Discontinuous Constituency Parsing of Morphologically Rich Languages

Maximin Coavoux

USPC
Université Sorbonne
Paris Cité

université
PARIS
DIDEROT
PARIS 7

Thèse de doctorat

de l'Université Sorbonne Paris Cité

Préparée à l'Université Paris Diderot

École doctorale Sciences du Langage (ED 132)

Laboratoire de Linguistique Formelle

Discontinuous Constituency Parsing of Morphologically Rich Languages

par Maximin Coavoux

Thèse de doctorat de linguistique théorique,

descriptive et automatique

Dirigée par Benoît Crabbé

Présentée et soutenue publiquement
à Paris le 11 décembre 2017

Rapporteur·e·s :
Claire GARDENT, Directrice de recherche, CNRS Nancy
Alexis NASR (Président du jury), Professeur, Aix Marseille Université
Examinateurs :
Carlos GÓMEZ-RODRÍGUEZ, Associate Professor, Universidade da Coruña
Alexandre ALLAUZEN, Maître de conférences, Université Paris-Sud
Directeur de thèse :
Benoît CRABBÉ, Maître de conférences, Université Paris Diderot

Titre : Analyse syntaxique automatique en constituants discontinus des langues à morphologie riche.

Résumé : L'analyse syntaxique consiste à prédire la représentation syntaxique de phrases en langue naturelle sous la forme d'arbres syntaxiques. Cette tâche pose des problèmes particuliers pour les langues non-configurationnelles ou qui ont une morphologie flexionnelle plus riche que celle de l'anglais. En particulier, ces langues manifestent une dispersion lexicale problématique, des variations d'ordre des mots plus fréquentes et nécessitent de prendre en compte la structure interne des mots-formes pour permettre une analyse syntaxique de qualité satisfaisante.

Dans cette thèse, nous nous plaçons dans le cadre de l'analyse syntaxique robuste en constituants par transitions. Dans un premier temps, nous étudions comment intégrer l'analyse morphologique à l'analyse syntaxique, à l'aide d'une architecture de réseaux de neurones basée sur l'apprentissage multi-tâches. Dans un second temps, nous proposons un système de transitions qui permet de prédire des structures générées par des grammaires légèrement sensibles au contexte telles que les LCFRS. Enfin, nous étudions la question de la lexicalisation de l'analyse syntaxique. Les analyseurs syntaxiques en constituants lexicalisés font l'hypothèse que les constituants s'organisent autour d'une tête lexicale et que la modélisation des relations bilexicales est cruciale pour désambiguïser. Nous proposons un système de transition non lexicalisé pour l'analyse en constituants discontinus et un modèle de scorage basé sur les frontières de constituants et montrons que ce système, plus simple que des systèmes lexicalisés, obtient de meilleurs résultats que ces derniers.

Mots clefs : Traitement automatique des langues naturelles, analyse syntaxique automatique, arbres en constituants discontinus, systèmes de transitions, apprentissage profond, apprentissage multi-tâches.

Title: Discontinuous Constituency Parsing of Morphologically Rich Languages.

Abstract: Syntactic parsing consists in assigning syntactic trees to sentences in natural language. Syntactic parsing of non-configurational languages, or languages with a rich inflectional morphology, raises specific problems. These languages suffer more from lexical data sparsity and exhibit word order variation phenomena more frequently. For these languages, exploiting information about the internal structure of word forms is crucial for accurate parsing.

This dissertation investigates transition-based methods for robust discontinuous constituency parsing. First of all, we propose a multitask learning neural architecture that performs joint parsing and morphological analysis. Then, we introduce a new transition system that is able to predict discontinuous constituency trees, i.e. syntactic structures that can be seen as derivations of mildly context-sensitive grammars, such as LCFRS. Finally, we investigate the question of lexicalization in syntactic parsing. Some syntactic parsers are based on the hypothesis that constituents are organized around a lexical head and that modelling bilexical dependencies is essential to solve ambiguities. We introduce an unlexicalized transition system for discontinuous constituency parsing and a scoring model based on constituent boundaries. The resulting parser is simpler than lexicalized parser and achieves better results in both discontinuous and projective constituency parsing.

Keywords: Natural language processing, syntactic parsing, discontinuous constituency trees, transition systems, deep learning, multitask learning.

Contents

1 Introduction **14**
1.1 Desiderata for Large-Scale Parsing 15
1.2 Parsing Morphologically Rich Languages 17
1.2.1 Solving Ambiguities 17
1.2.2 Modelling the Morphology-Syntax Interface 18
1.2.3 Predicting Discontinuous Constituency Trees 18
1.2.4 Learning Bilexical Relations 19
1.3 Contributions and Outline 19
1.3.1 A Dynamic Oracle to Deal with Locality Biases 20
1.3.2 Multitask Learning for Parsing and Morphological Analysis 20
1.3.3 Transition Systems for Discontinuous Parsing 21
1.3.4 Unlexicalized Systems for Constituency Parsing 21
1.3.5 Outline 21
1.3.6 Publications 23

I Background **24**

2 Probabilistic Chart Parsing **25**
2.1 Introduction: Parsing Paradigms 25
2.2 Syntactic Representations and Formalisms 26
2.2.1 Constituency and Dependency Trees 26
2.2.2 Representing Non-Local Dependencies 26
2.3 Chart Parsing 28
2.3.1 Probabilistic Models for CFG Parsing 29
2.3.1.1 Probabilistic Context-Free Grammar 29
Model Definition 29
Inference 30
The Limitations of PCFG 32
2.3.1.2 Lexicalized PCFG 33
2.3.1.3 PCFG with Latent Annotations 34
Discriminative Chart Parsing 36

2.3.2 Probabilistic LCFRS 36
2.3.2.1 Linear Context-Free Rewriting Systems 36
2.3.2.2 Properties of an LCFRS 38
2.3.2.3 Parsing with Probabilistic LCFRS 38
2.3.2.4 A Related Formalism: Discontinuous Data-Oriented Parsing 39
2.4 Conclusion 39

3 Transition-based Parsing 40
3.1 Introduction 40
3.1.1 Parsing as a Search Problem 41
3.1.2 Deterministic Parsing 42
3.2 Transition Systems 43
3.2.1 A Lexicalized Shift-Reduce Transition System 44
3.2.1.1 Set of Transitions 46
3.2.1.2 The Derivation Length Problem 46
3.2.1.3 Preconditions on Actions 47
3.2.1.4 Relationship with the Arc-Standard System 49
3.2.1.5 Related Transition Systems 49
3.2.2 Discontinuous Constituency Parsing Transition Systems . 50
3.2.2.1 The Reduction to Dependency Parsing Approach . . . 50
3.2.2.2 The Easy-First Transition System 51
3.2.2.3 The Shift-Reduce-Swap Transition System 52
Oracles and Terminal Reordering 53
Derivation Lengths 53
3.2.3 Preprocessing the Trees 54
3.2.3.1 Binarization 54
3.2.3.2 Merging Unary Productions 57
3.2.3.3 Head Annotation 58
3.2.3.4 Tree Transformations and Grammar Inflation 58
3.3 Conclusion 62

4 Weighted Parsing: Decoding and Learning 63
4.1 Introduction 64
4.2 Searching for the Best Derivation 65
4.2.1 Structure of the Search Space 65
4.2.2 Search Algorithms 65
4.2.2.1 Greedy Search 67
Definition and Complexity 67
Locality Biases: Garden Path Effects 67
4.2.2.2 Beam Search 68

4.2.2.3 Best-First Search 69
4.2.3 Interim Conclusion 70
4.3 A Linear Classifier: the Structured Perceptron 70
4.3.1 The Structured Perceptron Algorithm 71
4.3.2 Mapping Derivations to Vectors 72
4.3.3 Training a Structured Perceptron with Inexact Search . . . 75
4.4 Structured Prediction with Local Classifiers 76
4.5 Neural Models 77
4.5.1 Lexical Data Dispersion 77
4.5.2 Feed-Forward Neural Networks 78
4.5.2.1 Mapping Configurations to Vectors 78
4.5.2.2 A Feed-Forward Architecture 79
4.5.2.3 Objective Function and Training 80
4.5.2.4 Discussion 80
4.5.3 Bidirectional Recurrent Neural Networks 81
4.5.3.1 Building Context-aware Embeddings with a Bi-RNN . 82
Motivations 82
Recurrent Neural Network Encoder 82
Bidirectional RNN 83
4.5.3.2 Predicting Actions 83
4.6 Conclusion 85

II Projective Constituency Parsing 86

5 Parsing with a Dynamic Oracle 87
5.1 Introduction 87
5.2 Training a Parser with an Oracle 89
5.2.1 Limitations of Static Oracles 89
5.2.2 Learning with Exploration 90
5.3 A Dynamic Oracle for Lexicalized Shift-Reduce Parsing 92
5.3.1 Preliminary Definitions 92
5.3.1.1 Transition System 93
5.3.1.2 Constituent Decomposability 93
5.3.2 Cost Function 94
5.3.2.1 Cost for a Tree 94
5.3.2.2 Cost for a Transition 95
5.3.3 Finding 0-Cost Actions 97
5.3.3.1 Constituent Reachability 97
5.3.3.2 Constituent Decomposability 98
5.3.3.3 Computing the Cost of a Transition 99

5.3.4 An Oracle for SR-TMP . . . 99
Head Choice . . . 101
5.4 Experiments . . . 102
5.4.1 Experimental setting . . . 102
5.4.1.1 Datasets . . . 102
5.4.1.2 Classifier: Feed-Forward Neural Network . . . 102
Feature Templates . . . 102
Training . . . 103
5.4.2 Results . . . 104
5.4.2.1 Effect of the Dynamic Oracle . . . 104
5.4.2.2 Combined Effect of Beam Search and Dynamic Oracle 106
5.5 Conclusion . . . 107

6 Morphology and Syntactic Functions 108
6.1 Introduction . . . 108
6.2 Multitask Learning . . . 110
6.3 A Multitask Learning Architecture for Parsing and Tagging . . 113
6.3.1 A Shared Hierarchical Bidirectional LSTM Encoder . . . 114
6.3.2 Tagging Component . . . 115
6.3.3 Parsing Component . . . 115
6.3.4 Objective Function and Training . . . 116
6.4 Experiments . . . 117
6.4.1 Datasets . . . 118
6.4.2 Protocol . . . 118
6.4.3 Results and Discussion . . . 120
6.4.3.1 Effect of Tagging Auxiliary Tasks . . . 121
Morphological Analysis . . . 121
Functional Labelling . . . 121
6.4.3.2 Comparison with Existing Results . . . 121
Constituency Trees . . . 121
Labelled Dependency Trees . . . 121
6.4.3.3 Limitation and Perspective . . . 122
6.5 Conclusion . . . 123

III Discontinuous Constituency Parsing 124

7 Discontinuous Constituency Parsing 125
7.1 Introduction . . . 125
7.2 The Shift-Reduce-Gap Transition System . . . 127
7.2.1 Algorithm . . . 127

7.2.1.1 An Example Run . . . 130
7.2.1.2 Interpretation of the GAP Action . . . 131
7.2.1.3 Relationship with Covington's Algorithm . . . 131
7.2.1.4 Preconditions on Actions . . . 132
7.2.2 Oracle and Properties . . . 133
7.2.2.1 Preliminary Definitions . . . 134
7.2.2.2 Oracle . . . 134
7.2.2.3 Determinism of the Oracle . . . 134
7.2.2.4 Completeness and Soundness of SR-GAP . . . 135
7.2.3 Comparing Derivations with Different Lengths . . . 136
7.3 Experiments . . . 138
7.3.1 Datasets . . . 139
7.3.1.1 Corpora . . . 139
7.3.1.2 Preprocessing . . . 139
7.3.2 Classifier: Structured Perceptron . . . 139
7.3.3 Results . . . 140
7.3.3.1 Internal Comparisons . . . 142
7.3.3.2 External Comparisons . . . 143
Model Analysis . . . 144
7.4 Discussion: a Comparison of SR-SWAP and SR-GAP . . . 145
7.4.1 Derivation Length . . . 145
7.4.2 Feature Semantics and Feature Locality . . . 148
7.5 Conclusion . . . 150

8 Unlexicalized Constituency Parsing 151
8.1 Lexicalization in Statistical Parsing . . . 152
8.1.1 Lexicalized Chart Parsers . . . 153
8.1.2 Unlexicalized Chart Parsers . . . 153
8.1.3 Unlexicalized Transition-based Parsers . . . 154
8.1.4 Contributions . . . 154
8.2 Structure-Label Transitions Sytems . . . 155
8.2.1 Merge-Label-Gap . . . 155
8.2.2 Lexicalized Merge-Label-Gap . . . 159
8.2.3 Properties . . . 160
8.2.3.1 Oracles . . . 160
8.2.3.2 Number of Action Types . . . 162
8.2.3.3 Number of GAP Actions . . . 163
8.2.3.4 Incrementality . . . 165
8.2.3.5 Interim Conclusion . . . 165
8.3 Experiments . . . 166
8.3.1 Statistical Model . . . 167

8.3.2 Feature Templates . . . 167
8.3.3 Data . . . 168
Discontinuous Treebanks . . . 168
Projective Treebanks . . . 169
8.3.4 Results . . . 169
8.3.4.1 Multilingual Discontinuous Constituency Parsing . . 169
Internal Comparisons . . . 169
Morphological Analysis . . . 170
External Comparisons . . . 172
8.3.4.2 Multilingual Projective Constituency Parsing . . . 173
8.4 Error Analysis . . . 174
8.4.1 Methodology . . . 174
8.4.2 Observations . . . 176
8.4.2.1 Wh-extractions . . . 176
Non Detections . . . 176
Partial recognitions . . . 176
False positives . . . 179
8.4.2.2 Fronted Quotations . . . 179
8.4.2.3 Extrapositions . . . 180
8.4.2.4 Circumpositioned Quotations . . . 182
8.4.2.5 It-extraposition . . . 185
8.4.2.6 Subject-verb Inversions . . . 185
8.4.3 Discussion . . . 186
8.5 Dynamic Programming . . . 186
8.5.1 Equivalent Parsing States . . . 187
8.5.2 Tree Structured Stack . . . 187
8.5.3 Encoding a TSS in the Parsing States . . . 189
8.5.3.1 Projective Case . . . 191
8.5.3.2 Discontinuous Case . . . 191
8.5.3.3 Graph-Structured Stack . . . 192
8.5.4 Towards Exhaustive Search with Dynamic Programming . 192
8.6 Conclusion . . . 193

9 Conclusion and Perspectives 195

List of Tables

2.1 Deduction rules for CFG parsing. 31
2.2 Deduction rules for LCFRS parsing. 38

3.1 Derivation example with shift-reduce. 41
3.2 Lexicalized shift-reduce transition system. 45
3.3 Derivation with a lexicalized shift-reduce transition system. 47
3.4 Derivation with the shift-reduce-swap transition system. . . 53
3.5 Symbol distribution in a treebank and preprocessing. 58

4.1 Lexical data sparsity in different treebanks. 78

5.1 Well-formedness constraints for binarized trees. 92
5.2 Dynamic oracle cost calculation (tree). 96
5.3 Dynamic oracle cost calculation (transition). 97
5.4 Feature templates. 103
5.5 Size of morphological attributes embeddings. 104
5.6 Hyperparameters. 104
5.7 Results on the Penn Treebank. 105
5.8 Results on the SPMRL dataset. 106

6.1 Word label matrix for a full sentence. 113
6.2 Summary of models. 117
6.3 Hyperparameters 119
6.4 Parsing results on the SPMRL dataset. 120
6.5 Dependency parsing and tagging results. 122

7.1 Lexicalized Shift-Reduce-Gap transition system. 129
7.2 Full derivation with SR-GAP. 131
7.3 Preconditions for SR-GAP actions. 133
7.4 Feature templates. 141
7.5 Results on development sets for different beam sizes. 142
7.6 Final test results. 143
7.7 Final results with predicted POS tags. 144
7.8 Parsing results on the Discontinuous Penn Treebank. 144

7.9 Detailed results for discontinuous constituents. 145
7.10 Derivation with SR-SWAP. 147
7.11 Statistics about derivations with SR-SWAP and SR-GAP. . . . 148

8.1 Structure-Label transition system (Cross and Huang, 2016a). 155
8.2 An unlexicalized transition system for discontinuous parsing.158
8.3 Example derivation with ML-GAP. 159
8.4 A lexicalized structure-label system. 161
8.5 Action type statistics per transition system and corpus. . . . 162
8.6 GAP action statistics in different treebanks. 163
8.7 Incrementality measure per transition system and corpus. . . 166
8.8 Feature template set descriptions. 168
8.9 Discontinuous parsing results (development). 170
8.10 Morphological analysis results. 171
8.11 Discontinuous parsing results (test). 172
8.12 Projective parsing results. 173
8.13 UAS results with ML-LEX (dev). 174
8.14 Precision and recall (development) for the selected model. . 175
8.15 Evaluation statistics per phenomenon. 177
8.16 Limitations of the model feature templates. 184

List of Figures

1.1 Syntactic analysis. 15
1.2 Prepositional attachment ambiguities. 17
1.3 Lexicalized Constituency Tree. 19

2.1 Constituency and dependency trees. 27
2.2 Representation strategies for non-local dependencies. 29
2.3 Probabilistic Context-Free Grammar. 30
2.4 Lexicalized constituency trees. 33
2.5 Discontinuous tree and extracted LCFRS. 35

3.1 Preprocessed constituency trees. 44
3.2 Impossible binarized trees. 48
3.3 Binarized discontinuous constituency tree. 52
3.4 Four binarization strategies. 55
3.5 Symbol distribution bar chart for a discontinuous treebank before and after preprocessing. 60
3.6 Symbol distribution bar chart for a projectivized treebank before and after preprocessing. 61

4.1 Search space illustration. 66
4.2 Locality of a configuration. 73
4.3 Instantiation of a sparse feature vector. 75
4.4 Bi-RNN illustration on a three-element sequence. 83
4.5 Feature instantiation for bi-RNN parsing. 84

5.1 Binarized Penn Treebank tree. 95
5.2 Difficult case for SR-TMP. 100

6.1 Tree from the French Question Bank. 110
6.2 Single-task learning vs multitask learning. 111
6.3 Representations learned by multitask learning. 112
6.4 Deep bi-LSTM encoder with auxiliary tasks. 115
6.5 Feature templates for bi-RNN parsing. 117

6.6 Learning curves for parsing and tagging. 123

7.1 Reduction actions with shift-reduce and SR-GAP. 127
7.2 Tree from the Discontinuous Penn Treebank. 130
7.3 Illustration of the order $\prec$ on nodes. 135
7.4 Automata of licit action sequences for SR-GAP and SR-CGAP. 137
7.5 Abstract configuration. 140
7.6 Projectivization of a discontinuous tree. 146
7.7 Trees with longest derivations with SR-GAP and SR-SWAP. . . 148
7.8 Tree from the French Question Bank. 149

8.1 Automaton of licit action sequences. 157
8.2 Lexicalized tree and two different binarizations. 164
8.3 Partial recognition cases for wh-extractions: VP scope. 178
8.4 Partial recognition cases for wh-extractions: extraction site. . 180
8.5 Partial recognition cases for wh-extractions: shared dependents. 181
8.6 False positive wh-extractions. 181
8.7 Non-detected extrapositions. 182
8.8 Partially recognized circumpositioned quotations. 183
8.9 It-extraposition predictions. 185
8.10 Dynamic programming: state merging. 188
8.11 Tree-structured stack. 189
8.12 Search space and GSS. 190

List of Algorithms

1 CKY algorithm for CFG weighted parsing. 31
2 High-level transition-based parsing algorithm. 43
3 Left reordering algorithm. 54
4 Binarization algorithms. 56
5 Search algorithms for parsing. 69
6 Global averaged perceptron algorithm. 71
7 Online training with exploration. 91
8 Dynamic oracle algorithm (SR-BIN). 98
9 Dynamic oracle algorithm (SR-TMP). 101
10 Covington's algorithm. 132

Chapter 1

Introduction

Contents

1.1 Desiderata for Large-Scale Parsing **15**
1.2 Parsing Morphologically Rich Languages **17**
1.2.1 Solving Ambiguities 17
1.2.2 Modelling the Morphology-Syntax Interface 18
1.2.3 Predicting Discontinuous Constituency Trees 18
1.2.4 Learning Bilexical Relations 19
1.3 Contributions and Outline **19**
1.3.1 A Dynamic Oracle to Deal with Locality Biases 20
1.3.2 Multitask Learning for Parsing and Morphological Analysis 20
1.3.3 Transition Systems for Discontinuous Parsing 21
1.3.4 Unlexicalized Systems for Constituency Parsing . . . 21
1.3.5 Outline 21
1.3.6 Publications 23

This dissertation is about robust syntactic parsing of languages with a rich inflectional morphology. We propose a method for large-scale parsing of these languages with three key properties. First of all, the method is efficient and achieves empirical parsing time linear in the length of a sentence. Secondly, it integrates morphological and functional analysis into parsing. Finally, it is complete in the sense that it is capable of predicting

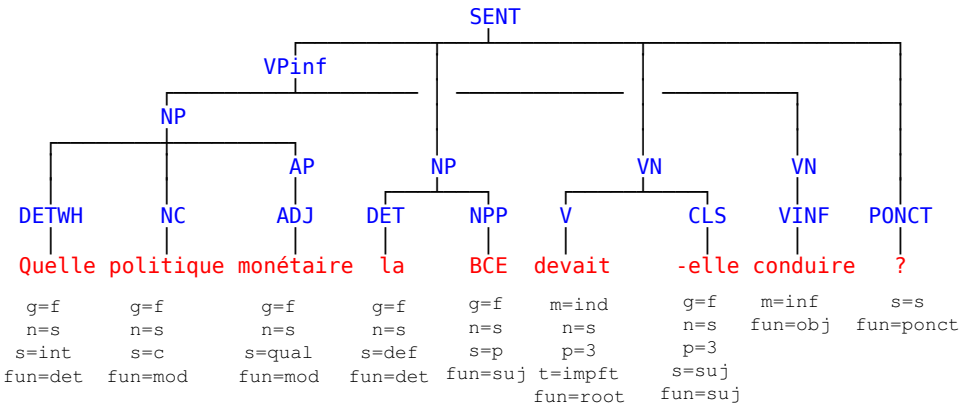


Figure 1.1: Syntactic analysis. *Which monetary policy should the ECB lead?* Notations: (g)ender, (s)ubcat, (n)umber, (m)ood, (t)ense, (p)erson, (fun)ction.

discontinuous constituency structures, and thus model long-distance dependencies that projective constituency trees do not represent adequately. The type of analysis we are interested in is illustrated in Figure 1.1, where a long-distance extraction is represented with a discontinuous VPinf, and each word is associated with its morphological analysis, including part-of-speech tag and morphological attributes, and its syntactic function (e.g. subject, object).

In the remainder of this introduction, we motivate the choice of the framework of transition-based parsing by presenting some desiderata for large scale discontinuous constituency parsing (Section 1.1). Then, we describe typical issues in parsing morphologically rich languages (Section 1.2) and how our contributions address them (Section 1.3).

