https://doi.org/10.18653/v1/2020.acl-main.300.

Ling, Wang et al. (2015). "Two/Too Simple Adaptations of Word2Vec for Syntax Problems". In: *The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1299–1304. DOI: 10.3115/v1/n15-1142. URL: https://doi.org/10.3115/v1/n15-1142.

Little, Nathaniel (2010). "Reevaluating Gapping in CCG: Evidence from English & Chinese". In: *Proceedings of the 10th International Workshop on Tree Adjoining Grammar and Related Frameworks*, pp. 25–34. URL: https://aclanthology.org/W10-4404/.

Liu, Jiangming and Yue Zhang (2017). "Shift-Reduce Constituent Parsing with Neural Lookahead Features". In: *Transactions of the Association for Computational Linguistics* 5, pp. 45–58. URL: https://transacl.org/ojs/index.php/tacl/article/view/927.

Lucas, Peter (1978). "On the Formalization of Programming Languages: Early History and Main Approaches". In: *The Vienna Development Method: The Meta-Language*. Vol. 61. Lecture Notes in Computer Science. Springer, pp. 1–23. DOI: 10.1007/3-540-08766-4_8. URL: https://doi.org/10.1007/3-540-08766-4_8.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz (1993). "Building a Large Annotated Corpus of English: The Penn Treebank". In: *Computational Linguistics* 19.2, pp. 313–330.

Marcus, Mitchell P. et al. (1994). "The Penn Treebank: Annotating Predicate Argument Structure". In: *Human Language Technology, Proceedings of a Workshop*. Morgan Kaufmann, pp. 114 – 119. URL: https://aclanthology.org/H94-1020/.

Markov, Andrey Andreyevich (1906). "Rasprostranenie zakona bol' shih chisel na velichiny, zavisyaschie drug ot druga". In: *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete* 15.135-156, p. 18.

— (1913). "Primer statisticheskogo issledovaniya nad tekstom "Evgeniya Onegina", illyustriruyuschij svyaz' ispytanij v cep' ". In: *Izvestiya Akademii Nauk* 7, p. 153.

McDonald, Ryan T. et al. (2005). "Non-Projective Dependency Parsing using Spanning Tree Algorithms". In: *Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 523–530. URL: https://aclanthology.org/H05-1066/.

Mikolov, Tomas et al. (2013). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26*, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.

Moore, Andrew and Jeremy Barnes (2021). "Multi-task Learning of Negation and Speculation for Targeted Sentiment Classification". In: *Proceedings of the 2021*

Conference of the North American Chapter of the Association for Computational Linguistics, pp. 2838–2869. DOI: 10.18653/v1/2021.naacl-main.227. URL: https://doi.org/10.18653/v1/2021.naacl-main.227.

Mrini, Khalil et al. (2020). "Rethinking Self-Attention: Towards Interpretability in Neural Parsing". In: *Findings of the Association for Computational Linguistics: EMNLP*, pp. 731–742. DOI: 10.18653/v1/2020.findings-emnlp.65. URL: https://doi.org/10.18653/v1/2020.findings-emnlp.65.

Munikar, Manish, Sushil Shakya, and Aakash Shrestha (2019). "Fine-grained Sentiment Classification using BERT". In: *CoRR* abs/1910.03474. arXiv: 1910.03474. URL: http://arxiv.org/abs/1910.03474.

Nguyen, Thanh-Tung et al. (2020). "Efficient Constituency Parsing by Pointing". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 3284–3294. DOI: 10.18653/v1/2020.acl-main.301. URL: https://doi.org/10.18653/v1/2020.acl-main.301.

Nivre, Joakim, Marco Kuhlmann, and Johan Hall (2009). "An Improved Oracle for Dependency Parsing with Online Reordering". In: *Proceedings of the 11th International Workshop on Parsing Technologies*, pp. 73–76. URL: https://aclanthology.org/W09-3811/.

Oettinger, Anthony (1961). *Automatic Syntactic Analysis and the Pushdown Store*. American Mathematical Society.