1.1 Desiderata for Large-Scale Parsing

A syntactic parser needs to be **robust**. Robustness is the capability to output a syntactic analysis even for an utterance that is not grammatically well-formed. Most actual language productions, textual or oral, contain a degree of non-canonicity, including disfluencies, non-sentential fragments, spelling or pronunciation mistakes, agreement errors. Yet, those productions are interpretable nonetheless. In that respect, robust parsing is not a recognition problem: it does not say anything about the grammaticality of a sentence. Robustness is a desirable property of parsers, because

typical applications use actual language production that can be very noisy, such as user-generated contents.

Another desideratum for a syntactic parser aiming at analysing a lot of textual data is **efficiency**. Chart parsers for Probabilistic Context-Free Grammars (PCFG) based on an exhaustive search have a complexity in $\mathcal{O}(n^3)$, which is unpractical for long sentences. In contrast, the methods we used in this dissertation run in linear time and scale easily to sentences of any length.

These properties are required by most applications of parsing, such as:

- **Building semantic representations**. Syntactic parsing is a first step towards building a semantic representation of a sentence or towards deeper understanding of documents. For example, semantic parsers have been shown to perform much better with features extracted from syntactic trees (Ribeyre et al., 2015). Syntactic trees are also helpful for discourse parsing (Feng and Hirst, 2012) and to identify discourse relations (Pitler and Nenkova, 2009).

- **Constructing resources automatically**. Syntactic parsers are used to automatically construct new corpora with predicted syntactic annotations (Seddah et al., 2012), which may be used either for unsupervised learning (Levy and Goldberg, 2014) or linguistic studies (Thuilier, 2012).

- **Psycholinguistic modelling**. Although they are not designed for this purpose in the first place, parsers may be models of human syntactic processing. In particular, incremental parsers have interesting properties for psycholinguistic modelling (Hale, 2001; Roark et al., 2009; Hale, 2014). Moreover, the properties we described as desiderata for parsers are also taken to be properties of human sentence processing: efficiency, robustness (Crocker, 1999).

In this dissertation, we focus on transition-based parsing. Transition-based parsing decomposes a tree in a sequence of operations that can be seen as a linearization of the tree. It casts the parsing problem as a sequence prediction problem. Transition-based parsing is inherently robust, as it does not usually rely on an explicit grammar. Transition-based methods are also generally more efficient as they support linear time decoding algorithms, such as greedy search and beam search. The experiments presented in this dissertation confirm that very simple approximate search methods can lead to very high results provided that the scoring system uses adequate representations.

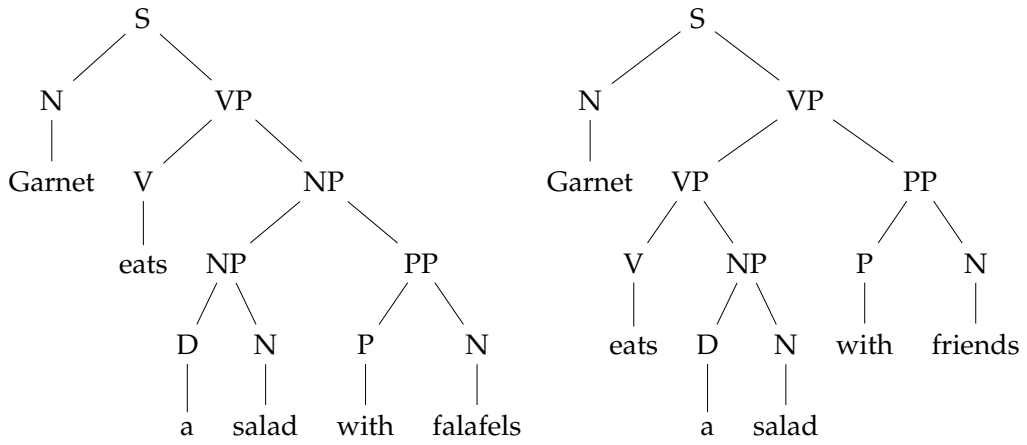


Figure 1.2: Prepositional attachment ambiguities.

1.2 Parsing Morphologically Rich Languages

This section presents some issues in constituency parsing. The general goal of parsing is solving structural ambiguities (Section 1.2.1). We do not address this problem directly, instead we focus on selected issues, which are prevalent when parsing morphologically rich languages: morphology (Section 1.2.2), discontinuities (Section 1.2.3) and bilexical dependency learning (Section 1.2.4).

1.2.1 Solving Ambiguities

The main difficulty of parsing is to solve the ambiguity exhibited by natural languages. For a single sentence, there are usually several different plausible syntactic analyses. Although some ambiguities are natural, most ambiguities are artificial. Most native speakers would not notice them as only one of the plausible analyses is prominent. Most attachment decisions are relatively easy to solve, e.g. finding which noun is the governor of a determiner, but some specific types of ambiguities are hard for a parser.

Typical artificial ambiguities are coordination ambiguities and prepositional attachment ambiguities. For example, in Figure 1.2, the prepositional phrases (PP) may be attached either as a modifier to the noun phrase (NP) or to the verb phrase (VP). The knowledge required to solve these ambiguities is very hard to learn for a parser. Due to attachment combinatorics, the number of possible parses for a single sentence can easily range in the millions.

For transition-based parsers that are incremental (and process the sentence left-to-right), the additional difficulty to solve ambiguities is that they might not have access to the relevant information, e.g. unprocessed words at the end of the sentence, at the time of the attachment decision. In other words, the parser is subject to a locality bias: it takes decisions based on local information, rather than the whole sentence.

1.2.2 Modelling the Morphology-Syntax Interface

Morphologically Rich Languages (MRL thereafter) raise specific problems in parsing (Seddah et al., 2013).[1]

They are more subject to lexical data sparsity, as a lexeme may have numerous different inflected forms. A lot of word forms in the test set will not have been seen during training and the parser will not have parameters for them. The treatment of these unknown words is crucial. Moreover, MRLs exhibit more flexible word order than configurational languages such as English. Important syntactic information is expressed through morphology. For example, the case of a noun determines its syntactic function, which is a very important cue to decide on its attachment in a syntactic tree. To recover the syntactic structure of a sentence, it is decisive to model the interaction between morphology and syntax. Most existing parsers for MRLs use a pipeline approach and rely on an external morphological tagger. Instead, we propose to model interaction between morphology and syntax with a multitask learning architecture (Chapter 6).

1.2.3 Predicting Discontinuous Constituency Trees

Projective constituency trees are not adequate to model a number of syntactic phenomena related to word order variations, e.g. long distance extractions. Although these phenomena also appear in configurational languages, they are more frequent in MRLs. However, discontinuous constituency parsing has attracted much less attention than projective constituency parsing so far, and it is a much more difficult task in terms of complexity. Existing results in discontinuous parsing are rather low compared to projective constituency parsing. In particular, discontinuous

[1] As Seddah et al. (2013), we take MRLs to denote languages with a richer inflectional morphology than English. This definition includes agglutinative languages (Hungarian, Korean), languages with some degree of word order flexibility (German, Polish), but also languages, such as French, that exhibit similar issues in parsing, such as lexical data sparsity.

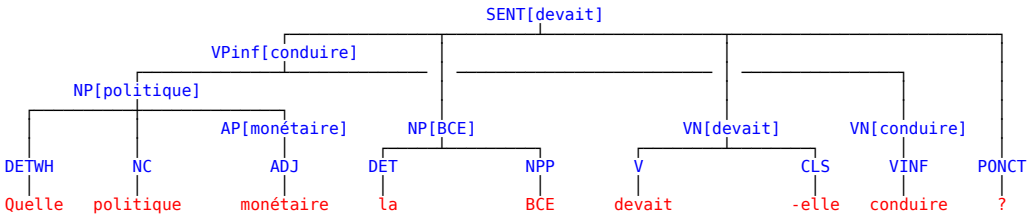


Figure 1.3: Lexicalized Constituency Tree.

constituents are very hard to predict. For example, Maier (2015) report F1 scores lower than 20 on discontinuous constituents.

1.2.4 Learning Bilexical Relations

Most transition-based constituency parsers are **lexicalized** in the sense that they predict lexicalized constituency trees. In a lexicalized tree, each constituent is associated with its lexical head, as illustrated in Figure 1.3. Lexicalized transition systems implicitly construct an unlabelled dependency tree during parsing and they rely on the head of constituents for feature extraction (see Section 3.2.1.4). In other words, they rely on the assumption that constituents are built around a lexical head and that capturing bilexical relations is an essential part of accurate parsing. Such a hypothesis seems all the more important for non-configurational languages such as MRLs, that exhibit more word order variation. Yet, bilexical relations are also more difficult to learn for these languages, due to data sparsity. As it is difficult to generalize from very sparse data, statistical parsers tend to overfit easily.

In this dissertation, we address the question of lexicalization: should transition-based constituency parsers try to model bilexical dependencies, or rely instead on structural information?

1.3 Contributions and Outline

Our contributions address several aspects of parsing: (i) learning biases, (ii) interaction with morphology (iii) transition systems for discontinuous constituency parsing.

1.3.1 A Dynamic Oracle to Deal with Locality Biases

In terms of learning methods, we introduce the first dynamic oracle for projective lexicalized constituency parsing. Transition-based parsers with greedy decoding cannot recover from errors. Wrong decisions at the beginning of parsing may compromise future decisions, due to error propagation. They are usually trained to predict the gold action from a gold configuration (obtained assuming every previous action was gold). They are in a much more different situation at test time, when mistakes can be made.

A way to alleviate this problem is to simulate a more realistic setting at training time, i.e. take into account that the parser may do wrong predictions, and training it to predict the best possible action in any configuration, including configurations where the gold tree is not reachable due to previous mistakes. A dynamic oracle is a function that outputs the set of the best actions (with respect to a loss function) for any possible parsing configuration. Our experiments show that training with the dynamic oracle improved a greedy parser in a multilingual setting.

1.3.2 Multitask Learning for Parsing and Morphological Analysis

In the case of languages with a rich inflectional morphology, tagsets are often very large, and structured into a coarse part-of-speech tag and a set of morphological attributes (such as tense, mood, case). The most common approach to integrate morphological information to parsing is the pipeline approach: use the output of a morphological tagger as features. A limitation of this strategy is that it relies on an external tool that may be hard to deploy, given the complexity of the tagsets. Moreover, pipeline systems are subject to error propagation.

Instead, we developed a parsing architecture that took advantage of the fact that recent proposals in parsing (Kiperwasser and Goldberg, 2016; Cross and Huang, 2016b) use a bi-LSTM component that is very similar to a tagging architecture (Plank et al., 2016). We introduce a joint model based on multitask learning, allowing syntax and morphology to interact seamlessly. We further show that including functional labelling as an additional word-level tagging task improves parsing. The resulting parser outputs both constituency trees and labelled dependency trees and obtains state-of-the-art results on the SPMRL dataset (Seddah et al., 2013).

1.3.3 Transition Systems for Discontinuous Parsing

Although corpora often contain annotations for non-local dependencies resulting from a range of linguistic phenomena (e.g. long distance extraction, extraposition, scrambling), this information is often ignored by constituency parsers. One way of representing the non-local dependencies is to allow crossing branches in constituency trees, as is shown in Figure 1.1, resulting in structures that can be seen as derivations of mildly context sensitive formalisms such as Linear Context-Free Rewriting Systems (LCFRS). This type of structure corresponds to the general case for the description of syntactic structures, as it is adequate for the description of languages with high degree of word order flexibility that are nearly always languages with a rich inflectional morphology (Futrell et al., 2015).

In this dissertation, we introduce a new transition system for lexicalized discontinuous constituency parsing called SR-GAP (shift-reduce-gap). This transition system can derive any labelled discontinuous tree in quadratic time. However, due to the scarcity of discontinuities in language data, it is empirically as fast as a transition-based projective constituency parser, and thus avoids the computational cost induced by the added expressivity of the underlying formalisms. This system obtains state-of-the-art results on several German treebanks.

1.3.4 Unlexicalized Systems for Constituency Parsing

Finally, we introduce an unlexicalized variant of SR-GAP, based on a structure-label strategy (Cross and Huang, 2016a) and its lexicalized counterpart. These transition systems aim at assessing the role of explicit lexicalization in parsing. Parsing results in both projective and discontinuous multilingual constituency parsing show that lexicalization is not necessary to achieve very high results.

1.3.5 Outline

The dissertation is organized in three parts.

Part I provides the necessary background in constituency parsing and consists of three chapters. Chapter 2 introduces chart parsing with probabilistic grammars. Chapter 3 reviews standard transition systems for projective and discontinuous constituency parsing and describes their properties. Chapter 4 focuses on learning and decoding algorithms used in transition-based parsing.

Part II deals with projective constituency parsing and presents two contributions. In Chapter 5, we introduce a dynamic oracle algorithm that improves the training of greedy transition-based constituency parsers. Chapter 6 focuses on the integration of morphological information in the parser and presents a parsing model that performs jointly constituency and dependency parsing, functional labelling and morphological analysis.

Part III focuses on discontinuous constituency parsing. Chapter 7 introduces a new transition system for discontinuous transition-based parsing. It is an extension of the shift-reduce algorithm and can derive any labelled discontinuous constituency tree in linear empirical time ($\mathcal{O}(n^2)$ in the worst case).

Chapter 8 investigates the question of lexicalization; it introduces transition systems for unlexicalized and lexicalized discontinuous parsing and compares them in different settings. It shows that explicit lexicalization is not necessary to achieve very strong results in discontinuous parsing. This result is confirmed with additional experiments on multilingual projective constituency parsing.

1.3.6 Publications

Some of the contributions presented in this dissertation have been published before in conferences or journals.

- Chapter 4 discusses some results published in TALN (Coavoux and Crabbé, 2015) and TAL (Coavoux and Crabbé, 2016).
- Chapter 5 has been published in ACL (Coavoux and Crabbé, 2016).[2]
- Chapter 6 is an extended version of an article published in EACL (Coavoux and Crabbé, 2017b).[3]
- Chapter 7 is an extended version of an article published in EACL (Coavoux and Crabbé, 2017a).[4]
- Chapter 8 presents mostly original unpublished work. The baseline model used in this chapter has been published in TALN (Coavoux and Crabbé, 2017).

Finally, the code used for all experiments carried out for this dissertation has been published online, as free software, in the following repositories:

- `www.github.com/mcoavoux/mtg`
- `www.github.com/mcoavoux/hyparse`
- `www.github.com/mcoavoux/french_disco_data`
- `www.github.com/mcoavoux/multilingual_disco_data`

[2]The chapter includes a few verbatim passages from the article.
[3]See footnote 2.
[4]See footnote 2.

Part I

Background

Chapter 2

Probabilistic Chart Parsing

Contents

2.1 Introduction: Parsing Paradigms **25**
2.2 Syntactic Representations and Formalisms **26**
2.2.1 Constituency and Dependency Trees 26
2.2.2 Representing Non-Local Dependencies 26
2.3 Chart Parsing **28**
2.3.1 Probabilistic Models for CFG Parsing 29
2.3.2 Probabilistic LCFRS 36
2.4 Conclusion **39**

2.1 Introduction: Parsing Paradigms

There are two main parsing paradigms: **chart parsing** and **transition-based parsing**.[1] A chart parser generally uses an explicit grammar and models directly trees. It searches for the best derivation for a sentence with dynamic programming to factorize common subtrees in different hypotheses. Typical chart parsers for projective structures run in $\mathcal{O}(|G| \cdot n^3)$ where n is the length of a sentence and G is the size of the grammar.

In contrast, transition-based parsers do not model directly trees, but sequences of actions that build trees. They are usually incremental and

[1] Excluding **graph-based parsing** (McDonald et al., 2005), a third paradigm only used in dependency parsing.

support fast approximate search. As they do not model a grammar explicitly, they are inherently robust.

There is some overlap between the two paradigms. Decoding in chart parsing may be performed incrementally (by computing the scores of subtrees on prefixes of the sentence). It is also possible to use dynamic programming in transition-based parsing under certain conditions (Huang and Sagae, 2010).

In this chapter, we review standard formalisms typically used to represent syntactic structures (Section 2.2), and chart parsing methods based on probabilistic grammars (Section 2.3).

2.2 Syntactic Representations and Formalisms

2.2.1 Constituency and Dependency Trees

There are two widely used types of syntactic representations in Natural Language Processing (NLP): dependency trees and constituency trees (Figure 2.1). Dependency trees represent the syntactic structure as a set of directed typed relations between governors and their dependents. For example, in Figure 2.1, the word *faut* is the root of the tree and has two arguments, a subject *il* and an object *étonner*, and several modifiers (negation particles).

In contrast, constituency trees focus on phrases, i.e. recursive groupings of words based on syntactic and distributional properties. For examples, a Noun Phrase (NP) such as *un flop* (Figure 2.1) may usually be substituted to another NP without changing the grammaticality of the sentence.

Although dependency and constituency trees focus on different aspects of the syntactic structure, there is a very large overlap in the information they represent. In particular, there are conversion procedures between the two formalisms (Candito et al., 2010). The dependency versions of the French Treebank (Abeillé et al., 2003) and the Penn Treebank (Marcus et al., 1993) were automatically converted from the original constituency versions. Moreover, some grammatical formalisms encode both constituency and dependency information, e.g. lexicalized Tree Adjoining Grammars and lexicalized Context-Free Grammars (see Section 2.3.1.2).

2.2.2 Representing Non-Local Dependencies

Long distance dependencies are syntactic relations that link words that are arbitrarily far in the sentence. A number of phenomena may produce

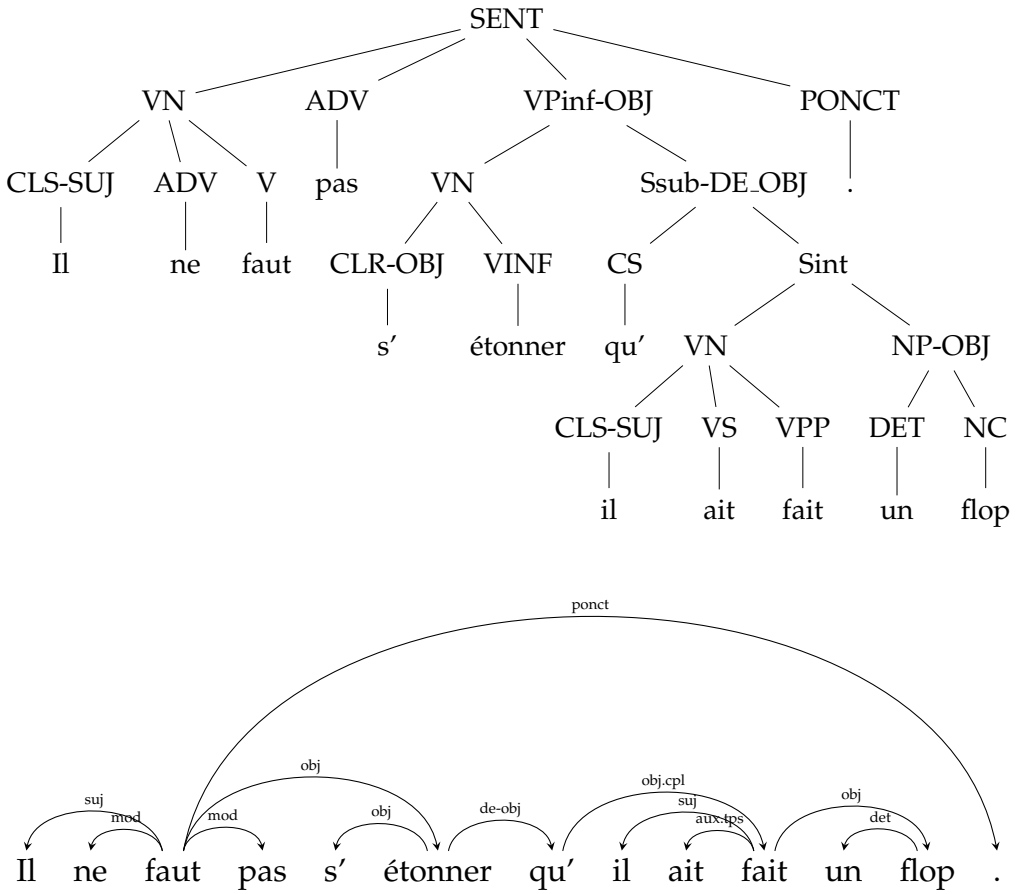


Figure 2.1: Constituency tree (upper part) and dependency tree (lower part). *It is not surprising that he flopped.*

non-local dependencies, such as long-distance extractions or dislocations, as illustrated by the following examples where the items involved in the non-local dependency are in bold:

- **Who** do you think will **come** ? (question extraction)
- **The cat** that the dog tried to **bite** ran away. (relative extraction)
- **An excellent environmental actor** he **is**. (dislocation)

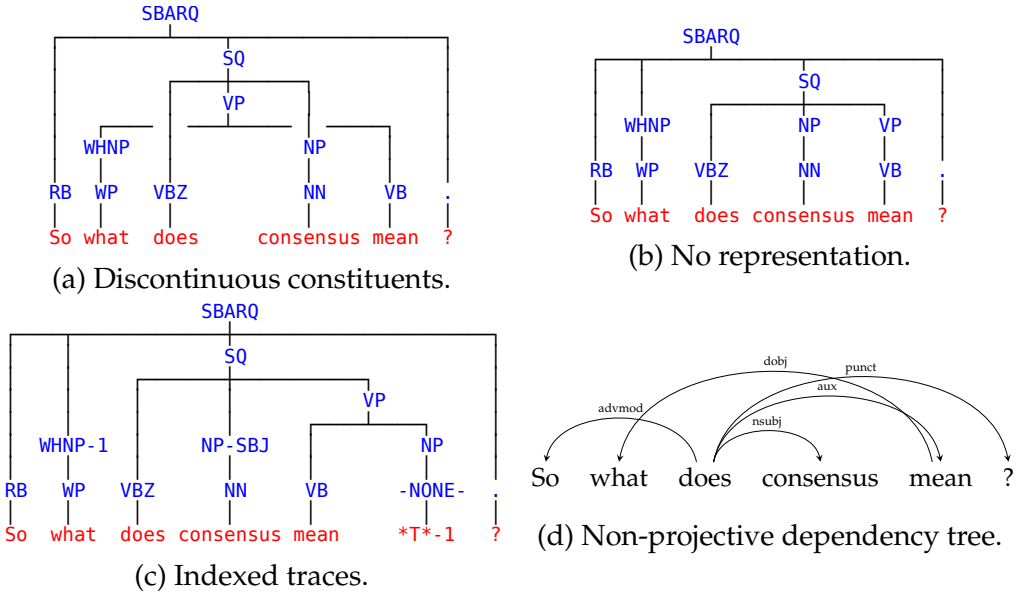
In dependency trees, these phenomena are usually accounted for with **non-projectivity**. In a projective dependency tree, every word dominates a contiguous string of words in the sentence. In contrast, in a non-projective tree, this constraint does not hold. For example, in Figure 2.2d, the word *mean* does dominate its argument *what* but not the auxiliary *does*, nor *consensus*.

Constituency trees can also use non-projectivity to represent non-local dependencies. Yet, few constituency corpora are natively annotated with discontinuous constituents. Among these, the corpora most used in parsing experiments are the Negra corpus (Skut et al., 1997) and the Tiger corpus (Brants et al., 2002). Instead, the long-distance dependencies are either not annotated (French Treebank) or represented with indexed traces (Penn Treebank). In Figure 2.2c, the dependency between *what* and *mean* is represented by an empty category `*T*` in the prototypical position occupied by the object of a verb, coindexed with the extracted element. However, constituency parsers typically train and evaluate on modified versions of treebanks where traces are deleted as well as functional annotations (Figure 2.2b). This preprocessing has become standard and makes parsing easier, as empty categories are hard to predict. Nevertheless, the drawback is that parsers completely ignore an important amount of information contained in the original trees.

In this dissertation, we will focus on two types of representations: projective constituency trees without empty categories and discontinuous constituency trees.

2.3 Chart Parsing

We first review grammar-based parsing for projective constituency trees with Probabilistic Context-Free Grammars (PCFG) and related models (Section 2.3.1). Then, we present parsing with Linear Context-Free Rewriting Systems (LCFRS), a grammar formalism that can derive discontinuous constituency trees (Section 2.3.2).



(a) Discontinuous constituents.

(b) No representation.

(c) Indexed traces.

(d) Non-projective dependency tree.

Figure 2.2: Representation strategies for non-local dependencies.

2.3.1 Probabilistic Models for CFG Parsing

After describing PCFG parsing and discussed its limitations (Section 2.3.1.1), we present two extensions of this model: lexicalized PCFG (Section 2.3.1.2) and PCFG with latent annotations (Section 2.3.1.3).

2.3.1.1 Probabilistic Context-Free Grammar

Model Definition A Probabilistic Context-Free Grammar (PCFG) is a Context-Free Grammar (CFG) associated with a function p that gives a probability for each rewrite rule, such that the probabilities of rewrite rules with the same left-hand side symbol sum to 1. The probability of a grammar rule $p(\mathrm{A} \rightarrow \alpha) = p(\alpha|\mathrm{A})$ is the probability that a nonterminal A rewrites as α.

A PCFG can be easily extracted from a treebank. For example, Figure 2.3 shows the PCFG extracted from the two trees in Figure 1.2. The probabilities of grammar rules are estimated by relative frequency.

A PCFG extends a CFG by defining a probability distribution on every possible tree derivable by the grammar. The probability of a tree is the product of probabilities of grammar rules used to derive it:

$$p(t) = \prod_{\mathrm{X} \rightarrow \alpha \in t} p(\alpha|X) \tag{2.1}$$

$$
\begin{aligned}
G &= (N, T, R, \mathrm{S}, p) \\
N &= \{\mathrm{S}, \mathrm{NP}, \mathrm{VP}, \mathrm{PP}\} \\
T &= \{\text{Garnet, eats, a, salad, with, falafels, friends}\} \\
R &= \left\{ \begin{array}{l|l}
\mathrm{S} \to \mathrm{N\ VP} : 1 & \mathrm{N} \to \text{Garnet} : \frac{1}{3} \\
\mathrm{PP} \to \mathrm{P\ N} : 1 & \mathrm{N} \to \text{salad} : \frac{1}{3} \\
\mathrm{VP} \to \mathrm{V\ NP} : \frac{2}{3} & \mathrm{N} \to \text{falafels} : \frac{1}{6} \\
\mathrm{VP} \to \mathrm{VP\ PP} : \frac{1}{3} & \mathrm{N} \to \text{friends} : \frac{1}{6} \\
\mathrm{NP} \to \mathrm{D\ N} : \frac{2}{3} & \mathrm{D} \to \text{a} : 1 \\
\mathrm{NP} \to \mathrm{NP\ PP} : \frac{1}{3} & \mathrm{P} \to \text{with} : 1 \\
 & \mathrm{V} \to \text{eats} : 1
\end{array} \right\}
\end{aligned}
$$

Figure 2.3: Probabilistic Context-Free Grammar.

where t is seen as a set of **anchored rules**. Anchored rules (or instantiated rules) associate a grammar rule with the relevant spans in the sentence, e.g. $\mathrm{X}_{i...j} \to \mathrm{A}_{i...k}\ \mathrm{B}_{k...j}$. A PCFG is also a language model, since it defines a probability distribution over strings of terminals:

$$p(s) = \sum_{t, \text{yield}(t)=s} p(t) \tag{2.2}$$

where the **yield** of the tree is the string of terminals that forms its leaves.

PCFG parsing uses the probability of trees to solve ambiguities. The best tree for a sentence s is the highest probability tree:

$$\hat{t} = \underset{t, \text{yield}(t)=s}{\operatorname{argmax}}\ p(t) \tag{2.3}$$

Inference Equation 2.3 can be solved efficiently with dynamic programming. Consider a binary PCFG. Thanks to the decomposition of trees as sets of anchored rules (Equation 2.1), the calculation of the probability of a tree decomposes into identical subproblems. For a tree t with subtrees t_l, t_r with respective roots X, A and B, the probability of t rewrites as:

$$p(t) = p(\mathrm{X} \to \mathrm{A\ B}) \cdot p(t_l) \cdot p(t_r) \tag{2.4}$$

Since an instantiated constituent will appear in many possible trees, the solutions to the subproblems are reused many times.

The CKY algorithm exploits this structure to find the highest probability tree in $\mathcal{O}(|G| \cdot n^3)$ (Algorithm 1). The grammar $G = (N, T, R, \mathrm{S}, p)$ must

$$\left.\frac{[\mathrm{A}, i, k, p_1] \quad [\mathrm{B}, k, j, p_2]}{[\mathrm{X}, i, j, p_1 \cdot p_2 \cdot p(\mathrm{X} \to \mathrm{A\ B})]}\right\} \mathrm{X} \to \mathrm{A\ B} \in R \qquad \left.\frac{}{[\mathrm{X}, i, i+1, p(\mathrm{X} \to s_i)]}\right\} \mathrm{X} \to s_i \in R$$

Table 2.1: Deduction rules for CFG parsing.

be in Chomsky Normal Form (CNF): the right-hand side of every grammar rule must consist of either (i) exactly two nonterminals or (ii) exactly one terminal (lexical rules). Any context-free grammar can be transformed into a weakly equivalent CNF grammar.[2]

The CKY algorithm iteratively fills an array P, called a chart, such that $P[i, j, X]$ stores the probability of the highest probability subtree with the label X and whose span in the sentence is (i, j). At the end of the loop starting at line 10, $P[0, n, \mathrm{S}]$ is the probability of the solution to Equation 2.3. The tree itself can be recovered in a second data structure B.

Algorithm 1 CKY algorithm for CFG weighted parsing.

```
1:  function PARSE(G = (N, T, R, S, p), s)
2:      s is a sentence of length n
3:      G is a CNF grammar
4:      P[i, j, X] ← 0 is an array of double
5:      B[i, j, X] is an array of tuples to store best subtrees
6:      for i = 0 to n − 1 do
7:          for X ∈ N do
8:              if X → s_i ∈ R then
9:                  P[i, i + 1, X] ← p(X → s_i)              ▷ Lexical rules
10:     for j = 2 to n do                                    ▷ j span end
11:         for i = 0 to j − 2 do                            ▷ i span beginning
12:             for k = i + 1 to j − 1 do                    ▷ k split point
13:                 for X → A B ∈ R do
14:                     q ← p(X → A B) · P[i, k, A] · P[k, j, B]
15:                     if q > P[i, j, X] then
16:                         P[i, j, X] ← q
17:                         B[i, j, X] ← (A, B, k)
18:     return (P, B)
```

From a broader point of view, the CKY algorithm may be viewed as a Viterbi-style strategy to solve a shortest path problem in a hypergraph

[2]Transformation algorithms are akin to those presented in Chapter 3.

with a semi-ring structure (Goodman, 1999; Huang, 2008). In this structure, other strategies are possible to speed up parsing, in particular the A* algorithm (Huang, 2008). For further speed-ups, it is customary to add only certain items to the chart (with a threshold, or by keeping only the k-best items), with the cost of optimality.

The CKY algorithm can be modified to output the k-best trees instead of the highest probability tree. Having access to several analyses is useful in order to rerank them using more complex scoring systems (Charniak and Johnson, 2005), or to delay the resolution of ambiguities until the next module in a pipeline approach (e.g. a semantic analyser).

The CKY algorithm is a dynamic programming implementation of the deduction system in Table 2.1. In the deduction-based approach to weighted parsing inference (Pereira and Warren, 1983; Nederhof, 2003), parsing items are tuples $[\mathrm{X}, i, j, p]$ of an anchored nonterminal $\mathrm{X}_{i\ldots j}$ and the probability p of a subtree with $\mathrm{X}_{i\ldots j}$ as a root. The goal is to deduce the final item $[\mathrm{S}, 0, n, p]$ that maximizes p, from axioms (the tokens in the sentence) and a set of inference rules to deduce new items from an agenda of current items. A weighted deduction system describes a high-level specification of a parsing algorithm, and abstracts away from the order in which operations are performed.

The Limitations of PCFG Raw PCFG parsers perform rather poorly, due to their strong independence assumptions. The probability of a grammar rule does not depend on its horizontal context (sibling nodes) nor its vertical context (parent nodes). Moreover, most attachment decisions completely ignore the lexicon despite its informativeness.

The only differences between the two hypotheses in Figure 1.2, ignoring terminals, are the two anchored rules $\mathrm{NP}_{2\ldots6} \rightarrow \mathrm{NP}_{2\ldots4}\ \mathrm{PP}_{4\ldots6}$ and $\mathrm{VP}_{1\ldots6} \rightarrow \mathrm{VP}_{1\ldots4}\ \mathrm{PP}_{4\ldots6}$. As a consequence, if $p(\mathrm{VP} \rightarrow \mathrm{VP\ PP}) > p(\mathrm{NP} \rightarrow \mathrm{NP\ PP})$, then the first structure will never be chosen. The parser will consistently prefer to attach PPs to VPs in ambiguous cases, which is not a desirable behaviour.

In order to address these problems, two research directions have been proposed. The first one, developed by Collins (1999), chose to model bilexical relations between words in order to use lexical information to solve ambiguities. The second one consists in refining the grammar in order to capture finer distributions (Klein and Manning, 2003).

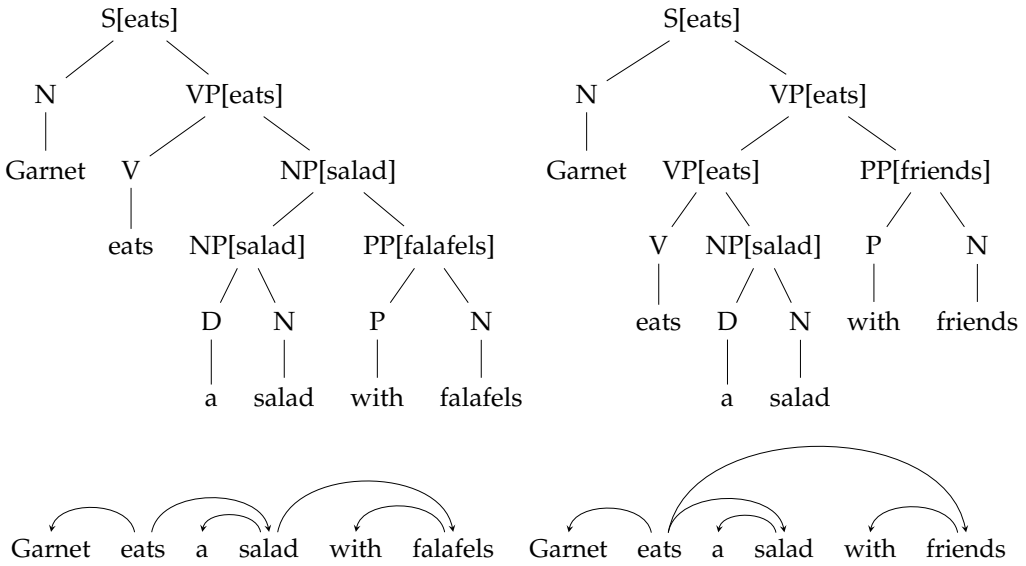


Figure 2.4: Lexicalized constituency trees with the corresponding dependency trees that they encode implicitly.

2.3.1.2 Lexicalized PCFG

In a lexicalized PCFG, each nonterminal is annotated with a terminal that is its lexical head. The rules of a lexicalized PCFG in Chomsky Normal Form have one of the two following forms:

- X[h] $\rightarrow$ A[h_a] B[h_b] where h $\in \{h_a, h_b\}$
- X[h] $\rightarrow$ h

where X[h] means that *h* is the lexical head of constituent X. Each binary rule encodes a bilexical dependency between a head and one of its dependents. As a consequence, a lexicalized constituency tree implicitly encodes an unlabelled dependency tree as shown in Figure 2.4.

The probability of grammar rules used to decide where to attach the prepositional phrase now includes lexical information. The probabilities

$$p(\text{NP[salad]} \rightarrow \text{NP[salad] PP[falafels]})$$
$$\text{and } p(\text{VP[eats]} \rightarrow \text{VP[eats] PP[falafels]})$$

encode the likelihood that *falafels* is a modifier for, respectively, *salad* or *eats*, leading to better attachment decisions.

The limitation of lexicalized PCFGs is that the probabilities of lexicalized rules are very hard to estimate, due to the sparsity of bilexical data

and the explosion of the number of nonterminal symbols ($|N| \times |T|$).[3] In order to make the model work in practice, Collins (1999) proposes several possible decompositions of the probability of lexicalized rules and needs to resort to smoothing methods and backoff strategies to part-of-speech tag.

2.3.1.3 PCFG with Latent Annotations

The second strategy renounces modelling bilexical dependencies, a problem deemed too hard. Instead, it aims at refining the grammar to capture finer distributions of nonterminals.

Raw categories in treebanks do not encode certain distributional properties of symbols. For example, the VP symbol in the Penn Treebank does not distinguish VPs with a finite verb and VPs with an infinitival verb. Moreover, subject NPs are more likely to rewrite as personal pronouns than object NPs. These finer grained distinctions may be captured by symbol-splitting (Klein and Manning, 2003): each symbol in the treebank is annotated with contextual or linguistic information. For example, an NP can be enriched with its parent node leading to a number of new annotated nonterminals (NP^S, NP^VP, NP^NP). Probabilistic grammars extracted from treebanks with such annotations are larger but also more effective for parsing than standard PCFG.

Instead of designing the new split categories manually, they can be learned automatically (Matsuzaki et al., 2005; Petrov et al., 2006) to obtain models called PCFG with Latent Annotations (PCFG-LA).

The split symbols make inference somewhat more complex, as the probability of a tree with the original set of nonterminals is a probability marginalized over each possible combinations of latent annotations:

$$p(t) = \sum_{x \in X} p(t/x) \tag{2.5}$$

where X is the set of all possible affectations of annotations to the nonterminals of t and $p(t/x)$ is the probability of the tree with latent annotations x. Exact parsing is NP-hard for PCFG-LA (Matsuzaki et al., 2005). However, there are several approximate inference methods, the simplest of which is to search for the most probable annotated tree:

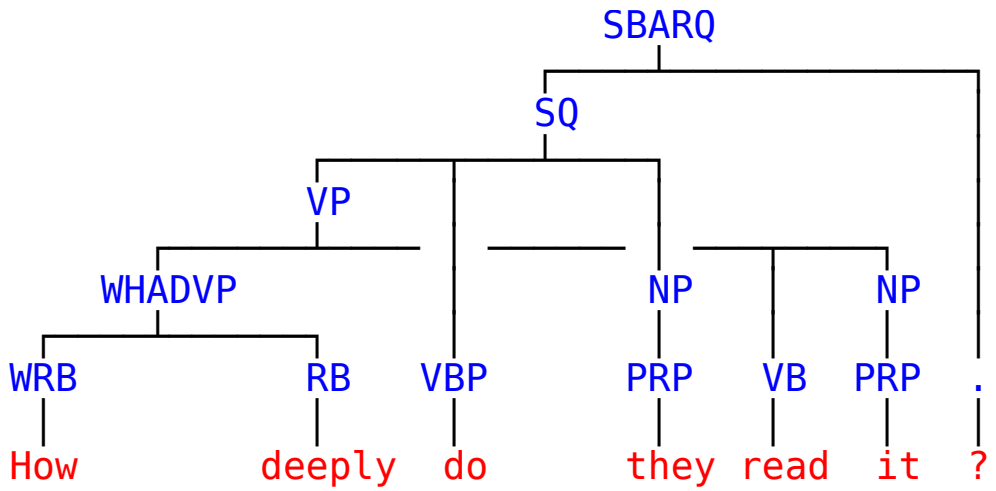
$$\hat{t} = \underset{yield(t/x)=s, x \in X}{\operatorname{argmax}} p(t/x) \tag{2.6}$$

[3] In practice, lexicalized nonterminals are generated dynamically when parsing.

and return the tree stripped of annotations.

PCFG-LA models and derived models, such as Tree Substitution Grammars with latent annotations (Shindo et al., 2012),[4] obtained state-of-the-art results on English. However, they do not perform well on non-configurational languages or languages with a rich inflectional morphology (Seddah et al., 2013). A possible explanation is that these parsers do not integrate morphological information easily.

[4]Tree Substitution Grammars are an extension of CFG in which a nonterminal can rewrite as a tree fragment. It has the same expressivity as a CFG.



$G = (N, T, V, P, \text{SBARQ})$

$N = \{\text{SBARQ, SQ, VP, WHADVP, NP, WRB, RB, VBP, PRP, VB, .}\}$

$T = \{\text{How, deeply, do, they, read, it, ?}\}$

$V = \{X, Y, Z, T\}$

$$P = \left\{ \begin{array}{l|l} \text{SBARQ}(XY) \rightarrow \text{SQ}(X)\ .(Y) & \text{WRB(How)} \rightarrow \epsilon \\ \text{SQ}(XYZT) \rightarrow \text{VP}(X,T)\ \text{VBP}(Y)\ \text{NP}(Z) & \text{RB(deeply)} \rightarrow \epsilon \\ \text{VP}(X, YZ) \rightarrow \text{WHADVP}(X)\ \text{VB}(Y)\ \text{NP}(Z) & \text{VBP(do)} \rightarrow \epsilon \\ \text{WHADVP}(XY) \rightarrow \text{WRB}(X)\ \text{RB}(Y) & \text{PRP(they)} \rightarrow \epsilon \\ \text{NP}(X) \rightarrow \text{PRP}(X) & \text{VB(read)} \rightarrow \epsilon \\ .(?) \rightarrow \epsilon & \text{PRP(it)} \rightarrow \epsilon \end{array} \right\}$$

Figure 2.5: Discontinuous tree and extracted LCFRS.

Discriminative Chart Parsing Finally, a development of unlexicalized chart parsing consists in using a discriminative scoring model instead of an explicit probabilistic grammar (Hall et al., 2014; Durrett and Klein, 2015; Stern et al., 2017). These models weigh subtrees using structural features, such as tokens at the boundaries of constituents.

2.3.2 Probabilistic LCFRS

Discontinuous constituency trees (such as those in Figure 2.2a and 2.5) can be seen as derivations of Linear Context-Free Rewriting Systems (LCFRS). LCFRS is a class of formal grammars that can generate mildly context-sensitive languages. LCFRS provides the expressivity required to describe natural languages as there are mildly-context sensitive phenomena that cannot be described adequately with CFG (Shieber, 1985; Joshi, 1985).

2.3.2.1 Linear Context-Free Rewriting Systems

In an LCFRS grammar, a nonterminal can span a tuple of subsequences of terminals in the sentence whereas in a CFG, a nonterminal spans a single subsequence. For example, in Figure 2.5, the VP spans two subsequences *How deeply* and *read it* that are not contiguous because they are separated by another subsequence *do they*. The **fan-out** of a nonterminal is the number of non-contiguous maximal subsequence it spans. The set of terminals dominated by a nonterminal is called its **yield**. An LCFRS[5] is a tuple $\langle N, T, V, P, \mathrm{S} \rangle$ where:

- N is a finite set of nonterminals with a function $dim : N \to \mathbb{N}$ that determines the fan-out of each nonterminal;
- T and V are disjoint finite sets of terminals and variables;
- $\mathrm{S} \in N$ is the axiom of the grammar with $dim(S) = 1$;
- P is a finite set of rules with the following form

  $$\mathrm{A}(\alpha_1, \ldots, \alpha_{dim(A)}) \to \mathrm{A}_1(X_1^{(1)}, \ldots, X_{dim(A_1)}^{(1)}) \ldots \mathrm{A}_m(X_1^{(m)}, \ldots, X_{dim(A_m)}^{(m)})$$

  such that:

  - A and A_i are nonterminals;

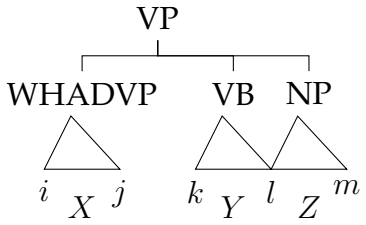
[5] We use the definition and notations from Kallmeyer and Maier (2013) that is based on the vocabulary of Simple Range Concatenation Grammar (Boullier, 1998), an equivalent formalism, rather than the original formulation of Vijay-Shanker et al. (1987).

- $X_j^{(i)}$ are variables:
- $\alpha_i \in (T \cup V)^*$ are sequences of variables and terminals;
- each variable X in the rule occurs exactly once in the left-hand side and exactly once in the left-hand side.

Grammar rules specify both hierarchical relations between nonterminals and how to compute the yield of the nonterminal in the left-hand side with that of the nonterminal in the right-hand side. Consider for instance the rule that generates the discontinuous VP in Figure 2.5:

$$\text{VP}(X, YZ) \rightarrow \text{WHADVP}(X)\ \text{VB}(Y)\ \text{NP}(Z)$$

It specifies that a VP rewrites as a WHADVP VB and NP and spans two subsequences of the sentence represented by X and YZ. This rule corresponds to the following tree fragment:

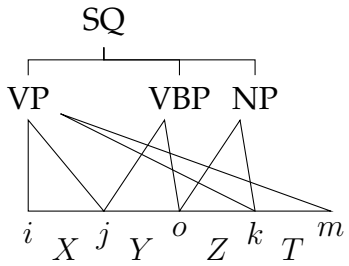


where $i < j < k < l < m$ are indices in the sentence.

Consider now the rule:

$$\text{SQ}(XYZT) \rightarrow \text{VP}(X, T)\ \text{VBP}(Y)\ \text{NP}(Z)$$

The SQ is not discontinuous (its fan-out is one), it rewrites as a discontinuous VP with fan-out two, with two nonterminals VBP and NP inside the gap of the VP. The rule can be illustrated by the following tree fragment:



Finally lexical rules have the form: A(w) $\rightarrow \epsilon$, meaning that A spans a single terminal w and that it has no nonterminal child.

$$\left.\frac{[\mathrm{A}, \boldsymbol{\rho}_A, p_1] \quad [\mathrm{B}, \boldsymbol{\rho}_B, p_2]}{[\mathrm{X}, \boldsymbol{\rho}_X, p_1 \cdot p_2 \cdot p(\mathrm{X}(\boldsymbol{\rho}_X) \rightarrow \mathrm{A}(\boldsymbol{\rho}_A)\ \mathrm{B}(\boldsymbol{\rho}_B))]}\right\} \mathrm{X}(\boldsymbol{\rho}_X) \rightarrow \mathrm{A}(\boldsymbol{\rho}_A)\ \mathrm{B}(\boldsymbol{\rho}_B) \in R$$

$$\left.\frac{}{[\mathrm{X}, \langle\langle i, i+1\rangle\rangle, p(\mathrm{X}(s_i) \rightarrow \epsilon)]}\right\} \mathrm{X}(s_i) \rightarrow \epsilon \in R$$

Table 2.2: Deduction rules for LCFRS parsing.