Pak, Alexander and Patrick Paroubek (2010). "Twitter as a Corpus for Sentiment Analysis and Opinion Mining". In: *Proceedings of the International Conference on Language Resources and Evaluation*. European Language Resources Association. URL: http://www.lrec-conf.org/proceedings/lrec2010/summaries/385.html.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543. DOI: 10.3115/v1/d14-1162. URL: https://doi.org/10.3115/v1/d14-1162.

Pullum, Geoffrey K. and Gerald Gazdar (1982). "Natural Languages and Context-Free Languages". In: *Linguistics and Philosophy* 4.4, pp. 471–504. ISSN: 01650157, 15730549. URL: http://www.jstor.org/stable/25001071 (visited on 10/27/2022).

Pustejovsky, James (1998). *The Generative Lexicon*. MIT Press. ISBN: 978-0-262-66140-9. URL: http://mitpress.mit.edu/books/generative-lexicon.

Pyysalo, Sampo et al. (2015). "Overview of the cancer genetics and pathway curation tasks of bionlp shared task 2013". In: *BMC bioinformatics* 16.10, pp. 1–19.

Ratnaparkhi, Adwait (1997). "A Linear Observed Time Statistical Parser Based on Maximum Entropy Models". In: *Second Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/W97-0301/.

Ruprecht, Thomas 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*, pp. 2923–2935. DOI: 10.18653/v1/2021.naacl-main.232. URL: https://doi.org/10.18653/v1/2021.naacl-main.232.

Sagae, Kenji and Jun'ichi Tsujii (2008). "Shift-Reduce Dependency DAG Parsing". In: *COLING 2008, 22nd International Conference on Computational Linguistics, Proceedings of the Conference*, pp. 753–760. URL: https://aclanthology.org/C08-1095/.

Sakai, Itiroo (1961). "Syntax in Universal Translation". In: *Proceedings of the International Conference on Machine Translation and Applied Language Analysis*.

Seddah, Djamé et al. (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*. Association for Computational Linguistics, pp. 146–182. URL: https://aclanthology.org/W13-4917/.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016a). "Improving Neural Machine Translation Models with Monolingual Data". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. DOI: 10.18653/v1/p16-1009. URL: https://doi.org/10.18653/v1/p16-1009.

— (2016b). "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 1715 —1725. URL: https://www.aclweb.org/anthology/P16-1162/.

Shannon, Claude E. (1948). "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27.3, pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x. URL: https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

Shen, Yikang et al. (2018a). "Neural Language Modeling by Jointly Learning Syntax and Lexicon". In: *6th International Conference on Learning Representations*. URL: https://openreview.net/forum?id=rkgOLb-0W.

Shen, Yikang et al. (2018b). "Straight to the Tree: Constituency Parsing with Neural Syntactic Distance". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 1171–1180. DOI: 10.18653/v1/P18-1108. URL: https://www.aclweb.org/anthology/P18-1108/.

Shen, Yikang et al. (2019). "Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks". In: *7th International Conference on Learning Representations*. URL: https://openreview.net/forum?id=B1l6qiR5F7.

Socher, Richard et al. (2013a). "Parsing with Compositional Vector Grammars". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pp. 455–465. URL: https://www.aclweb.org/anthology/P13-1045/.

Socher, Richard et al. (2013b). "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642. URL: https://www.aclweb.org/anthology/D13-1170/.

Stanojevic, Milos 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*, pp. 111–121. URL: https://www.aclweb.org/anthology/2020.iwpt-1.12/.

Steedman, Mark (1997). *Surface Structure and Interpretation*. Vol. 30. Linguistic inquiry. MIT Press. ISBN: 978-0-262-69193-2.

— (2004). *The Syntactic Process*. Language, speech, and communication. MIT Press. ISBN: 978-0-262-69268-7.

Stern, Mitchell, 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*, pp. 818–827. DOI: 10.18653/v1/P17-1076. URL: https://doi.org/10.18653/v1/P17-1076.

Sun, Zijun et al. (2020). "Self-Explaining Structures Improve NLP Models". In: *CoRR* abs/2012.01786. arXiv: 2012.01786. URL: https://arxiv.org/abs/2012.01786.

Tai, Kai Sheng, Richard Socher, and Christopher D. Manning (2015). "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language*

Processing of the Asian Federation of Natural Language Processing, pp. 1556–1566. URL: https://www.aclweb.org/anthology/P15-1150/.