2.3.2.2 Properties of an LCFRS

There are two important properties to characterize an LCFRS grammar: its **rank** and its **fan-out**. The fan-out of an LCFRS grammar is that of the nonterminal with the highest fan-out. An LCFRS with fan-out one is equivalent to a CFG. The **rank** of an LCFRS rule is the number of nonterminals in its right-hand side and the rank of an LCFRS grammar is the highest rank of its rules. Like CFG, LCFRS grammars can be binarized with reversible algorithms. However, it is possible that a binarized LCFRS has a fan-out greater than the fan-out of the original grammar. As the complexity of LCFRS parsing depends on the fan-out, binarization methods have been proposed to minimize the fan-out of the resulting binary grammar (Gómez-Rodríguez et al., 2009).

2.3.2.3 Parsing with Probabilistic LCFRS

A probabilistic LCFRS is an LCFRS associated with a function $p : R \rightarrow [0, 1]$ that gives a probability for each grammar rule, such that the probabilities of rules with the same left-hand side sum to one. A binary probabilistic LCFRS can be parsed in $\mathcal{O}(n^{|3f|})$ (Kallmeyer, 2010) with CKY-style dynamic programming, where f is the fan-out of the grammar. Table 2.2 presents deduction rules for CKY LCFRS parsing. Parsing items have the form $[\mathrm{A}, \boldsymbol{\rho}_A, p]$ where $(\mathrm{A}, \boldsymbol{\rho}_A)$ is an instantiated nonterminal and p is the probability of the parsing item. An instantiated nonterminal is associated with a vector of ranges $\boldsymbol{\rho}_A = \langle\langle l_1, r_1\rangle, \ldots \langle l_m, r_m\rangle\rangle$ where $dim(A) = m$ and each $\langle l_i, r_i\rangle$ is the span of a subsequence of the sentence dominated by A. Likewise, an instantiated grammar rule $\mathrm{X}(\boldsymbol{\rho}_X) \rightarrow \mathrm{A}(\boldsymbol{\rho}_A)\ \mathrm{B}(\boldsymbol{\rho}_B)$ is a rule where variables and terminals are replaced by range vectors. The goal of the deduction system is to derive the item $[\mathrm{S}, \langle\langle 0, n\rangle\rangle, p]$ with the highest probability.

2.3.2.4 A Related Formalism: Discontinuous Data-Oriented Parsing

Data-Oriented Parsing (DOP) models are probabilistic grammars that considers arbitrary tree fragments to define the probability of tree, instead of only local grammar rules. Initially developed for projective constituency parsing, DOP was adapted to discontinuous constituency parsing by van Cranenburgh et al. (2011) and subsequent work (van Cranenburgh, 2012; van Cranenburgh and Bod, 2013; van Cranenburgh et al., 2016). DOP models perform better than Probabilistic LCFRS parsing, but require heuristics to make inference time reasonable (van Cranenburgh et al., 2016).

2.4 Conclusion

In this chapter, we have presented chart parsing methods for two types of syntactic structures: projective constituency trees and discontinuous constituency trees. Discontinuous trees can model directly syntactic phenomena that are difficult to represent with projective trees without an additional layer of annotation, such as indexed traces. They can be viewed as derivations from mildly context-sensitive grammars, such as Linear Context-Free Rewriting Systems. With exact decoding, chart parsers based on probabilistic grammars can decide grammaticality for an input sentence. However, they have a high polynomial complexity and are impractical for large-scale robust parsing. In the next two chapters, we focus on another parsing paradigm: transition-based parsing.

Chapter 3

Transition-based Parsing

Contents

3.1 Introduction **40**
3.1.1 Parsing as a Search Problem 41
3.1.2 Deterministic Parsing 42
3.2 Transition Systems **43**
3.2.1 A Lexicalized Shift-Reduce Transition System 44
3.2.2 Discontinuous Constituency Parsing Transition Systems 50
3.2.3 Preprocessing the Trees 54
3.3 Conclusion **62**

3.1 Introduction

In the transition-based parsing framework, a syntactic tree is decomposed into a sequence of elementary actions called **transitions**. The problem of syntactic parsing is thus reduced to a sequence prediction problem. For a sequence of tokens $w_1^n = (w_1, \ldots, w_n)$, the goal is to predict a sequence of transitions $a_1^m = (a_1, \ldots, a_m)$, called a **derivation** that builds the best syntactic tree for the sentence. Transitions are applied to the **state** of the parser, represented by a tuple of data structures. A parsing example in this framework is illustrated in Table 3.1 with a toy sentence, where the couple (Stack, Buffer) defines the state of a shift-reduce parser.

	Parsing state	
Action	Stack	Buffer
		Cats like catnip
Shift $\Longrightarrow$	Cats	like catnip
Reduce-Unary(NP) $\Longrightarrow$	(NP Cats)	like catnip
Shift $\Longrightarrow$	(NP Cats) like	catnip
Shift $\Longrightarrow$	(NP Cats) like catnip	
Reduce-Unary(NP) $\Longrightarrow$	(NP Cats) like (NP catnip)	
Reduce-Binary(VP) $\Longrightarrow$	(NP Cats) (VP like (NP catnip))	
Reduce-Binary(S) $\Longrightarrow$	(S (NP Cats) (VP like (NP catnip)))	

Table 3.1: Derivation of a syntactic tree with the shift-reduce algorithm.

3.1.1 Parsing as a Search Problem

Transition-based parsing can be viewed as a search problem, where an agent aims at achieving a goal by accomplishing a sequence of actions (Russell and Norvig, 2003). A typical search problem is defined by the following components:

- An **initial state**.
- A set of possible **actions**. Actions are partial functions from states to states, that can only be performed if some preconditions on input states are satisfied.
- A **goal test** that determines whether the current state is a final state, i.e. whether the agent's goal is achieved.
- A **path cost** function that weighs every possible path, generally with the sum of costs of actions.

In the context of parsing, states are tuples of data structures that typically contain subtrees. Actions construct new trees by creating new labelled nodes and arcs. The preconditions on actions ensure the well-formedness of the syntactic tree being constructed. The goal test simply checks whether the parser has constructed a syntactic tree that spans every tokens in the input sentence. Finally, the cost function is based on a statistical classifier that weighs each possible action in a given state. In parsing, the size of the search space is exponential in the length of the sentence, which makes exhaustive search impractical.

Transition-based parsing is an **online search** problem. Online search problems are a subtype of search problems, where the agent only observes its current state, and thus must perform actions to explore the state space

and find a solution (Russell and Norvig, 2003). Usually, the parser can only partially observe its current state. It only uses a local region of the state to construct an input for the classifier. As a result, it is subject to biases when weighing actions.[1]

3.1.2 Deterministic Parsing

Given a set of actions A, the size of the search space $\mathcal{O}(|A|^k)$ is exponential in the number k of actions in a derivation. To reduce the size of the search space, the set of actions is usually designed to forbid cycles. Preconditions on actions make sure that there is a bound on the longest possible derivation, that depends on the length of the sentence. Typically, a transition system for projective parsing derives a tree in $\mathcal{O}(n)$ steps, whereas a system for discontinuous parsing can perform at most $\mathcal{O}(n^2)$ steps, where n is the length of the sentence.

A complete transition-based parser usually comprises the following components:

- A **transition system** defines parsing configurations and a set of transitions with their preconditions.
- An **oracle** is an algorithm used to determine the gold derivation from a tree with a specific transition system (see Chapter 5 for more precise definitions).
- A **statistical model** is responsible for giving scores to different derivations for the same sentence and disambiguating between them. Usually, the score of a derivation decomposes as the sum of the scores of actions, which facilitates search for the best tree.
- A **search algorithm** is used to determine the highest-scoring derivation.

A transition system consists of a set of states and a set of transitions, which are partial functions from states to states. In the context of parsing, states are usually called **configurations** and defined as tuples containing partial structures being constructed and unprocessed tokens. A specific transition system must define:

- Configurations, including special states such as the initial configuration and the final configurations;

[1] Models based on recurrent neural networks (see Section 4.5.3) address these biases by constructing features based on the whole parsing state.

- A finite set A of transitions;
- A boolean function specifying whether transition $a \in A$ is allowed in a configuration c.

Parsing with a transition-system starts with the initial configuration and consists in deriving new configurations until a final configuration is reached. Transition systems are non-deterministic, the state space size is exponential in the size of the sentence to parse. Determinism can be provided by a set of heuristics designed from linguistic knowledge (Nivre, 2003). Most of the time though, parsers rely on a multi-class classifier which scores each possible action in a given configuration and resort to approximate search algorithms to select the best action sequence. In Algorithm 2, we present a generic transition-based parsing algorithm, based on greedy search.

Algorithm 2 High-level transition-based parsing algorithm.

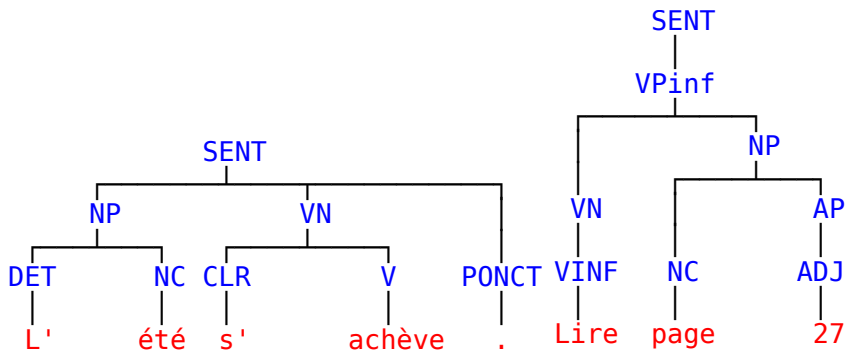
function PARSE(w_1^n, f)
  ▷ f: weight function (statistical model)
  $c \leftarrow$ INITIAL(w_1^n)
  while c is not a final configuration **do**
    $T \leftarrow \{a | \text{ISPOSSIBLE}(a, c)\}$
    $t \leftarrow \text{argmax}_{a \in T} f(c, a)$
    $c \leftarrow t(c)$
  return GETTREE(C) ▷ Recover tree from final configuration

3.2 Transition Systems

Depending on the type of syntactic representations used and their properties (e.g. projectivity), researchers have proposed several transition systems.[2] Most transition systems for constituency parsing are based on the shift-reduce algorithm inherited from the theory of compilers (Knuth, 1965). Early works on natural language parsing extended the LR algorithm to handle the inherent ambiguity of natural language grammars (Generalized LR algorithm by Tomita, 1987). Data-driven constituency parsers are based on Sagae and Lavie (2005), who drew inspiration from classifier-based dependency parsers that were proposed a couple of years earlier (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004).

[2]See Nivre (2008) for an overview of dependency parsing transition systems.

(a)



(b)

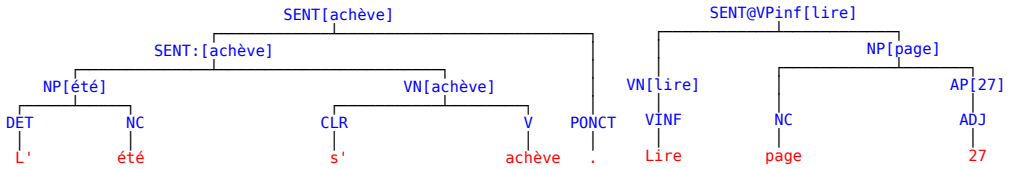


Figure 3.1: Trees from the French Treebank (Abeillé et al., 2003) (a), and the result of the preprocessing step (b). *Summer ends. Read page 27.*

In this section, we present standard transition systems for projective constituency parsing, focusing on the system that we will use in Chapters 5 and 6 (Section 3.2.1). Then, we present discontinuous transition systems (Section 3.2.2) to which we will compare the SR-GAP system introduced in Chapter 7. Most transition systems can only output a subset of labelled syntactic trees, usually projective binary trees with a limited number of unary constituents. In order to predict n-ary trees, it is standard to preprocess trees in the training set with reversible modifications so that they satisfy those constraints, and to unprocess predicted trees to obtain n-ary syntactic trees (Section 3.2.3).

3.2.1 A Lexicalized Shift-Reduce Transition System

In this section, we present a standard lexicalized transition system introduced by Crabbé (2014) that we will use in later chapters (5 and 6). This is a modified version of the transition system of Sagae and Lavie (2005), with a mechanism designed to make sure that a derivation for a sentence of length k is bounded by a function of k. This property is important because

Input	$t_1[w_1]t_2[w_2]\dots t_n[w_n]$
Axiom	$\langle \epsilon, t_1[w_1]t_2[w_2]\dots t_n[w_n]\rangle$
Goal	$\langle A[w], \epsilon\rangle$
SHIFT	$\frac{\langle S, t[w]\|B\rangle}{\langle S\|t[w], B\rangle}$
REDUCE-UNARY-X	$\frac{\langle S\|s_0[h], B\rangle}{\langle S\|X[h], B\rangle}$
REDUCE-RIGHT-X	$\frac{\langle S\|s_1[h]\|s_0[h'], B\rangle}{\langle S\|X[h'], B\rangle}$
REDUCE-LEFT-X	$\frac{\langle S\|s_1[h]\|s_0[h'], B\rangle}{\langle S\|X[h], B\rangle}$
GHOST-REDUCE	$\frac{\langle S\|t[h], B\rangle}{\langle S\|t[h], B\rangle}$

Table 3.2: Lexicalized shift-reduce transition system (Crabbé, 2014) as a deduction system.

statistical models may be biased towards longer or shorter derivations, harming comparability between derivations of different lengths. This issue is important in particular when the parser uses a global model (Zhu et al., 2013).

In what follows, we assume that syntactic trees are binary, and that the only unary constituents generate preterminals. We also assume that trees are lexicalized: each constituent is annotated with its lexical head. The preprocessing steps that are needed to transform the trees accordingly are described in Section 3.2.3. We illustrate in Figure 3.1 the result of this transformation for two sentences from the French Treebank.

3.2.1.1 Set of Transitions

A configuration is defined as a couple $\langle S, B \rangle$ where S is a stack containing subtrees and B is a buffer containing tokens. In the initial configuration, S is empty and B contains the sequence of tokens to be parsed. A configuration $\langle S, B \rangle$ is final iff S contains a single tree rooted by the axiom and B is empty. Actions are defined as follows:

- SHIFT pops a token from the buffer and pushes it on the stack.
- REDUCE-UNARY-X pops the subtree at the top of the stack, creates a new unary constituent labelled X with the subtree as its child and pushes X onto the stack.
- REDUCE-RIGHT-X and REDUCE-LEFT-X pop two subtrees from the stack, create a new binary constituents with the two subtrees as its children, choose the lexical head of the right (resp. left) child to be the lexical head of X, and push X onto the stack.

The transitions are summarized as a deduction system in Table 3.2. In this table, $X[h]$ denotes a nonterminal X and its head h. The input consists of tokens $w_1 \dots w_n$ and their POS tags $t_1 \dots t_n$. The symbol | is used for concatenation, s_0 and s_1 denote the two topmost elements in the stack. Finally, A is the root symbol.

3.2.1.2 The Derivation Length Problem

With the set of actions defined above, a derivation for a sentence of length n has n SHIFTS, $n - 1$ binary reductions and between 0 and n unary reductions. As stated above, statistical models used to disambiguate between possible derivations might be biased towards longer or shorter derivations. For example, Crabbé (2014) shows that the score of a partial derivation computed by a perceptron is approximately linear in the number of actions. As a consequence, the parser may try to maximize the number of unary reductions when searching for the best derivation.

To solve this problem, Crabbé (2014) introduced a GHOST-REDUCE action that has no effect, and added the constraint that each SHIFT action must obligatorily be followed by either a REDUCE-UNARY-X or a GHOST-REDUCE action. This constraint ensures that the sum of GHOST-REDUCE and REDUCE-UNARY-X occurrences in a derivation is exactly n. Consequently, there is exactly $3n - 1$ actions in any derivation. We present a full derivation in Table 3.3 for the binarized tree of Figure 3.1. The tree is derived in exactly 14 steps ($3 \times 5 - 1$) since the sentence has 5 tokens.

Action	Stack	Buffer
		L'$_1$ été$_2$ s'$_3$ achève$_4$.$_5$
SHIFT $\Longrightarrow$	L'	été$_2$ s'$_3$ achève$_4$.$_5$
GHOST-REDUCE $\Longrightarrow$	L'	été$_2$ s'$_3$ achève$_4$.$_5$
SHIFT $\Longrightarrow$	L' été	s'$_3$ achève$_4$.$_5$
GHOST-REDUCE $\Longrightarrow$	L' été	s'$_3$ achève$_4$.$_5$
REDUCE-RIGHT-NP $\Longrightarrow$	(NP[2] L' été)	s'$_3$ achève$_4$.$_5$
SHIFT $\Longrightarrow$	(NP[2] L' été) s'	achève$_4$.$_5$
GHOST-REDUCE $\Longrightarrow$	(NP[2] L' été) s'	achève$_4$.$_5$
SHIFT $\Longrightarrow$	(NP[2] L' été) s' achève	.$_5$
GHOST-REDUCE $\Longrightarrow$	(NP[2] L' été) s' achève	.$_5$
REDUCE-RIGHT-VN $\Longrightarrow$	(NP[2] L' été) (VN[4] s' achève)	.$_5$
REDUCE-RIGHT-SENT: $\Longrightarrow$	(SENT:[4] (NP[2] L' été) (VN[4] s' achève))	.$_5$
SHIFT $\Longrightarrow$	(SENT:[4] (NP[2] L' été) (VN[4] s' achève)) .	
GHOST-REDUCE $\Longrightarrow$	(SENT:[4] (NP[2] L' été) (VN[4] s' achève)) .	
REDUCE-LEFT-SENT $\Longrightarrow$	(SENT[4] (SENT:[4] (NP[2] L' été) (VN[4] s' achève)) .)	

Table 3.3: Derivation with the transition system of Crabbé (2014). For legibility, lexical heads are indicated by the terminal index instead of the full word form.

3.2.1.3 Preconditions on Actions

The set of nonterminals for preprocessed trees includes the nonterminals in the original trees as well as the following two categories of new nonterminals:

- **Collapsed symbols** contain '@' and represent tree fragments made only of unary rewrites. For example, the symbol SENT@VPinf in Figure 3.1 represents the two nodes SENT $\rightarrow$ VPinf. Any symbol that starts with X@, where X is the root symbol of every tree in the original treebank is considered to be a possible axiom by the transition system.

- **Temporary symbols** are used to binarize an n-ary constituent and are suffixed by ':'. For example, the constituent SENT: in Figure 3.1 is part of a constituent labelled SENT.

A direct consequence of the binarization algorithm (Section 3.2.3) is that the binarized trees have the two following properties: (i) no node may have two temporary symbols as children (ii) a temporary symbol X: must contain in its span the lexical head of the corresponding X constituent. Figure 3.2 features two trees that cannot have been obtained by the binarization algorithm. In the left-hand tree, a nonterminal has two children labelled with temporary symbols. In the right-hand tree, the temporary symbol X: does not span the lexical head of X.

The preconditions on actions both make sure that the predicted binary trees can be unambiguously unbinarized and that some basic conditions are satisfied (e.g. impossible to shift if the buffer is empty). The two main

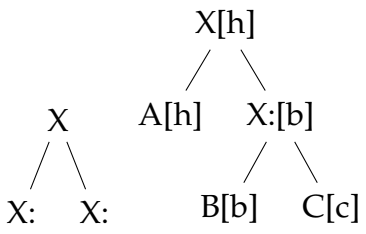


Figure 3.2: Impossible binarized trees.

constraints to handle temporary symbols is that it is forbidden to apply a binary reduction to two temporary symbols, or to derive a configuration in which the buffer is empty and the two topmost symbols in the stack are temporary symbols. The parser state must satisfy the following conditions for a transition to be performed:

- SHIFT
  - B is not empty
  - the previous action is not SHIFT
- REDUCE-LEFT-X (resp. REDUCE-RIGHT-X)
  - s_0 (resp. s_1) is not a temporary symbol (conditions i and ii above)
  - S has at least 2 elements
  - the previous action is not SHIFT
  - if B is empty and s_2 is a temporary symbol, X must not be a temporary symbol (condition i above)
  - if B is empty and S has 2 elements, X must be an axiom
- REDUCE-UNARY-X
  - The previous action is SHIFT
  - If the sentence has only one token, X must be an axiom
- GHOST-REDUCE
  - The previous action is SHIFT
  - The sentence has more than 1 token

3.2.1.4 Relationship with the Arc-Standard System

Since this transition system predicts lexicalized trees, it constructs simultaneously a dependency tree and a constituency tree. The dependency tree construction is explicit with the binary reductions. If s_1 and s_0 have lexical heads h_1 and h_0, REDUCE-LEFT (resp. REDUCE-RIGHT) produces an unlabelled dependency arc $h_1 \rightarrow h_0$ (resp. $h_0 \rightarrow h_1$).

In fact the algorithm constructs a dependency tree in the same fashion as the arc-standard transition system formulated by Nivre (2004)[3] In the arc-standard transition system, a configuration is a triple $\langle \sigma, \beta, A \rangle$ where σ is a stack, β is a buffer and A is a set of labelled dependency arcs with the form (governor, label, dependent). Its transitions are defined as follows:

- $\text{SHIFT}(\langle \sigma, b|\beta, A \rangle) = \langle \sigma|b, \beta, A \rangle$
- $\text{RIGHT}_l(\langle \sigma|s_1|s_0, \beta, A \rangle) = \langle \sigma|s_1, \beta, A \cup \{(s_1, l, s_0)\} \rangle$
- $\text{LEFT}_l(\langle \sigma|s_0, \beta, A \rangle) = \langle \sigma|s_0, \beta, A \cup \{(s_0, l, s_1)\} \rangle$

With this transition system, if we ignore the dependency labels, the derivation for the dependency tree encoded in the lexicalized constituency tree is the following sequence: SHIFT, SHIFT, RIGHT, SHIFT, SHIFT, RIGHT, RIGHT, SHIFT, LEFT, which is identical to the derivation in Figure 3.2, modulo the GHOST-REDUCE actions.

3.2.1.5 Related Transition Systems

Since Sagae and Lavie (2005), several variants have been introduced, which are all based on the same actions (SHIFT, REDUCE-LEFT, REDUCE-RIGHT, REDUCE-UNARY) but have different strategies to address the issues of unary constituents and varying derivation lengths.

Zhang and Clark (2009) do not collapse unary constituents during preprocessing but limit the number of consecutive unary reductions to three to ensure that the length of a derivation is bounded. Zhu et al. (2013) use a padding mechanism, with an action called IDLE that can only be performed if a goal state has been reached. During a beam search, if one item in the beam has already reached a goal state, it is padded with IDLE actions until every hypothesis in the beam has also reached a goal state. Since these actions are scored like any other actions, the scores between derivations in the beam are more comparable.

[3]This formulation differs slightly from the formulation usually presented (Nivre, 2008).

Finally, Mi and Huang (2015) use an approach similar to that of Crabbé (2014) but more general. At parsing step i, if i is even, the parser can only SHIFT or perform a binary reduction; if i is odd, the parser can only perform a unary reduction or an action with no effect. They preprocess trees to allow unary constituents as long as they immediately dominate a binary constituent or a preterminal (but not another unary constituent). This method ensures that any derivation has exactly $4n - 2$ steps, and that different hypotheses are synchronized in the beam. It also requires a lighter preprocessing, as not all unary constituents have to be collapsed to single symbols, which alleviates the grammar inflation that typically results from treebank preprocessing (see Section 3.2.3).

3.2.2 Discontinuous Constituency Parsing Transition Systems

Transition-based methods for parsing discontinuous constituents were proposed recently (Versley, 2014a,b; Maier, 2015; Maier and Lichte, 2016). Previous approaches to data-driven discontinuous constituency parsing rely either on probabilistic grammars (Section 2.3.2) or on a reduction to dependency parsing. We briefly review the latter method (Section 3.2.2.1), before presenting two transition systems for discontinuous constituency parsing: an easy-first system (Section 3.2.2.2) and a swap-based system (Section 3.2.2.3).

3.2.2.1 The Reduction to Dependency Parsing Approach

The problem of discontinuous constituency parsing can be reduced to non-projective dependency parsing by using a reversible transformation from constituency trees to dependency trees. To avoid loss of information in the transformation, it is necessary to encode structural information in the set of labels used in dependency trees. The advantage of this approach is the ability to use state-of-the-art dependency parsers with rich feature templates, and to scale easily to full corpora, whereas methods based on probabilistic grammars usually use inference algorithms that have a high polynomial complexity and are hardly practical for very long sentences. The approach based on a reduction to dependency parsing was first proposed by Hall and Nivre (2008) and notably developed by Fernández-González and Martins (2015).

In the rest of this section, we present two transition systems for discontinuous constituency parsing: EAFI (Easy-first, Section 3.2.2.2) and SR-

SWAP (shift-reduce-swap, Section 3.2.2.3). They both rely on the same strategy: using a swap action (Nivre, 2009) to reorder terminals and reduce the problem to projective parsing.

3.2.2.2 The Easy-First Transition System

Versley (2014b) introduced EAFI, a transition system for discontinuous constituency parsing inspired by the easy-first transition system for dependency parsing (Goldberg and Elhadad, 2010a). The easy-first strategy aims at minimizing error propagation by taking easy attachment decisions first and delaying the difficult decisions as long as possible.

In this framework, in contrast to other transition systems, parsing is non-directional. Trees are constructed in a bottom-up fashion. A parsing configuration $\langle L \rangle$ consists of a single list, called *pending* in the terms of Goldberg and Elhadad (2010a), that initially contains the sequence of tokens to be parsed. At each step, the parser chooses one of the following actions:

- REDUCE-UNARY-X$(\langle l_0 \ldots, l_{i-1}, l_i, l_{i+1} \ldots l_n \rangle)$
  $= \langle l_0 \ldots, l_{i-1}, X, l_{i+1} \ldots l_n \rangle$;
- REDUCE-BINARY-X$(\langle l_0 \ldots, l_{i-1}, l_i, l_{i+1}, l_{i+2} \ldots l_n \rangle)$
  $= \langle l_0 \ldots, l_{i-1}, X, l_{i+2} \ldots l_n \rangle$,
  X has l_i and l_{i+1} as subtrees;
- ADD-LEFT$(\langle l_0 \ldots, l_{i-1}, l_i, l_{i+1} \ldots l_n \rangle) = \langle l_0 \ldots, l_{i-1}, l_{i+1} \ldots l_n \rangle$,
  l_{i-1} has l_i as subtree;
- ADD-RIGHT$(\langle l_0 \ldots, l_{i-1}, l_i, l_{i+1} \ldots l_n \rangle) = \langle l_0 \ldots, l_{i-1}, l_{i+1}, \ldots l_n \rangle$,
  l_{i+1} has l_i as subtree;
- SWAP$(\langle l_0 \ldots, l_{i-1}, l_i, l_{i+1}, l_{i+2}, \ldots l_n \rangle) = \langle l_0 \ldots, l_{i-1}, l_{i+1}, l_i, l_{i+2}, \ldots l_n \rangle$.

The REDUCE-BINARY actions also assign the head of the new constituent based on a head percolation table.[4] The SWAP action has the precondition that the heads h and h' of l_i and l_{i+1} are in surface order. This precondition prevents cycles of useless SWAPS. Parsing terminates when the configuration $\langle A \rangle$ is generated, where A is the root symbol.

As each action can be performed at any position in the pending list, there are $\mathcal{O}(k \cdot |G|)$ possible actions, where k is the length of the pending list and $|G|$ is the number of nonterminal in the underlying grammar. Though

[4] A head percolation table specifies for each grammar rule X $\rightarrow$ A B how to find the lexical head of X.

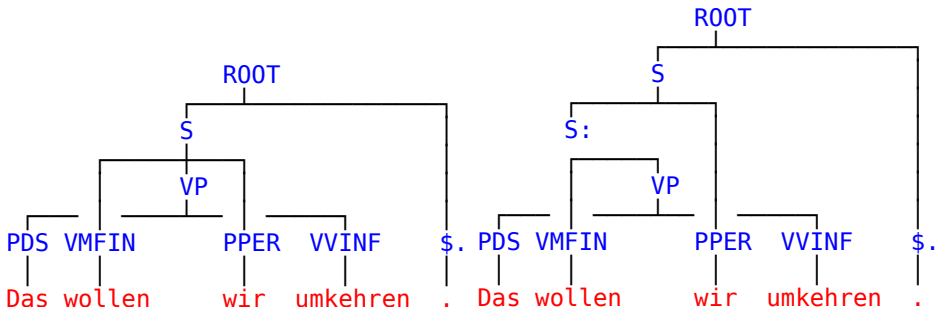


Figure 3.3: Tree from the Tiger treebank and its binarization. *We want to reverse that.*

this seems to incur a high computational cost, in practice, as transitions only modify configurations locally (i.e. in a small node window), the scores of most actions do not change after a transition. Therefore, only a small number of features and action scores have to be recomputed after each parsing step.

Due to the very nature of the strategy, there are a lot of possible derivations for a single tree. A statistical model is responsible for scoring actions and determining which actions will be taken first. Thanks to the ADD-LEFT and ADD-RIGHT actions, the algorithm can derive non-binary trees, bypassing the issues raised by treebank binarization.

3.2.2.3 The Shift-Reduce-Swap Transition System

The Shift-Reduce-Swap transition system (henceforth SR-SWAP), proposed by Maier (2015), is an extension of a lexicalized shift-reduce system such as the one presented earlier (Section 3.2.1, Table 3.2). It relies on the same data structures: a stack of subtrees and a buffer of tokens. The only addition to the standard shift-reduce system is a SWAP action defined as follows:

$$\text{SWAP}(\langle S|s_1|s_0, B\rangle) = \langle S|s_0, s_1|B\rangle$$

that can be performed only if

- s_1 is a preterminal;
- B is not empty;
- s_1 occurs before s_0 in the sentence.

Transitions	Configurations	
	Stack	Buffer
		Das wollen wir umkehren .
SHIFT $\Longrightarrow$	Das	wollen wir umkehren .
SHIFT $\Longrightarrow$	Das wollen	wir umkehren .
SHIFT $\Longrightarrow$	Das wollen wir	umkehren .
SHIFT $\Longrightarrow$	Das wollen wir umkehren	.
SWAP $\Longrightarrow$	Das wollen umkehren	wir .
SWAP $\Longrightarrow$	Das umkehren	wollen wir .
RR-VP $\Longrightarrow$	VP[umkehren]	wollen wir .
SHIFT $\Longrightarrow$	VP[umkehren] wollen	wir .
RR-S: $\Longrightarrow$	S:[wollen]	wir .
SHIFT $\Longrightarrow$	S:[wollen] wir	.
RL-S $\Longrightarrow$	S[wollen]	.
SHIFT $\Longrightarrow$	S[wollen] .	
RL-ROOT $\Longrightarrow$	ROOT[wollen]	

Table 3.4: Derivation for the tree in Figure 3.3 with SR-SWAP (POS tags are ignored).

Oracles and Terminal Reordering A derivation example for a discontinuous tree with SR-SWAP is presented in Table 3.4. This is the derivation given by the oracle described in Maier (2015), which corresponds to a post-order tree traversal and uses a simple reordering strategy (left reordering), which is defined as a recursive procedure in Algorithm 3.

There can be several derivations for the same tree, depending on the reordering strategy (Maier and Lichte, 2016). Alternative reordering strategies are defined by modifying the condition on line 6 of Algorithm 3. These can be conditions on the structure, size, or labels of trees (Maier and Lichte, 2016). The strategy choice may have a substantial effect on parsing because a good reordering strategy minimizes the length of derivations, making them easier to learn for a statistical model.

Derivation Lengths The derivation length for a sentence of size n may widely vary, as a derivation might contain 0 to n unary reductions, and $\mathcal{O}(n^2)$ SWAPS and SHIFTS.[5] Maier (2015) adopts two strategies to handle

[5]A swapped terminal must be shifted again later in the derivation. In some cases, the same terminal must be swapped several times to derive certain trees (see Section 7.4 for an example).

Algorithm 3 Left reordering algorithm, adapted from Maier and Lichte (2016).

```
1: function REORDER(node)
2:     if ISPRETERMINAL(node) then
3:         return node
4:     s1, s2 ← SUBTREES(node)
5:       ▷ INDEX returns the index of the leftmost preterminal in the yield
   of a tree
6:     if INDEX(s1) < INDEX(s2) then
7:         return concat(REORDER(s1), REORDER(s2))
8:     else
9:         return concat(REORDER(s2), REORDER(s1))
```

this issue: the padding strategy of Zhu et al. (2013), presented in Section 3.2.1.5, and a variant of SR-SWAP which uses a COMPOUNDSWAP$_i$ action. This compound action bundles i SWAPS in a single action, which reduces substantially the length of some derivations.

3.2.3 Preprocessing the Trees

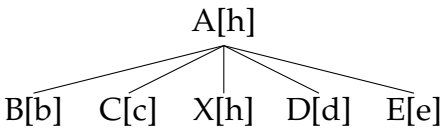
As transition systems derive only trees that satisfy some structural constraints (e.g. binary trees), parsers need to resort to reversible algorithms to preprocess the trees before learning, and post-process the predicted trees before evaluating them. In this section, we present and discuss standard preprocessing algorithms.

3.2.3.1 Binarization

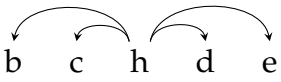
Both the lexicalized shift-reduce transition system and the SR-SWAP transition systems require trees to be binary.[6] We present binarization procedures in Algorithm 4. Figure 3.4 illustrates these binarization strategies for a lexicalized subtree. The simplest binarization algorithm consists in forming a right-branching (3.4c) or left-branching (3.4d) tree by using either BINARIZERIGHT or BINARIZELEFT as the node binarization procedure (Algorithm 4). These are not suitable for lexicalized transition systems as they do not represent adequately constituency headedness information (Figure 3.4b).

In contrast, head-outward binarization (Figures 3.4e and 3.4f) keeps this information, as every binary constituent corresponds to a bilexical

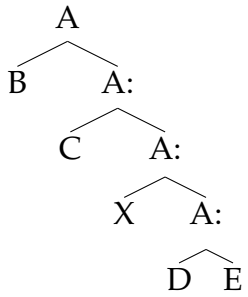
[6]Binarization procedures are identical for projective and discontinuous parsing.



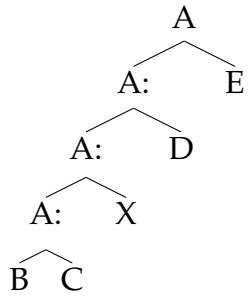
(a) Original lexicalized n-ary tree fragment



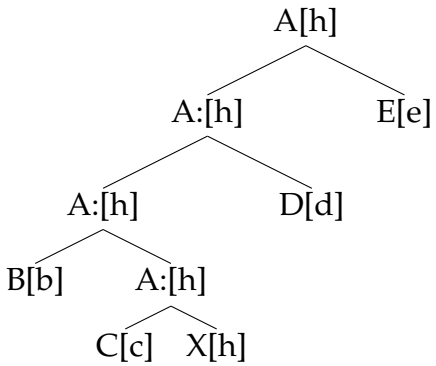
(b) Corresponding dependency subtree



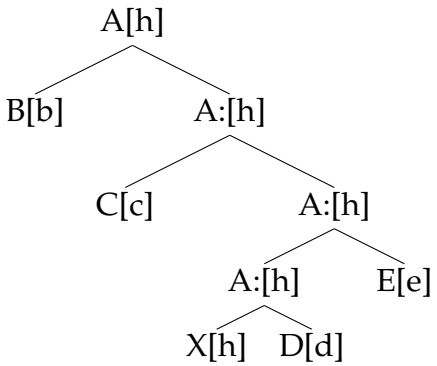
(c) Right-first binarization



(d) Left-first binarization



(e) Head-outward binarization (starting left of head)



(f) Head-outward binarization (starting right of head)

Figure 3.4: Four binarization strategies.

Algorithm 4 Binarization algorithms.

```
1: function BINARIZE(node = { label, children })
2:     for child in children do
3:         BINARIZE(child)
       BINARIZENODE(node)
4:                                  ▷ BINARIZENODE is one of the following functions
5: function BINARIZELEFT(node = {label, children = first|second|rest})
6:     while length(children) > 2 do
7:         newnode ← CREATENODE(GETTMP(label), [first, second])
8:         children ← newnode|rest
9:         first|second|rest ← children
10: function BINARIZERIGHT(node = {label, children = rest|secLast|last})
11:     while length(children) > 2 do
12:         newnode ← CREATENODE(GETTMP(label), [secLast, last])
13:         children ← newnode|rest
14:         rest|secLast|last ← children
15: function BINARIZEHEADOUTWARDLEFT(node ={label, children})
16:     if length(children) > 2 then
17:         headIndex ← FINDHEADINDEX(node)
18:         if headIndex = 0 then
19:             BINARIZELEFT(NODE)
20:         else if headindex = length(children) - 1 then
21:             BINARIZERIGHT(NODE)
22:         else
23:             childrenLeft ← children[:headIndex+1]
24:             childrenRight ← children[headIndex:]
25:             newnode ← CREATENODE(GETTMP(label), childrenLeft)
26:             BINARIZERIGHT(newnode)
27:             children ← newnode|childrenRight
28:             BINARIZELEFT(node)
29: function BINARIZEHEADOUTWARDRIGHT(node ={label, children})
30:     if length(children) > 2 then
31:         headIndex ← FINDHEADINDEX(node)
32:         if headIndex = 0 then
33:             BINARIZELEFT(NODE)
34:         else if headindex = length(children) - 1 then
35:             BINARIZERIGHT(NODE)
36:         else
37:             childrenLeft ← children[:headIndex]
38:             childrenRight ← children[headIndex-1:]
39:             newnode ← CREATENODE(GETTMP(label), childrenRight)
40:             BINARIZELEFT(newnode)
41:             children ← childrenLeft|newnode
42:             BINARIZERIGHT(node)
```

dependency. For example, the tree fragment `(A:[h] B[b] A:[h])` in Figure 3.4c encodes the dependency arc $h \rightarrow b$ of the dependency tree in Figure 3.4b. Head-outward binarization may start either with nodes that precede the lexical head (Figure 3.4e) or follow it (Figure 3.4f). In the context of parsing, left-first head-outward binarization seems a better choice, because it enables the parser to start reducing as soon as the constituent head is available. Another consequence of this choice is that the parser is more incremental in the sense of Nivre (2004): the average length of the stack during a derivation is shorter than it would be with right-first head-outward binarization.

It has to be noted that even with unlexicalized chart parsers, be they CFG parsers or LCFRS parsers, using a head-outward binarization algorithm leads to better results than left-first or right-first binarization (Klein and Manning, 2003; van Cranenburgh et al., 2016). This fact suggests that the organization of a constituent around a lexical head is an important cue for parsing, even when it is not explicitly used as a feature by the statistical model.

If we ignore lexicalization –as do standard evaluators in constituency parsing– the four binarized structures in Figure 3.4 would yield the same n-ary tree. The number of possible binarization for a single n-ary constituent is the number of binary trees with n leaves. However, only a few of those possible binary trees would encode the correct bilexical dependencies.

In the binarization algorithms we have presented, the only temporary symbol for a nonterminal X is X: despite the fact that temporary constituents do not have the same distribution (A: may rewrite as C X or as A: D). In other contexts, it is standard to distinguish different temporary symbols according to their context, by using symbols annotated with their k immediate siblings, k being a bound low enough to avoid an explosion of the size of the resulting grammar. This process is called order-k horizontal Markovization (Klein and Manning, 2003). The head-outward algorithm we presented above is a particular case with $k = 0$. When k is high enough, there is a bijection between original trees and binarized trees.

3.2.3.2 Merging Unary Productions

The tree transformation that is required to avoid the unary constituents that do not produce a preterminal is straightforward: replace each node X that rewrites as Y (where Y is not a preterminal) by a collapsed node X@Y. We illustrate this transformation in Figure 3.1 (right-hand side).

	Tiger (train)		Projectivized Tiger (train)	
	original	preprocessed	original	preprocessed
Number of nonterminal symbols	25	112	26[a]	596
Temporary symbols (types)	0	24	0	22
Temporary symbols (occurrences)	0	343,780	0	378,500
Collapsed symbols (types)	0	63	0	549
Collapsed symbols (occurrences)	0	5,821	0	60,195
Hapaxes	1	22	1	251
Kurtosis	2.5	16.1	2.2	126.5

Table 3.5: Statistics about the symbol distribution differences after treebank preprocessing for two versions of the Tiger treebank.
[a]A ROOT symbol has been added to each sentence to ensure each tree has the same root label.

3.2.3.3 Head Annotation

In order to annotate each constituent in a treebank with its lexical head, the most common approach is to use a head percolation table (Collins, 1999). A head percolation table specifies a set of rules to find the head of a constituent depending on the grammar rule it instantiates. Such tables are also used to convert constituency treebanks to dependency treebanks.

Another method, introduced by Crabbé (2015), consists in projecting dependency arcs from a dependency corpus to a constituency corpus, and using heuristics to solve ambiguous cases. To do so, the two corpora must be aligned at the sentence level.

3.2.3.4 Tree Transformations and Grammar Inflation

Both tree transformations, the binarization and the merging of unary constituents, introduce new symbols: collapsed unary symbols (X@Y@Z), and temporary symbols (X:). In fact, the number of distinct nonterminals in the resulting treebank is often much larger than in the original treebank.

To illustrate this, we plotted the symbol distributions before and after preprocessing in the training sets of two versions of the Tiger treebank. The first version (Figure 3.5) is the original discontinuous corpus (it is the TIGERM15 corpus used in Chapter 7). The second version (Figure 3.6) is a projective treebank automatically converted from the discontinuous treebank by the SPMRL shared task organizers (Seddah et al., 2013).

In both cases, we observe an explosion of the number of nonterminals (more than 20 times as many symbols in the projective case, 5 times in the discontinuous case). In terms of types, the collapsed unaries are the most

numerous nonterminal symbols (Table 3.5). However, in terms of occurrences, the temporary symbols are much more frequent. Both graphs in Figures 3.5 and 3.6 show a very long tail of collapsed unaries. Indeed, the distributions of symbols after the preprocessing have a very high kurtosis[7] compared to the distributions of symbols before preprocessing. In particular, a high proportion of new symbols have only a single occurrence in the preprocessed corpus (20% in the discontinuous case, nearly 50% in the projective case).

The striking difference in the number of collapsed unary symbols in both (preprocessed) versions of the same corpus comes from the automatic projectivization that produces a lot of unary chains (e.g. NP@NP@PP@PP@-NP@S) as artefacts of discontinuous constituents.[8]

Since many symbols may be left-headed or right-headed, or found in unary constituents, the total number of parsing actions exceeds the number of symbols. The SR-SWAP transition system uses 182 different actions (types) to derive the discontinuous version of the treebank. The lexicalized shift-reduce transition system introduced earlier uses 921 actions to derive the projective version. Such unbalanced distributions of labels are hard to learn for a classifier, as it is not likely to infer anything about the distribution of a symbol from a single occurrence.

Several questions arise from these observations. First of all, the design of transition systems that do not require binarization would reduce the number of possible labels and therefore the number of possible actions, which may make predictions easier. Recent proposals in projective constituency parsing explored this possibility with different strategies and obtained very good results (Cross and Huang, 2016b,a; Dyer et al., 2016). We introduce in Chapter 8 a model that generalizes the approach of Cross and Huang (2016a) to discontinuous parsing, using the GAP action introduced in Chapter 7 to predict discontinuous constituents. Secondly, the strategy we presented to handle unary constituents tends to produce a high number of very rare labels. Alternative strategies would allow the parser to perform a unary reduction at any time (Sagae and Lavie, 2005), but it would then need heuristics (e.g. bound on the number of consecutive REDUCE-UNARY) to avoid endless cycles of unary reductions. Finally, it appears that the choice of tree representations has an even bigger impact. In the case of the Tiger treebank, it seems much more reasonable

[7]Kurtosis measures the tailedness of a distribution, a high value indicates a high number of low-frequency items overall.

[8]To a lesser extent, we observe the same issue for other corpora obtained by conversion, for example the version of the Penn Treebank used for constituency parsing has been stripped of empty categories, which produces a lot of unary chains.

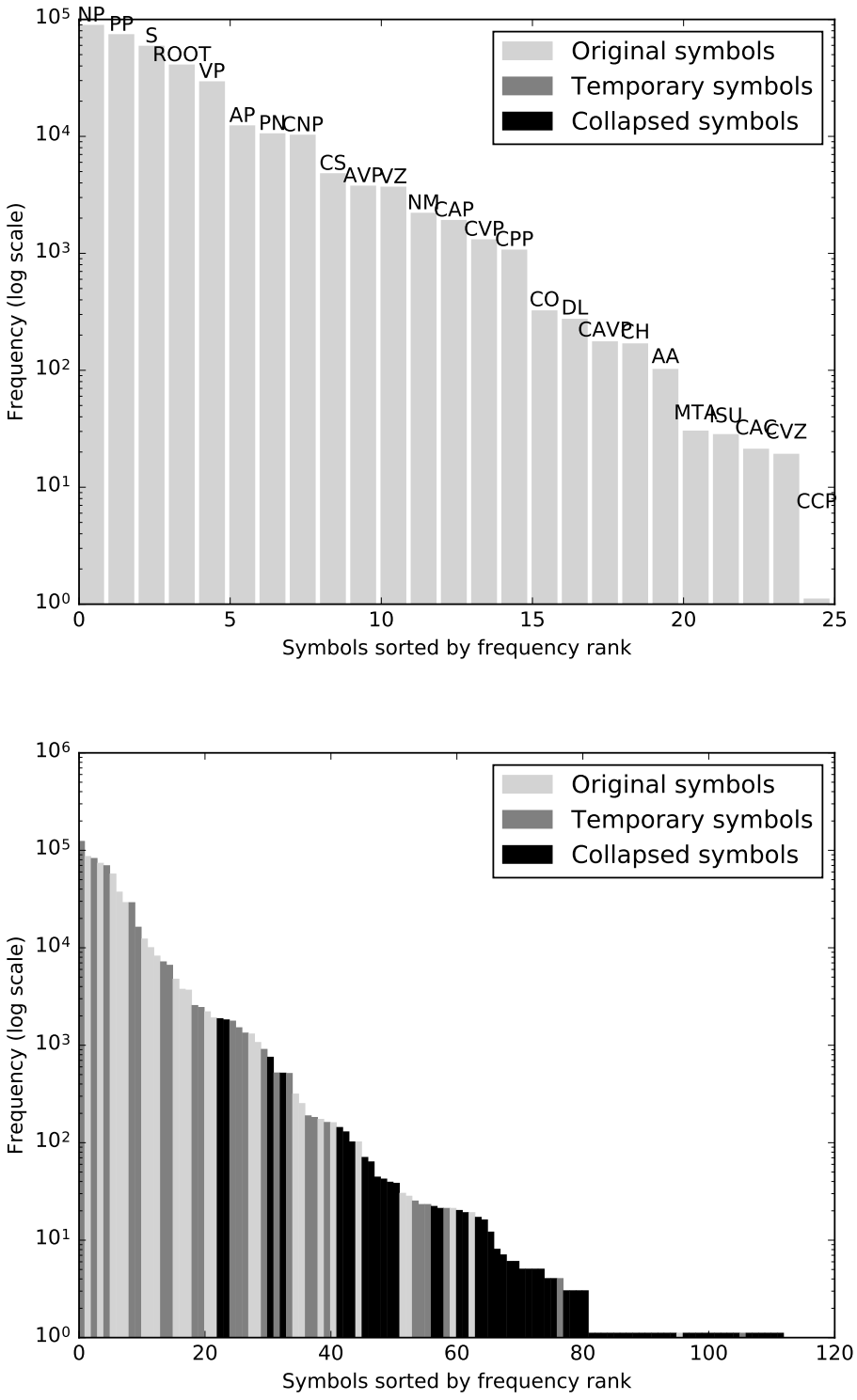


Figure 3.5: Symbol distribution in the Tiger treebank before (upper part) and after (lower part) preprocessing.