Tesnière, Lucien (1959). "Elements of Structural Syntax (Éléments de syntaxe structural), Klincksieck, Paris. Préface by Jean Fourquet, professeur à la Sorbonne. reviewed and corrected. ISBN 2-252-02620-0. Re-edition of: Tesniere". In.

Tokgöz, Alper and Gülsen Eryigit (2015). "Transition-based Dependency DAG Parsing Using Dynamic Oracles". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pp. 22–27. DOI: 10.3115/v1/p15-3004. URL: https://doi.org/10.3115/v1/p15-3004.

Versley, Yannick (2014). "Incorporating Semi-supervised Features into Discontinuous Easy-First Constituent Parsing". In: *CoRR* abs/1409.3813. arXiv: 1409.3813. URL: http://arxiv.org/abs/1409.3813.

Vijay-Shanker, K., David J. Weir, and Aravind K. Joshi (1987). "Characterizing Structural Descriptions produced by Various Grammatical Formalisms". In: *25th Annual Meeting of the Association for Computational Linguistics*, pp. 104–111. DOI: 10.3115/981175.981190. URL: https://aclanthology.org/P87-1015/.

Vilares, David and Carlos Gómez-Rodríguez (2020). "Discontinuous Constituent Parsing as Sequence Labeling". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 2771–2785. DOI: 10.18653/v1/2020.emnlp-main.221. URL: https://doi.org/10.18653/v1/2020.emnlp-main.221.

Watanabe, Taro and Eiichiro Sumita (2015). "Transition-based Neural Constituent Parsing". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pp. 1169–1179. URL: https://www.aclweb.org/anthology/P15-1113/.

Wei, Yang, Yuanbin Wu, and Man Lan (2020). "A Span-based Linearization for Constituent Trees". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 3267–3277. DOI: 10.18653/v1/2020.acl-main.299. URL: https://doi.org/10.18653/v1/2020.acl-main.299.

Weir, David Jeremy (1988). *Characterizing Mildly Context-sensitive Grammar Formalisms*. University of Pennsylvania.

Xin, Xin, Jinlong Li, and Zeqi Tan (2021). "N-ary Constituent Tree Parsing with Recursive Semi-Markov Model". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pp. 2631–2642. DOI: 10.18653/v1/2021.acl-long.205. URL: https://doi.org/10.18653/v1/2021.acl-long.205.

Xue, Nianwen et al. (2000). "The bracketing guidelines for the Penn Chinese Treebank (3.0)". In: *IRCS Technical Reports Series*, p. 39.

Yang, Kaiyu and Jia Deng (2020). "Strongly Incremental Constituency Parsing with Graph Neural Networks". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*. URL: https://proceedings.neurips.cc/paper/2020/hash/f7177163c833dff4b38fc8d2872f1ec6-Abstract.html.

Yang, Zhilin et al. (2019). "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *CoRR* abs/1906.08237. URL: http://arxiv.org/abs/1906.08237.

Younger, Daniel H. (1967). "Recognition and Parsing of Context-Free Languages in Time n^3". In: *Information and control* 10.2, pp. 189–208. DOI: 10.1016/S0019-9958(67)80007-X. URL: https://doi.org/10.1016/S0019-9958(67)80007-X.

Zhang, Yu, Houquan Zhou, and Zhenghua Li (2020). "Fast and Accurate Neural CRF Constituency Parsing". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pp. 4046–4053. DOI: 10.24963/ijcai.2020/560. URL: https://doi.org/10.24963/ijcai.2020/560.

Zhao, Yang et al. (2018). "Addressing Troublesome Words in Neural Machine Translation". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 391–400. URL: https://aclanthology.info/papers/D18-1036/d18-1036.

Zhou, Junru and Hai Zhao (2019). "Head-Driven Phrase Structure Grammar Parsing on Penn Treebank". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pp. 2396–2408. URL: https://www.aclweb.org/anthology/P19-1230/.

Zwicky, Arnold M. (1985). "Heads". In: *Journal of Linguistics* 21.1, pp. 1 – 29. DOI: 10.1017/S0022226700010008.