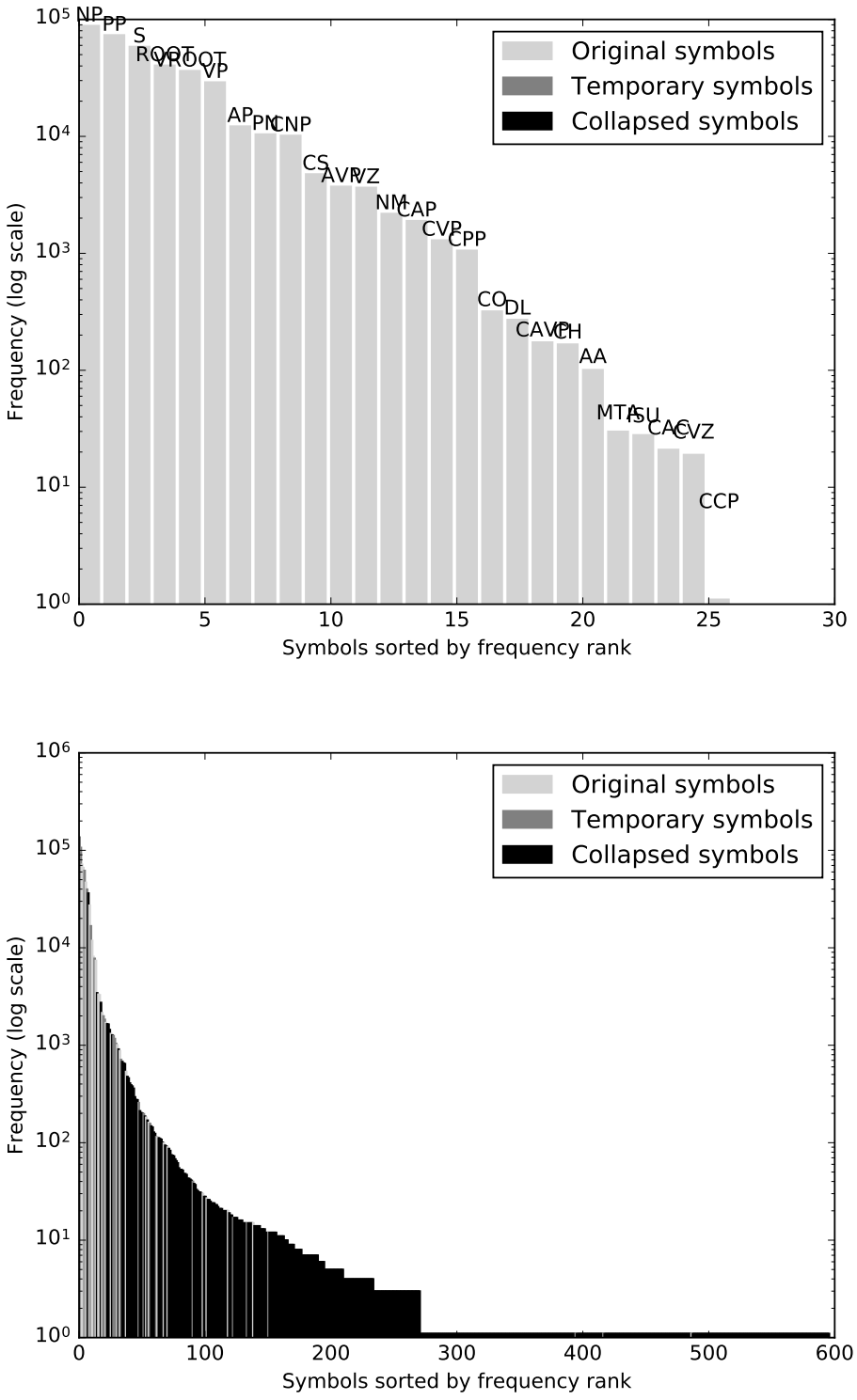


Figure 3.6: Symbol distribution in the SPMRL projectivized version of the Tiger treebank, before (upper part) and after (lower part) preprocessing.

to parse discontinuous constituency trees directly. This is a problem we investigate in Chapter 7.

3.3 Conclusion

This chapter has introduced transition systems for both projective and discontinuous constituency parsing. It has also highlighted typical issues in transition-based constituency parsing: the grammar inflation caused by treebank preprocessing, and the problem of derivation lengths, which may be both detrimental to a classifier. In the next chapter, we present statistical models and search algorithms typically used in constituency parsing.

Chapter 4

Weighted Parsing: Decoding and Learning

Contents

4.1 Introduction **64**
4.2 Searching for the Best Derivation **65**
4.2.1 Structure of the Search Space 65
4.2.2 Search Algorithms 65
4.2.3 Interim Conclusion 70
4.3 A Linear Classifier: the Structured Perceptron **70**
4.3.1 The Structured Perceptron Algorithm 71
4.3.2 Mapping Derivations to Vectors 72
4.3.3 Training a Structured Perceptron with Inexact Search 75
4.4 Structured Prediction with Local Classifiers **76**
4.5 Neural Models **77**
4.5.1 Lexical Data Dispersion 77
4.5.2 Feed-Forward Neural Networks 78
4.5.3 Bidirectional Recurrent Neural Networks 81
4.6 Conclusion **85**

4.1 Introduction

The transition systems that we have introduced in the previous chapter may be viewed as non-deterministic finite state machines. As they do not enforce grammatical constraints (except those related to temporary symbols), they can derive any labelled tree given a set of labels. In other words, transition systems massively overgenerate, which is desirable for robustness, but makes the space of possible trees very large.

To make a transition system deterministic, statistical parsers rely on a scoring system s that gives a weight for every possible derivation t for a sequence of tokens x. With such a function, searching for the best tree is equivalent to solving the following problem:

$$\hat{t} = \underset{t \in \text{GEN}(x)}{\text{argmax}}\, s(x, t), \tag{4.1}$$

where $\text{GEN}(x)$ is the set of all possible derivations for a sentence of length $|x|$. Equation 4.1 formulates the parsing problem as a structured classification problem since both the input x and the output t are structured as sequences of different lengths. In most cases, the scoring system is a classifier and the scoring function s decomposes as a sum of scores of actions:

$$s(x, t) = \sum_{i=1}^{K} s_l(c_{i-1}, a_i) \tag{4.2}$$

where $c_0, \ldots c_K$ is the sequence of configurations successively obtained when applying the sequence of actions $t = (a_1, \ldots, a_K)$ to the initial configuration c_0 for the input sentence x. This decomposition enables the parser to search incrementally for the best derivation as the scores of partial derivations with the same prefix can be factorized.

In this chapter, we make explicit two aspects of equations 4.1: (i) **decoding**: how to solve the equation and (ii) **learning**: how to define s and learn its parameters.

First of all, we describe search algorithms to search for the best derivation (Section 4.2). Then, we discuss how to instantiate the scoring function s and to learn its parameters. In particular, we focus on the structured perceptron (Section 4.3), local classifiers (Section 4.4) and on more recent models based on deep learning (Section 4.5).

4.2 Searching for the Best Derivation

Given a transition system, a scoring function s and a sentence, weighted parsing can be cast as a search problem. The search space of the problem is structured as a tree where each node is a parsing configuration and each arc is a transition weighted by its cost. Figure 4.1 illustrates the search space: c_i^x are configurations and s_i^x are transition scores. The root of the tree c_0 is the initial configuration. The problem is to find the highest-weight path from the initial state to a final state (i.e. any terminal configuration). Before presenting standard search algorithms used in parsing, we briefly describe the structure of the search space.

4.2.1 Structure of the Search Space

Thanks to the design of the transition system and the preconditions on actions, the search space is finite. However, the number of states in the search space grows exponentially with the distance (in number of arcs) to the root. Under certain conditions on the configuration-scoring function s, the search space can be factorized as a Directed Acyclic Graph (Huang and Sagae, 2010).[1] In the general case, we defined s as a function of a whole configuration (and of an action). In practice though, s often only depends on a local region of a configuration c, namely the topmost elements of the stack and the buffer. In such a case, the configurations that are *locally* identical will be given the same transition scores by s. By defining equivalent search states as states that are locally identical –and therefore have the same outgoing scores– and merging them, the search space becomes a DAG. This factorization makes it possible to use dynamic programming in order to explore efficiently a greater part of the search space (Huang and Sagae, 2010; Mi and Huang, 2015), or even perform optimal search (Zhao et al., 2013; Thang et al., 2015) in a reasonable time.

4.2.2 Search Algorithms

In this section, we describe standard search algorithms used in transition-based parsing. We use the following notations throughout the section:

[1] This observation holds for both transition-based dependency parsing (Huang and Sagae, 2010; Zhao et al., 2013) and constituency parsing (Mi and Huang, 2015; Thang et al., 2015).

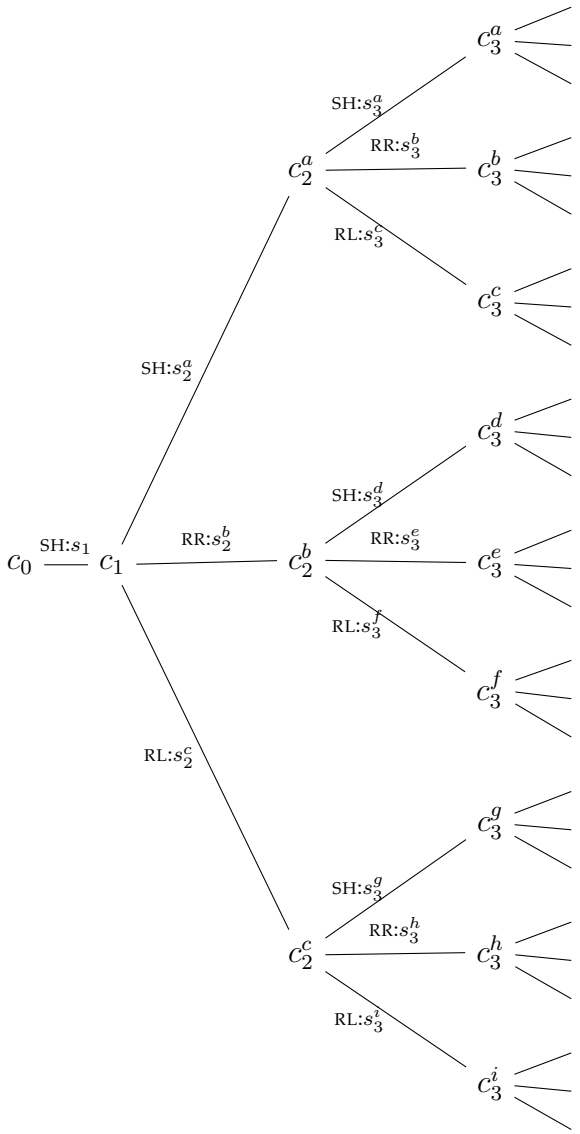


Figure 4.1: Search space illustration. Each edge is labelled by an action and its score. In practice, the branching factor is larger and corresponds to the number of available actions.

- $s(c)$ denotes the score of a configuration, i.e. the sum of the scores of the successive transitions required to reach c. It is sometimes referred to as the **prefix score** as it is the score of the prefix of a derivation.
- NEXT(c) denotes the set of states adjacent to c.
- K-ARGBEST generalizes the argmax operator to the k arguments with the highest values in a given set. For example, for a function f defined as $f(a) = 1$, $f(b) = 0$, $f(c) = 3$, we have

$$2\text{-ARGBEST}_{e \in \{a,b,c\}}\{f(e)\} = \{a, c\}.$$

  1-ARGBEST is the argmax operator.

Moreover, we assume that the transition system uses one of the strategies described in Section 3.2.1.2 to make sure that derivations have the same length and avoid a bias towards long or short derivations.

4.2.2.1 Greedy Search

Definition and Complexity Greedy search is the simplest and most efficient search algorithm. It consists in choosing the single highest scoring action at each parsing step (function GREEDYSEARCH in Algorithm 5). It has a complexity in $\mathcal{O}(m(n))$ where $m(n)$ is the bound on the number of actions for a sentence of size n.[2] The efficiency is the main advantage of greedy search. Nevertheless, it has no guarantee of optimality and is very sensitive to search error: once a single mistake has been made, the parser cannot recover. This effect is particularly detrimental to parsing when the parser has only access to local information and must commit to a hypothesis before being able to access relevant information. We illustrate such bias with an analogy with human sentence processing about the garden path effect.

Locality Biases: Garden Path Effects When searching incrementally, the parser may need to solve ambiguities without having access to the relevant information at the time of the decision. This difficulty is similar to the garden path effect in human sentence processing. The garden path effect arises when the preferred analysis for the first words of the sentence becomes ungrammatical once the following token is processed, and the analysis must be revised. Consider the following sentence: *the*

[2]For projective dependency parsing transition systems, $m(n) \in \mathcal{O}(n)$. For the SR-SWAP transition system, $m(n) \in \mathcal{O}(n^2)$.

government plans to raise taxes were defeated. When hearing the beginning of the sentence, the listener is likely to parse it as a sentence (analysis 1 below). However, when the next token *were* is processed, it cannot be integrated to this analysis. Instead, the processor must revise the analysis of the beginning of the sentence to obtain a grammatical parse (analysis 2).

1. ($_{\text{S}}$ ($_{\text{NP}}$ the government) ($_{\text{VP}}$ plans to raise taxes)
2. ($_{\text{S}}$ ($_{\text{NP}}$ the government plans to raise taxes) ($_{\text{VP}}$ were defeated))

Such sentences causes processing difficulty for humans, e.g. higher reading times.

This effect is also a fundamental issue in incremental transition-based parsing, in particular with a greedy search algorithm. When a greedy parser commits to an analysis, it cannot revise it. In order to alleviate the locality biases, there are two main approaches:

- Using a more complex search algorithm to explore a greater part of the search space and be less likely to dismiss the gold analysis at an early stage of parsing. We describe more elaborate search algorithms in the next paragraphs.
- Using representations that give a global view on the sentence in order to better inform parsing decisions, for example bi-RNN models (Section 4.5.3).

4.2.2.2 Beam Search

Beam search is an approximate breadth-first search algorithm that only keeps the k best hypotheses at each parsing step (function BEAMSEARCH in Figure 5). As beam search has a complexity in $\mathcal{O}(k \cdot m(n))$, linear in the length of a derivation, it is a very popular choice of search algorithm in transition-based parsing (Zhang and Clark, 2009; Zhu et al., 2013; Crabbé, 2015; Watanabe and Sumita, 2015; Wang et al., 2015; Maier, 2015, among many others). The size of the beam k controls the tradeoff between efficiency and accuracy.

When the scoring function s only depends on a local region of a configuration, dynamic programming may be used together with beam search (Mi and Huang, 2015), letting the parser explore a huge part of the search space while still being fast in practice. Indeed, Huang and Sagae (2010) and Mi and Huang (2015) report empirical linear time parsing.

Algorithm 5 Search algorithms for parsing.

```
1: function GREEDYSEARCH(c)                                  ▷ c: initial state
2:     while c is not a goal state do
3:         c ← argmax_{c'∈NEXT(c)} s(c')
4:     return c
5: function BEAMSEARCH(c, k)                                 ▷ k: size of beam
6:     B ← {c}                                               ▷ Current beam
7:     while ∃c ∈ B st c is not a goal state do
8:         H ← ⋃_{c∈B} {c' ∈ NEXT(c)}
9:         B ← K-ARGBEST_{c∈H} {s(c)}
10:    return argmax_{c∈B} s(c)
11: function BESTFIRST(c)
12:    Q ← MAXPRIORITYQUEUE()
13:    ADD(Q, c, 0)
14:    while c ← POP(Q) is not a goal state do
15:        for c' ∈ NEXT(c) do
16:            ADD(Q, c', s(c'))
17:    return c
```

Beam search is not suited to unnormalized locally trained models.[3] Zhang and Nivre (2012) showed that a locally trained perceptron obtains its best results with greedy search and that its performance drastically decreases with the size of the beam. However, beam search may improve the performance of normalized locally trained models such as the feed-forward neural network that we present in Section 4.5.2.

4.2.2.3 Best-First Search

The best-first search strategy (function BESTFIRST in Algorithm 5) consists in exploring the most promising hypotheses first. The parser maintains a priority queue of search states using their scores as priority keys. At each step, it removes the best state from the priority queue, expands it to new candidate states and adds them onto the queue. It returns the first final configuration extracted from the priority queue.

Best-first search is optimal under the condition that the search space satisfies the superiority property. This property holds when, for two states c and $c' \in \text{NEXT}(c)$, $s(c') < s(c)$. In other words, scores should get worse after each transition. It is the case when $s(c)$ is the sum of the

[3] See also Section 4.4 for a discussion of local classifiers.

log probabilities of actions leading to c, for example when the classifier is a MaxEnt model (Sagae and Lavie, 2006; Mi and Huang, 2015). When the scoring function is based on an unnormalized model, such as a perceptron, heuristics can make sure that the search space satisfies the superiority condition (Thang et al., 2015). In fact, best-first search in a space that satisfies the superiority condition can be reduced to an instance of Dijkstra algorithm for finding a shortest path, by trying to minimize $-s(c)$, instead of maximizing $s(c)$.

A variant of best-first, A*, uses as priority keys the sum of the state score and of a heuristic cost. Under certain conditions on the heuristic cost, A* has the same optimality guarantees as best-first search while being faster as it explores fewer search states. Together with dynamic programming, these search algorithms have been used to perform optimal search in transition-based constituency parsing (Thang et al., 2015). However, the complexity of exact search is high: $\mathcal{O}(n^3)$ to $\mathcal{O}(n^6)$ depending on the feature templates (Thang et al., 2015).

4.2.3 Interim Conclusion

We have presented algorithms to search for the best tree for a given sentence. In this dissertation, we will focus on fast approximate algorithms, such as greedy search or, sometimes, beam search. We now turn to the scoring function $s(c)$ and describe standard statistical models to define it and learn its parameters.

4.3 A Linear Classifier: the Structured Perceptron

The structured perceptron (Collins, 2002) is an algorithm that aims at learning a linear scoring function s for structured prediction problems. It is widely used in constituency parsing (Zhang and Clark, 2009; Zhu et al., 2013; Crabbé, 2014, 2015; Thang et al., 2015, among many others) and discontinuous constituency parsing (Versley, 2014b; Maier, 2015; Coavoux and Crabbé, 2017a). The structured perceptron, in contrast to more recent neural models, is rather fast to train and has much fewer hyperparameters (essentially the number of iterations and the beam size).

We first introduce the learning algorithm (Section 4.3.1). Then, we describe how to obtain vector representations of derivations (Section 4.3.2).

Finally, we discuss how to adapt the learning algorithm when only approximate inference is available (Section 4.3.3).

4.3.1 The Structured Perceptron Algorithm

The structured perceptron algorithm[4] (Collins, 2002) was introduced as an extension of the perceptron (Rosenblatt, 1958) to the problem of structured classification. A linear scoring function s has the following formulation:

$$s(x, y) = \mathbf{w} \cdot \mathbf{\Phi}(x, y)$$

where $\mathbf{\Phi}$ is a feature function (see Section 4.3.2) and $\mathbf{w}$ is a parameter vector. The structured perceptron is an algorithm to estimate $\mathbf{w}$ from a set of training example $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^{N}$, where each $x^{(i)}$ is a sequence of (possibly tagged) tokens, and each $y^{(i)}$ is the transition sequence for deriving the corresponding gold tree.[5]

Algorithm 6 Global averaged perceptron algorithm.

function PERCEPTRONLEARN($\{x^{(i)}, y^{(i)}\}_{i=1}^{N}$, $\mathbf{\Phi}$, Epochs)
  $k \leftarrow 1$
  $\mathbf{w}^{(k)} \leftarrow (0, \ldots, 0)$
  for e = 1 to Epochs **do**
    for i = 1 to N **do**
      $\hat{y} = \text{argmax}_{y \in \text{GEN}(x^{(i)})} \mathbf{w}^{(k)} \cdot \mathbf{\Phi}(x^{(i)}, y)$
      $k \leftarrow k + 1$
      if $\hat{y} \neq y^{(i)}$ **then**
        $\mathbf{w}^{(k)} \leftarrow \mathbf{w}^{(k-1)} + \mathbf{\Phi}(x^{(i)}, y^{(i)}) - \mathbf{\Phi}(x^{(i)}, \hat{y})$
      else
        $\mathbf{w}^{(k)} \leftarrow \mathbf{w}^{(k-1)}$
  return $\frac{1}{k} \sum_{i=1}^{k} \mathbf{w}^{(i)}$

The perceptron algorithm (Algorithm 6) iterates several times over the data (or until convergence). At each step, it makes a prediction for the current example with the current parameters $\mathbf{w}^{(k)}$ and updates $\mathbf{w}$ if the prediction was incorrect.

The function $\mathbf{\Phi}$ maps the sequence of tokens $x^{(i)}$ and a derivation y to a real-valued vector (see following section). In the plain perceptron

[4] Sometimes called *global perceptron*.

[5] We assume that $y^{(i)}$ is computed by a deterministic oracle: if there are several gold derivations for the same tree, the oracle only returns one of them, and solves ambiguity in a consistent way.

algorithm, the learner would return $\mathbf{w}^{(k)}$. Instead, on the last line of Algorithm 6, it returns the average of every successive set of parameters $\frac{1}{k}\sum_{i=1}^{k}\mathbf{w}^{(i)}$. The averaged perceptron usually gives much better generalization in practice. In the case where the data is linearly separable, the perceptron will converge to 0 error on the training set. However, to avoid overfitting, it is usually best to stop training before the error on the development set starts to increase.

The perceptron algorithm may be viewed as a particular case of stochastic gradient descent (SGD) to optimize the following loss:

$$L(\mathbf{w};\mathcal{D}) = \sum_{i=1}^{N} \max\left\{0, \max_{y\in \text{GEN}(x^{(i)})}\{\mathbf{w}\cdot\mathbf{\Phi}(x^{(i)},y)\} - \mathbf{w}\cdot\mathbf{\Phi}(x^{(i)},y^{(i)})\right\} \quad (4.3)$$

which is known as the structured perceptron loss. This loss is a variant of the hinge loss used in max-margin classifiers such as Support Vector Machines (SVMs), that enforces a null margin.

In the following paragraphs, we describe how to define the function $\mathbf{\Phi}$ first, and then how to adapt the algorithm in the case of inexact search, that is in situations where instead of solving $\text{argmax}_{y\in\text{GEN}(x^{(i)})}\,\mathbf{w}\cdot\mathbf{\Phi}(x^{(i)},y)$ on line 6 of the algorithm, $\hat{y}$ is calculated by approximate search, as is generally the case in parsing.

4.3.2 Mapping Derivations to Vectors

In order to apply the perceptron to the training data, we need a function $\mathbf{\Phi}$ that maps a whole derivation to a real-valued vector that represents it. The global feature representation $\mathbf{\Phi}$ decomposes as a sum of local feature representation $\mathbf{\Phi}_l$:

$$\mathbf{\Phi}(x,y) = \sum_{i=1}^{K} \mathbf{\Phi}_l(c_{i-1}, y_i) \quad (4.4)$$

where c_i is the configuration obtained by applying the first i transitions y_1^i of $\mathbf{y}$ to the sentence x. This decomposition into local feature representation makes it possible to search incrementally for the best derivation as the scores of different derivations with the same prefix can be factorized.

The function $\mathbf{\Phi}_l$ is said to be local because it only depends on a local region of a parsing configuration, i.e. on the topmost elements in the stack and the first elements in the buffer. Each coefficient of $\mathbf{\Phi}_l(c,y)$ is valued by a feature function that specifies that the action to be scored is y and that a conjunction of conditions is satisfied such as:

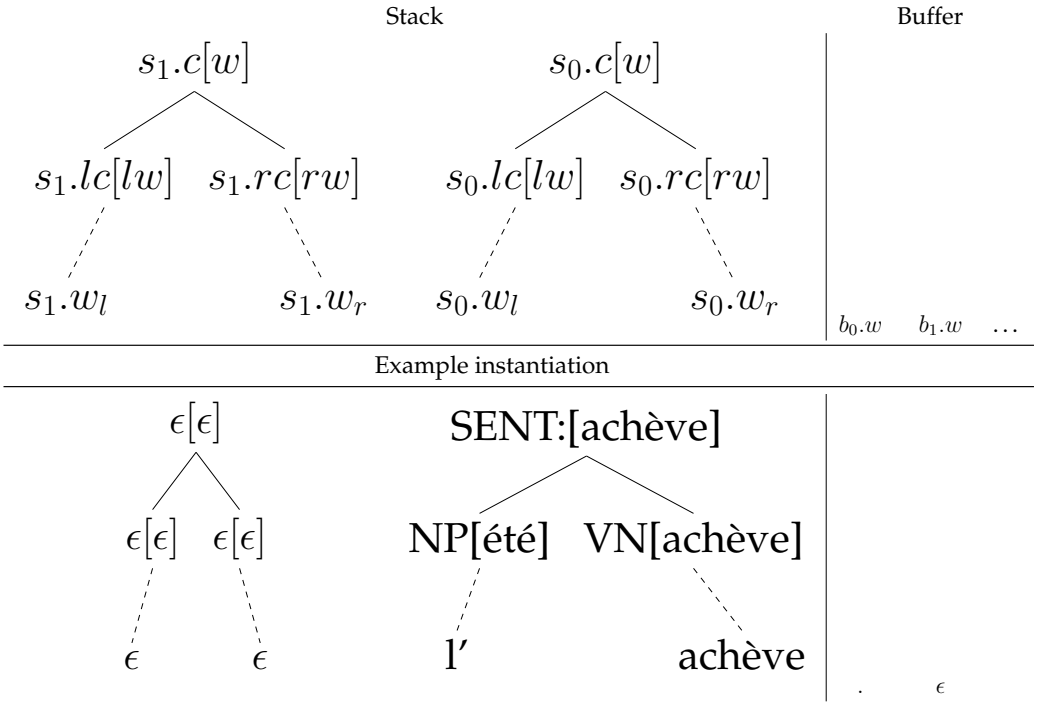


Figure 4.2: Illustration of the locality of an abstract configuration and a concrete configuration when deriving the tree in Figure 3.1 (lower left-hand part).

- **Unigram feature function**: the nonterminal symbol of the first element of the stack (s_0) is NP.
- **Bigram feature function**: the word form at the top of the buffer (b_0) is `cat` and its POS tag is NC.

A conjunction of conditions aims at capturing feature interactions. These functions can be arbitrarily complex and focus on any information available from the configuration. In particular, they could be conditions of the morphological attributes of tokens when available ("the case of b_0 is accusative"). A practical limit to the complexity of feature functions is the number of conjoined conditions: due to the sparsity of the n-gram feature functions when n is too high, standard sets of feature functions only use up to 4-gram conditions.

In order to specify concisely a set of feature functions to define $\mathbf{\Phi}_l$, it is customary to describe **feature templates**. Feature templates are high-level specifications for a set of feature functions. Most of the time, they are defined as a position (or a conjunction of positions) in a configuration. For

example, the template $s_0.c$ represents N feature function of the form "the nonterminal at the top of s_0 is X" where N is the number of distinct nonterminals in the treebank and X is a variable for a nonterminal. Similarly, the template $s_0.rc\&s_0.rw\&b_0.w$ represents $N \times V^2$ feature functions of the form "the nonterminal which is the right child of s_0 is X, its lexical head is Y and the word form of b_0 is Z", where V is the size of the vocabulary and X, Y and Z are variables.

Figure 4.2 introduces notations for local positions in a configuration. Traditionally, standard feature templates (Zhu et al., 2013) focus on the top four nodes in the stack (s_0 to s_3) and on the first four elements in the buffer (b_0 to b_3). Zhu et al. (2013) observed that features on the left and right children of s_0 and s_1 are also informative. Following Hall et al. (2014), Crabbé (2015) used additional span features, i.e. features valued by the tokens at the right and left boundaries of constituents in the stack (denoted by $s_i.w_l$ and $s_i.w_r$ in Figure 4.2) because the boundaries of constituents often contain useful cues, such as function words.

Although the type of representation induced by $\mathbf{\Phi}_l$ has been very successful for numerous tasks in Natural Language Processing in general, they have several limitations that we discuss here. First of all, finding a good set of feature templates for a task –feature engineering– is generally a difficult problem. For n atomic pieces of information available in a configuration, there are n^3 potential trigram templates.[6] The problem becomes all the more complex when a lot of different information types are available, increasing the number of potential atomic features in a configuration. For example, when the parser has access to morphological attributes for each token[7] (Crabbé, 2015). Although some systematic feature selection methods may be used (Bawden and Crabbé, 2016), they are often relatively slow.

Representations based on feature functions are very sparse. $\mathbf{\Phi}$ typically has millions of coefficients, but only a handful of non-zero coefficients. The underlying representation of the lexicon is entirely symbolic: the lexicon is a set of symbols with no relationship between them. Two features $s_0.w = \texttt{achève}$ and $s_0.w = \texttt{termine}$ will be coded on different dimensions of $\mathbf{\Phi}$. Different feature templates with the same value will also be encoded on different dimensions (for example $s_0.w = \texttt{achève}$ and $s_0.w_r = \texttt{achève}$, Figure 4.3). Conjunctions of features might encode bilexical dependencies but they are very hard to estimate due to data

[6] A typical set of feature templates contains 40 to 200 templates.

[7] In the SPMRL dataset (Seddah et al., 2013), depending on the language, annotations for 7 (German) to more than 20 (Basque) morphological attributes are available.

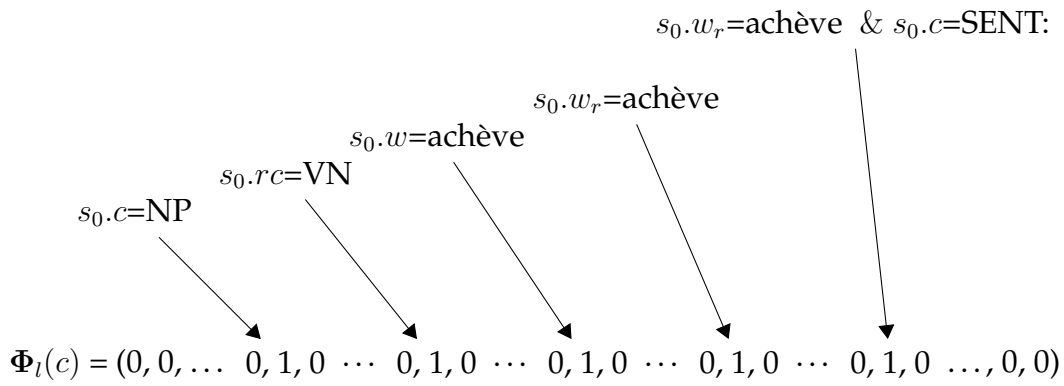


Figure 4.3: Instantiation of $\mathbf{\Phi}$ (only a few features are shown).

sparsity. In other words, the lexicon is seen as an unstructured object, that completely ignores the notion of similarity between symbols.

4.3.3 Training a Structured Perceptron with Inexact Search

The standard training of a structured perceptron requires exact inference to solve equation 4.1 at each training step. However, although it is possible to find the highest-scoring derivation in reasonable time under certain conditions using dynamic programming (Thang et al., 2015), most parsers use an approximate search algorithm for efficiency (see Section 4.2). Under approximate inference, the convergence guarantees of the perceptron do no longer hold.

Let us assume that the parser solves the problem on line 6 of Algorithm 6 with a beam search, which does not provide optimality guarantees. In such a case, the learner might make invalid updates. An invalid update (Huang et al., 2012) happens when the gold sequence y is the highest scoring sequence but was not found by the search algorithm. The predicted sequence $\hat{y}$ is such that:

$$\mathbf{w} \cdot \mathbf{\Phi}(x, \hat{y}) < \mathbf{w} \cdot \mathbf{\Phi}(x, y)$$

In such a case, the update rule (line 9) will give a bonus to the gold sequence and a penalty to the predicted sequence, which would not have happened with exact decoding.

In order to maintain the convergence properties of the perceptron, the trainer must only do valid updates (Huang et al., 2012). The condition for a valid update with a predicted sequence $\hat{y}$ and a gold sequence y is that $\mathbf{w} \cdot \mathbf{\Phi}(x, \hat{y}) > \mathbf{w} \cdot \mathbf{\Phi}(x, y)$. In other words, the predicted sequence must

have a higher score than the gold sequence. A heuristic to make sure that every update satisfies this condition is the **early update** strategy (Collins and Roark, 2004) that consists in making an update on partial sequences as soon as the gold derivation falls out of the beam. [8]

4.4 Structured Prediction with Local Classifiers

Although transition-based parsing is a structured prediction problem, it can be reduced to an unstructured problem by considering that the data is not a set of sentence-tree couples, but a set $\mathcal{D} = \{(c^{(i)}, a^{(i)})\}_{i=1}^{M}$ of configurations and actions (that can be extracted statically from the treebank using an oracle). In such a case, the representation $\mathbf{\Phi}_l$ for a configuration would not change, nor the form of the scoring function:

$$s(x,t) = \sum_{i=1}^{K} \mathbf{w} \cdot \mathbf{\Phi}_l(c_{i-1}, y_i) \tag{4.5}$$

What fundamentally changes with respect to the global perceptron is the objective function. Instead of trying to optimize the scores of whole derivations for given sentences, the trainer would optimize the scores of single actions for given configurations:

$$L(\mathbf{w}; \mathcal{D}) = \sum_{i=1}^{M} \max\left\{0, \max_{a \in A}\{\mathbf{w} \cdot \mathbf{\Phi}_l(c^{(i)}, a)\} - \mathbf{w} \cdot \mathbf{\Phi}_l(c^{(i)}, a^{(i)})\right\} \tag{4.6}$$

where A is the set of all possible actions.

The main advantage of local classifiers is that they are faster to train and easier to implement. However, they suffer a lot from error propagation and locality biases.[9] As they only see gold configurations at training time, they fail to learn how to behave with noisy configurations, i.e. with configurations that result from wrong parsing decisions.

There are few works using a local classifier for transition-based constituency parsing (Sagae and Lavie, 2005) since structured classifiers (most often structured perceptrons) perform much better and are still relatively fast to train. However, recently, approaches with non-linear classifiers such as neural networks obtained good results despite these locality biases (Section 4.5.2.4).

[8] See Huang et al. (2012) for the theoretical justification for the early update strategy as well as the description of the *max-violation* update rule that ensures that every update is valid.

[9] A way to improve the training of local classifiers is to let the parser explore the search space during training. We investigate this method in Chapter 5.

4.5 Neural Models

This section presents neural models for parsing. Early attempts at parsing with neural networks date back to the beginning of the 2000s (Mayberry III and Miikkulainen, 1999; Lane and Henderson, 2001; Henderson, 2003, 2004; Titov and Henderson, 2007). However, these proposals precede the *deep learning tsunami* (Manning, 2015) and the popularization of word embeddings in NLP. They use word representations that limit the full potential of neural network methods. More recently, Collobert (2011) introduced a parser based on a cascade of chunkers that used distributed word representations. Other recent proposals construct compositional representations of subtrees during parsing (Socher et al., 2010; Stenetorp, 2013; Socher et al., 2013).

We first present the issue of lexical data dispersion in parsing that motivates the use of neural models (Section 4.5.1). We go on to describe a simple feed-forward architecture (Section 4.5.2) and a bidirectional recurrent neural network model (Section 4.5.3).

4.5.1 Lexical Data Dispersion

An important problem in NLP, and in particular in parsing, is the scarcity of lexical data. Due to the Zipfian distribution of word forms in textual data, few (closed-class) word forms are very frequent and most word forms have very few occurrences, as illustrated by the proportion of hapaxes in corpora (Table 4.1).

For a parser, it is difficult to infer generalizations about rare words. The problem is exacerbated in languages with a rich inflectional morphology as they exhibit a higher type-token ratio and a higher Out-Of-Vocabulary (OOV) rate. Among the 9 languages in the SPMRL dataset (Seddah et al., 2013), 6 have an OOV rate (types) higher than 30 percent, whereas it is only 12.8 for the Penn Treebank. For 5 languages, more than 10% of tokens in the development set are unknown word forms. The amount of data about bilexical dependencies is even scarcer.

In order to limitate the consequences of the lexical data sparsity problem, the choice of representation for words is crucial, in particular, the shift to vector representations for words and the use of morphological information to better represent unknown tokens.

	SPMRL									PTB
	Arabic	Basque	French	German	Hebrew	Hungarian	Korean	Swedish	Polish	English
					Train					
Tokens	589,220	96,368	443,113	719,530	128,046	170,141	296,446	76,332	66,778	950,028
Types	36,906	25,136	27,470	77,220	15,971	40,782	85,671	14,100	21,793	44,389
Hapax rate (word types)	41.0	67.1	46.0	58.7	50.9	67.8	69.7	61.7	72.7	46.5
					Dev					
Tokens	73,932	13,851	38,820	76,704	11,301	29,989	25,278	9,339	8,382	4,0117
Types	12,342	5,551	6,695	15,852	3,175	10,673	12,164	2,689	4,269	6,840
OOV rate (word occurrences)	3.8	18.4	3.2	7.6	9.6	19.9	26.6	11.9	24.8	2.8
OOV rate (word types)	18.5	41.0	16.2	30.9	27.0	48.9	48.2	35.8	47.2	12.8

Table 4.1: Lexical data sparsity in the SPMRL dataset (Seddah et al., 2013) and the Penn Treebank.

4.5.2 Feed-Forward Neural Networks

In this section, we motivate and describe an instantiation of the scoring function s as a feed-forward neural network, corresponding to the transposition to constituency parsing of the model of Chen and Manning (2014) for dependency parsing.

The two main differences between this model and a linear discriminative classifier, such as a local perceptron, are (i) that feature templates are valued by word embeddings, instead of one-hot vectors, and (ii) the non-linearity added by hidden layers. Neural networks address several issues related to the sparse representation of configurations $\mathbf{\Phi}_l$ identified in Section 4.3.2:

- **Structured lexicon**: by using symbol embeddings as the primitive units to represent a configuration, neural networks make it possible to exploit information about the similarity between symbols and thus alleviate the data sparseness problem.
- **Feature learning**: the hierarchical structure of deep neural networks aims at learning automatically what feature combinations are informative for parsing, bypassing the problem of defining complex feature templates.
- **Parameter sharing**: the same atomic symbol has a represention that does not depend on its position in the configuration. The embedding matrices are shared across positions, which limits data sparseness.

4.5.2.1 Mapping Configurations to Vectors

The representation of a configuration $\mathbf{h}^{(0)}$ –the input to the network– is the concatenation of k embeddings of symbols extracted from the configura-

tion. The specification of these symbols is determined by unigram feature templates, i.e. by k distinct typed positions such as those in Figure 4.2. Instead of being valued by one-hot vectors, as would be implicitly the case when using feature functions, these templates are valued by the embedding corresponding to the symbol that occupies the specified position.[10] There are as many embedding matrices as there are types of symbol used to form $\mathbf{h}^{(0)}$ (nonterminal, word form, POS tag, morphological attributes, etc.), and these matrices are parameters of the model, that can either be initialized randomly or with pretrained embeddings.

4.5.2.2 A Feed-Forward Architecture

The architecture we describe is a local classifier aiming at modelling $P(a|c)$, the probability distribution of actions given a configuration c. Given an input vector $\mathbf{h}^{(0)}$ computed for a configuration c, the feed-forward network computes the probability of each possible action with the following steps:

$$\mathbf{h}^{(1)} = f(\mathbf{W}^{(1)} \cdot \mathbf{h}^{(0)} + \mathbf{b}^{(1)}) \tag{4.7}$$

$$\mathbf{h}^{(2)} = f(\mathbf{W}^{(2)} \cdot \mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \tag{4.8}$$

$$\mathbf{h}^{(o)} = \text{Softmax}(\mathbf{W}^{(o)} \cdot \mathbf{h}^{(2)} + \mathbf{b}^{(o)}) \tag{4.9}$$

$$s(c, a) = \log P(a|c) \tag{4.10}$$

$$= \log h_a^{(o)} \tag{4.11}$$

where $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$ are parameters, $\mathbf{h}^{(i)}$ are successive layers of the network, f is a non-linear activation function, and the Softmax function normalizes a vector to a distribution with the following operation:

$$\text{Softmax}(\mathbf{v}) = \frac{\exp(\mathbf{v})}{\sum_{i=1}^{k} \exp(\mathbf{v}_i)}$$

The number of non-linear layers and their sizes are hyperparameters of the model. In practice, a frequent choice is to use two hidden layers as shown in the equations above. When the score of an action is given by its log probability, the additive model defined in equation 4.2 still applies. Assuming the probability of a derivation is the product of the probabilities of its actions, the parser searches for the highest probability derivation.

[10] The lookup may be viewed as the product $\mathbf{E}^{(t)} \cdot \mathbf{e}_s$ where $\mathbf{E}^{(t)}$ is the embedding matrix for type t and $\mathbf{e}_s$ is a one-hot vector for symbol s of type t, which makes embedding matrices explicit as parameters of the model.

4.5.2.3 Objective Function and Training

The feed-forward network we presented in the previous paragraph is a local model as it sees the data as a set of couples of configurations and actions $\mathcal{D} = \{(c^{(i)}, a^{(i)})\}_{i=1}^{M}$. The parameters $\boldsymbol{\theta}$ of the model are usually trained by stochastic gradient descent (SGD)[11] to minimize the following objective function:

$$L(\boldsymbol{\theta}; \mathcal{D}) = -\sum_{i=1}^{M} \log P(a^{(i)}|c^{(i)}; \boldsymbol{\theta}) \tag{4.12}$$

4.5.2.4 Discussion

Despite being subject to important locality biases, this architecture can obtain good results in terms of accuracy.[12] For example, such models outperform structured perceptrons (Coavoux and Crabbé, 2016; Coavoux and Crabbé, 2016) and can be much faster at test time, since some intermediary results (for the first hidden layer) can be precomputed in the initialization step of the parser.

Both the change to symbol embedding representations and the use of non-linear hidden layers are important for achieving good results. We show in Coavoux and Crabbé (2015) that a linear model where the output layer is computed directly from the concatenation of input embeddings $\mathbf{h}^{(0)}$ performed rather poorly. This observation suggests that taking into account interactions between features, which is done with bigram and trigram feature functions in a perceptron, is important for achieving good results and that non-linear hidden layers play this role in neural network architectures.

In order to extend this feed-forward architecture to a global model, Weiss et al. (2015) introduced a model for dependency parsing that uses a pretrained feed-forward neural network, similar to the one we presented, as a tool to build vector representations for a global perceptron. For a given configuration c, they use the concatenation $[\mathbf{h}^{(1)}; \mathbf{h}^{(2)}; \mathbf{h}^{(o)}]$ as the input of a global perceptron. This model achieved good accuracies in dependency parsing. We adapted it to constituency parsing and tested it on the French Treebank and Penn Treebank (Coavoux and Crabbé, 2016). The results showed that the global perceptron trained with a beam of 16 achieved slightly better accuracy that the feed-forward greedy model, i.e. a

[11] Or other gradient-descent-based methods.

[12] This section discusses preliminary results of our PhD research (Coavoux and Crabbé, 2015; Coavoux and Crabbé, 2016).

much smaller improvement than what was observed by Weiss et al. (2015) in dependency parsing.

4.5.3 Bidirectional Recurrent Neural Networks

Feed-forward neural networks obtained very good results in dependency (Chen and Manning, 2014; Weiss et al., 2015) and constituency parsing (Wang et al., 2015; Coavoux and Crabbé, 2016) but are still subject to locality biases since the objective function is only defined and optimized for couples made of a configuration and an action.

To mitigate the effects of locality, two complementary strategies consist either in using a better search algorithm that explores a larger part of the search space, or using a global statistical model. For the latter strategies, different approaches were introduced that are based on a global objective function. A first approach consists in learning to search during training (Weiss et al., 2015; Watanabe and Sumita, 2015; Zhou et al., 2015; Andor et al., 2016). These proposals use either a perceptron loss or a globally normalized cross-entropy loss, and require beam search at training time. Another approach aims at conditioning the score of an action on a whole configuration (Dyer et al., 2015; Kiperwasser and Goldberg, 2016; Cross and Huang, 2016b) by exploiting the ability of recurrent neural networks to construct fixed-size vector representations of arbitrarily long sequences.

In this section we describe the scoring model of Cross and Huang (2016b) that we extend in Chapter 6 to morphological analysis. It models the probability of a derivation a_1^K for a sentence x_1^n as:

$$P(a_1^K|x_1^n) = \prod_{i=1}^{K} P(a_i|a_1^{i-1}, x_1^n) \tag{4.13}$$

The architecture consists in two components that are trained jointly. The first component is a bidirectional Recurrent Neural Network (bi-RNN)[13] encoder that constructs context-aware representation for every token in the sentence. The second component is very similar to the feed-forward network presented above, but replaces the symbol embeddings in its input by the output of the bi-RNN encoder.

[13]We use RNN as a generic term which includes simple RNN (Elman, 1990) and more complex types of recurrent cells such as Gated Recurrent Unit (Cho et al., 2014, GRU) or Long Short-Term Memory Network (Hochreiter and Schmidhuber, 1997, LSTM).

4.5.3.1 Building Context-aware Embeddings with a Bi-RNN

Motivations The bi-RNN encoder component aims at addressing two issues of the feed-forward model of Section 4.5.2. First of all, the lexical representations used by the feed-forward network are static. They only depend on the word form (and possibly its associated symbols, be they POS tags or morphological attributes), but not on the context. However, the context of occurrence of a word is important because it may help disambiguate ambiguous word forms, or capture information about the properties of an unknown word form. Secondly, the feed-forward parsing model has only a bounded view on the buffer, which is particularly detrimental to parsing when the parser must make decisions before having access to the relevant information that might be arbitrarily far in the buffer. The bi-RNN encoder constructs token representations that depends on the whole sentence. Thus, the whole buffer may be represented by a single vector.

Recurrent Neural Network Encoder A Recurrent Neural Network (RNN) encoder is a network that constructs fixed size representations of arbitrary length sequences. The input to an RNN is a sequence of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$. The RNN maintains a state vector $\mathbf{s}_i$ initialized to a parameter vector $\mathbf{s}_{init}$ and updates iteratively $\mathbf{s}_i$ at each new input vector $\mathbf{v}_i$. The simplest RNN type, Elman's network (Elman, 1990), uses the following update rule:

$$\mathbf{s}_{i+1} = g(\mathbf{W}^{(r)} \cdot \mathbf{s}_i + \mathbf{W}^{(i)} \cdot \mathbf{v}_i + \mathbf{b}) \tag{4.14}$$

where $\mathbf{W}^{(r)}$, $\mathbf{W}^{(i)}$ and $\mathbf{b}$ are parameters for, respectively, the recurrent connections, the input connections and a bias; and g is a non-linear activation function. Elman's networks are hard to optimize, due to the vanishing gradient problem (repeated multiplications during backpropagation makes the gradient either vanish or explode for some state vectors). For this reason, it is customary to use other RNN cell types designed to avoid this issue, such as Long Short-Term Memory Networks[14] (Hochreiter and Schmidhuber, 1997, LSTM).

The output of the encoder is the sequence of state vectors $\mathbf{s}_1, \ldots, \mathbf{s}_n$. Each $\mathbf{s}_i$ is the representation of the first i elements of the input sequence. Thus, an RNN encoder constructs representations for each possible prefix of the input sequence in $\mathcal{O}(n)$. The last state $\mathbf{s}_n$ represents the whole sequence.

[14]We refer the reader to Goldberg (2015) for an overview of RNNs in NLP.

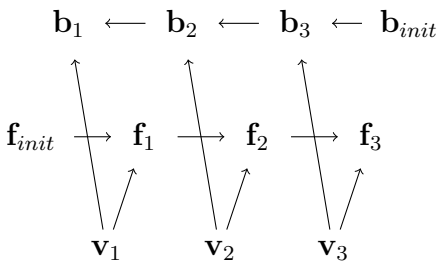


Figure 4.4: Bi-RNN illustration on a three-element sequence.

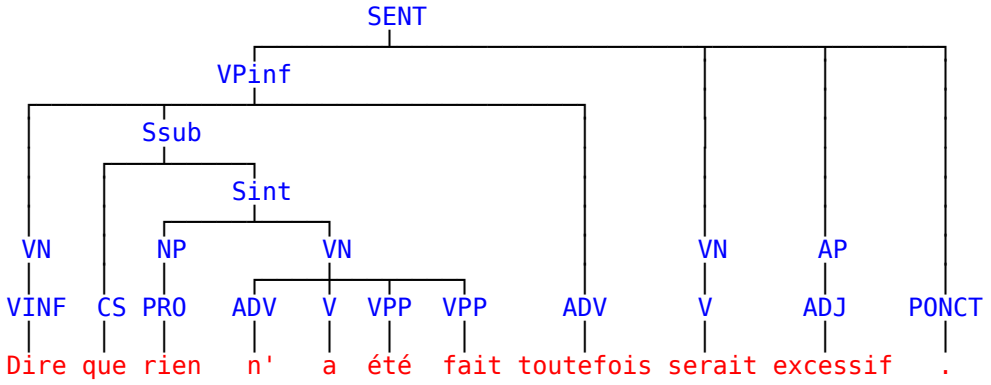
Bidirectional RNN In order to compute contextual representations for each token, a bidirectional RNN uses two RNN encoders (with distinct parameters, but with the same input): a left-to-right RNN computes successive forward states $\mathbf{f}_1, \ldots, \mathbf{f}_n$ and a right-to-left RNN computes successive backward states $\mathbf{b}_n, \ldots \mathbf{b}_1$ (Figure 4.4). The contextual representation for token i is the concatenation $\mathbf{c}_i = [\mathbf{f}_i; \mathbf{b}_i]$. A bi-RNN may also be used to represent a whole sequence instead of each element in the sequence, for example to compute a character-based representation of a word form (Ballesteros et al., 2015; Plank et al., 2016). In such a case, we use the concatenation of the last forward and backward states $[\mathbf{f}_n; \mathbf{b}_1]$ to represent the whole sequence.

4.5.3.2 Predicting Actions

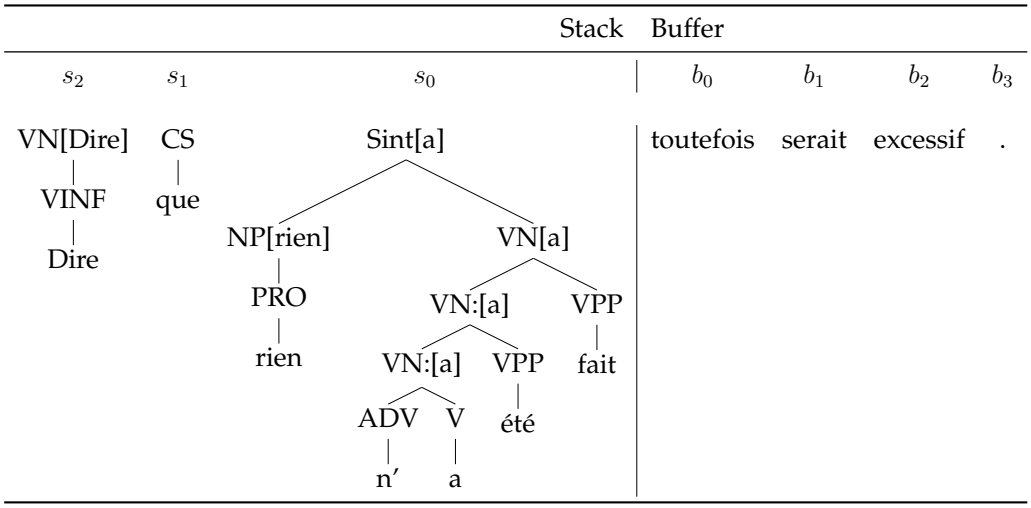
The computation of a distribution on actions for a given configuration is very similar to what is done with a feed-forward neural network (Section 4.5.2.2). Feature templates specify a list of typed positions in the configuration. These positions correspond either to a nonterminal or to a lexical element. The input to the action classifier $\mathbf{h}^{(0)}$ is the concatenation of nonterminal embeddings and of the contextual embeddings $\mathbf{c}_i$ corresponding to the lexical elements. We illustrate the instantiation of $\mathbf{h}^{(0)}$ for a full example in Figure 4.5, with the following template set:

$$\{s_0.c, s_1.c, s_0.w, s_0.w_l, s_0.w_r, s_1.w, s_1.w_l, s_1.w_r, s_2.w_r, b_0\}$$

In this template set, b_0 and $s_2.w_r$ are meant to represent, respectively the whole content of the buffer and the content of the stack beyond s_1. The other templates represent the right corner, left corner and head element of s_1 and s_0. This template set is rather small compared to those typically used with a feed-forward architecture (Chen and Manning,



(a) Tree from the French Treebank. *Saying that nothing has been done though would be excessive.*



(b) Example configuration from a derivation for the tree in 4.5a.

Template	Type	Symbol value	Vector value
$s_2.w_r$	lexical	Dire	$\mathbf{c}_1$
$s_1.c$	nonterminal	CS	$\mathbf{e}_{\text{CS}}$
$s_1.w$	lexical	que	$\mathbf{c}_2$
$s_1.w_r$	lexical	que	$\mathbf{c}_2$
$s_1.w_l$	lexical	que	$\mathbf{c}_2$
$s_0.c$	nonterminal	Sint	$\mathbf{e}_{\text{Sint}}$
$s_0.w$	lexical	a	$\mathbf{c}_5$
$s_0.w_l$	lexical	rien	$\mathbf{c}_3$
$s_0.w_r$	lexical	fait	$\mathbf{c}_7$
b_0	lexical	toutefois	$\mathbf{c}_8$

(c) A feature template set and its instantiation for the configuration in 4.5c. $\mathbf{e}_s$ is a simple embedding for symbol s. $\mathbf{c}_i$ is the vector representation of token i computed by the bi-LSTM.

Figure 4.5: Feature instantiation for bi-RNN parsing.

2014; Coavoux and Crabbé, 2016). Kiperwasser and Goldberg (2016) and Cross and Huang (2016a) showed that a minimal template set is generally informative enough to achieve state-of-the-art accuracies in, respectively, dependency parsing and projective constituency parsing, thanks to the quality of bi-RNN representations.

The data consists of a set of couples of sentences and derivations $\mathcal{D} = \{(x^{(i)}, a^{(i)})\}_{i=1}^{N}$. The objective function to train this architecture is the likelihood of full derivations:

$$L(\mathcal{D}; \boldsymbol{\theta}) = \sum_{i=1}^{N} - \log P(a^{(i)}_{1\ldots K_i} | x^{(i)}; \boldsymbol{\theta}) \tag{4.15}$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{K_i} - \log P(a^{(i)}_{j} | a^{(i)}_{1\ldots j-1}, x^{(i)}; \boldsymbol{\theta}) \tag{4.16}$$

4.6 Conclusion

In this chapter, we have presented an overview of decoding techniques and learning algorithms for transition-based constituency parsing. Transition-based parsers usually resort to approximate linear time search methods, such as beam search or greedy search. However, those algorithms are subject to locality biases. Under certain conditions on the scoring function, a parser can also rely on dynamic programming to explore a greater part of the search space, or even perform exact decoding in polynomial time.

After having presented the traditional global perceptron algorithm, we have discussed motivations for the recent shift towards neural methods. In particular, the use of symbol embeddings limits the data sparsity problem and leverages the similarity between symbols encoded in the embedding vector space. Finally, the use of Recurrent Neural Networks to construct global feature representations efficiently addresses the issue of locality biases of approximate search methods.

Part II

Projective Constituency Parsing

Chapter 5

Improving Learning and Searching with a Dynamic Oracle

Contents

5.1 Introduction **87**
5.2 Training a Parser with an Oracle **89**
5.2.1 Limitations of Static Oracles 89
5.2.2 Learning with Exploration 90
5.3 A Dynamic Oracle for Lexicalized Shift-Reduce Parsing . **92**
5.3.1 Preliminary Definitions 92
5.3.2 Cost Function 94
5.3.3 Finding 0-Cost Actions 97
5.3.4 An Oracle for SR-TMP 99
5.4 Experiments **102**
5.4.1 Experimental setting 102
5.4.2 Results 104
5.5 Conclusion **107**

5.1 Introduction

This chapter introduces an algorithm aimed at improving the training of greedy constituency parsers. Locally trained transition-based greedy

parsers are subject to inherent locality biases. Due to the locality of feature extraction, they must often make decisions before having access to the relevant information that may be arbitrarily far in the buffer. In other words, they run the risk of committing too early to a specific hypothesis.

Furthermore, early mistakes may have devastating consequences on the rest of an analysis. During training, the parser learns to predict a gold action from a gold configuration. At test time, due to error propagation, it must predict actions from noisy configurations and noisy features, but had not been prepared to do so, which intensifies error propagation.

There are different strategies to mitigate these effects. In principle, they are complementary, and their benefits might add up:

- **Better search**: abandon greedy decoding for beam search. By keeping several hypotheses in the beam, the parser avoids committing too early to a particular analysis. Yet, it is still possible for the correct hypothesis to fall out of the beam early.
- **Global features**: condition the probability of an action on the whole configuration (Cross and Huang, 2016b,a).
- **Non-monotonic transition systems**: allow the parser to cancel previous decisions in some cases. Such systems have been proposed for dependency parsing (Honnibal et al., 2013; Honnibal and Johnson, 2015; Fernández-González and Gómez-Rodríguez, 2017) but never applied to constituency parsing.
- **Learning to search**: instead of training the parser only on gold configurations, train it to predict the best action in any likely configuration.

In this chapter, we investigate how to apply the last strategy to constituency parsing. The learning-to-search approach has been initially developed to perform structured classification with a locally trained classifier (Daumé et al., 2009). Instead of seeing the training data as a set of gold configurations with corresponding gold actions, it considers the whole search space for each sentence. For a single sentence, the training data is generated on the fly by sampling likely configurations and training the parser to predict the best possible actions in these configurations. In order to use this training policy, it is necessary to have a dynamic oracle available, that is a function able to compute what the best action is in any configuration.

In dependency parsing, dynamic oracles have been proposed for most transition systems (Goldberg and Nivre, 2012, 2013; Goldberg et al., 2014;

Gómez-Rodríguez and Fernández-González, 2015) and shown to improve greedy parsing. In constituency parsing though, dynamic oracle training had not been proposed until recently (Coavoux and Crabbé, 2016; Cross and Huang, 2016a). Soon after the work presented in this chapter was published, Cross and Huang (2016a) introduced an unlexicalized transition system for constituency parsing that does not require binarization, as well as a dynamic oracle that is provably optimal for F1. In contrast, the dynamic oracle we introduce in this chapter is designed for a lexicalized shift-reduce transition. Though we could not prove optimality for our oracle in the general case, it brought similar absolute improvements ($\approx +0.3$) as that of Cross and Huang (2016a).

We first motivate the learning-to-search approach in greedy parsing (Section 5.2). Then, we introduce a dynamic oracle for transition-based constituency parsing (Section 5.3). Finally, we show that the proposed oracle improves parsing in a multilingual setting (Section 5.4).

5.2 Training a Parser with an Oracle

In this section, we motivate the use of a dynamic oracle when training a transition-based parser based on the lexicalized shift-reduce transition system presented in Section 3.2.1 and on a local statistical model such as the feed-forward network of Section 4.5.2.2.

5.2.1 Limitations of Static Oracles

The oracle is an important component of a transition-based parser. An oracle is a function that computes the best action given a parsing configuration and a gold tree. Most of the time, greedy parsers use a **static oracle**, that is an oracle that is only defined for gold configuration and that outputs a single gold action.

For some transition systems, a single tree may be derived by several distinct action sequences. In such cases, a static oracle favours only one of those sequences. The choice between those sequences may have a significant effect on parsing (Maier and Lichte, 2016), as some sequences may be easier to learn than others.

A limitation of static oracles is that the parser only sees gold configurations at training time, that is a tiny region of the search space. Learning consists in optimizing the likelihood of gold actions:

$$\sum_{i=1}^{N} -\log P(a^{(i)}|c^{(i)})$$

where the dataset $\mathcal{D} = \{(a^{(i)}, c^{(i)})\}_{i=1}^{N}$ is a set of couples of gold configurations and actions extracted statically from the treebank. At test time, the parser is in a completely different situation. Due to wrong decisions, it may end up in configurations very different from those it had seen during training. The situation is analogous to training a parser with gold POS tags as features and having only access to predicted tags at test time. Parsers are usually trained on data with predicted tags, to match the evaluation setting.

The motivation for training the parser on a greater part of the search space is the same: making the training setting as close as possible to the test setting. Instead of optimizing only the likelihood of gold actions, the parser optimizes the likelihood of best actions given any configuration,

$$\sum_{c \sim D} -\log P(o(c)|c)$$

where training examples are drawn from a distribution D of configurations and $o(c)$ is the set of best actions for configuration c. Ideally, D is the distribution of configurations the parser is likely to encounter at test time. In practice, D is approximated by sampling paths in the search space for sentences in the training set. Overall, by training the parser on a greater part of the search space, we hope to limit error propagation. We need two components to train a parser on likely configurations in the search space:

- An **exploration policy**, that is a way to sample likely configurations in the search space (Section 5.2.2).
- A **dynamic oracle**: a complete non-deterministic oracle (Goldberg and Nivre, 2012). A dynamic oracle determines the non-empty set of best actions for any possible configuration. We define a dynamic oracle in Section 5.3.

5.2.2 Learning with Exploration

An online trainer iterates several times over each sentence in the treebank, and updates its parameters until convergence. When a static oracle is used, the training examples can be pregenerated from the sentences.

When we use a dynamic oracle instead, we generate training examples on the fly, by following the prediction of the parser (given the current parameters) instead of the gold action, with probability p, where p is a hyperparameter that controls the degree of exploration. The online training algorithm for a single sentence s, with an oracle function o is shown in Algorithm 7. It is a slightly modified version of algorithm 3 from Goldberg and Nivre (2013), an approach they called *learning with exploration*.

In particular, since the neural network we used in our experiments is based on a log-likelihood loss, and not the perceptron loss used in Goldberg and Nivre (2013), updates are performed even when the prediction is correct. When $p = 0$, the algorithm acts identically to a static oracle trainer, as the parser always follows the gold transition. When the set of actions predicted by the oracle has more than one element, the best scoring element among them is chosen as the reference action to update the parameters of the neural network.[1]

Algorithm 7 Online training for a single annotated sentence s, using an oracle function o.

function TRAINONESENTENCE($s, \boldsymbol{\theta}, p, o$)
  $c \leftarrow$ INITIAL(s)
  while c is not a final configuration **do**
    $A \leftarrow o(c, s)$ ▷ set of best actions
    $\hat{a} \leftarrow \text{argmax}_a \, f_{\boldsymbol{\theta}}(c)_a$ ▷ prediction
    if $\hat{a} \in A$ **then**
      $t \leftarrow \hat{a}$ ▷ t: target
    else
      $t \leftarrow \text{argmax}_{a \in A} \, f_{\boldsymbol{\theta}}(c)_a$
    $\boldsymbol{\theta} \leftarrow$ UPDATE($\boldsymbol{\theta}, c, \hat{a}, t$) ▷ backpropagation
    if RANDOM() $< p$ **then**
      $c \leftarrow \hat{a}(c)$ ▷ follow prediction
    else
      $c \leftarrow t(c)$ ▷ follow best action
  return $\boldsymbol{\theta}$

[1] Another possibility would have been to reinforce every action in the set predict by o in such a case.

Stack: $S\|(C,l,i)\|(B,i,k)\|(A,k,j)$	
Action	Preconditions
REDUCE-LEFT/RIGHT-X:, X:$\in N_{\text{TMP}}$	C$\notin N_{tmp}$ or $j < n$
REDUCE-RIGHT-X	B$\notin N_{tmp}$
REDUCE-LEFT-X	A$\notin N_{tmp}$

Table 5.1: Constraints to ensure that binary trees can be unbinarized. The set N_{TMP} is the set of temporary symbols and n is the length of the sentence.

5.3 A Dynamic Oracle for Lexicalized Shift-Reduce Parsing

In this section, we introduce a dynamic oracle for lexicalized shift-reduce constituency parsing to instantiate the function o used in Algorithm 7.

To compute the best actions to perform in a given configuration, the oracle relies on a function $\mathcal{L}(a, c, T)$ that calculates the cost of applying action a in configuration c when the gold tree is T. The correctness of the oracle depends on the cost function (Goldberg and Nivre, 2013). A correct dynamic oracle o will have the following general formulation:

$$o(c, T) = \{a | \mathcal{L}(a, c, T) = \min_{a'} \mathcal{L}(a', c, T)\} \tag{5.1}$$

The correctness of the oracle is not necessary to improve training. In theory, the oracle only needs to be 'good enough' (Daumé et al., 2009), as confirmed in the context of dependency parsing by Straka et al. (2015).

The rest of this section is structured as follows. We first introduce some definitions and notations (Section 5.3.1). We go on to define a cost function (Section 5.3.2) and an algorithm to compute it (Section 5.3.3) and thus solve Equation 5.1.

5.3.1 Preliminary Definitions

In this section, we reformulate a lexicalized transition system to introduce notations that we will use to design the oracle (Section 5.3.1.1). Then, we define the notion of constituent decomposability (Section 5.3.1.2), a property of transition systems that is a sufficient condition for computing efficiently the cost of actions.

5.3.1.1 Transition System

We use the lexicalized transition system presented in Section 3.2.1. We reformulate it here to make explicit which constituents are constructed during a derivation. A configuration c is a triple $\langle S, j, \gamma \rangle$ where j is the index of the next token in the buffer, S is a stack of constituents, and γ is the set of constituents constructed so far in the derivation.

Constituents are defined as instantiated nonterminal symbols, i.e. triples (X, i, j) such that X is a nonterminal and (i, j) are integers denoting its span. Although in principle, the content of γ could be retrieved from the stack, we make it explicit to simplify the description of the oracle. The effects of the transition are defined as follows (we ignore lexicalization):

- SHIFT$(\langle S, j, \gamma \rangle) = \langle S|(w_j, j, j+1), j+1, \gamma \rangle$
- REDUCE-LEFT-X$(\langle S|(A, i, k)|(B, k, j), j, \gamma \rangle)$
  $= \langle S|(X, i, j), j, \gamma \cup \{(X, i, j)\} \rangle$
- REDUCE-RIGHT-X$(\langle S|(A, i, k)|(B, k, j), j, \gamma \rangle)$
  $= \langle S|(X, i, j), j, \gamma \cup \{(X, i, j)\} \rangle$
- REDUCE-UNARY-X$(\langle S|(A, k, j), j, \gamma \rangle)$
  $= \langle S|(X, k, j), j, \gamma \cup \{(X, k, j)\} \rangle$

We recall in Table 5.1 the preconditions on actions that make sure that the predicted tree is well-formed with respect to temporary symbols.

5.3.1.2 Constituent Decomposability

Goldberg and Nivre (2013) identified **arc-decomposability**, a powerful property of certain dependency parsing transition systems[2] for which we can easily derive correct efficient oracles. When this property holds, we can infer whether a tree is reachable from the reachability of individual arcs, simplifying the calculation of the cost of each transition. We rely on an analogue property we call **constituent decomposability**. A set of constituents is **tree-consistent** if it is a subset of a set corresponding to a well-formed tree. A phrase structure transition system is constituent-decomposable iff *for any configuration c and any tree-consistent set of constituents γ, if every constituent in γ is reachable from c, then the whole set is reachable from c* (constituent reachability will be formally defined in Section 5.3.2).

[2]The arc-eager, arc-hybrid, and easy-first transition systems are arc-decomposable, but the arc-standard system is not.

There are two main issues when defining a cost function for constituent parsing. The first is that the F-score measure, typically used to evaluate predicted trees, is not easily decomposable, in contrast to the Labelled and Unlabelled Attachment Score (LAS, UAS) used in dependency parsing. For that reason, we use a proxy metric based on constituent accuracy instead of F-measure. The second issue is related to temporary symbols, as they impose some constraints on the parser that make the calculation of the cost harder. Thus, we treat separately two cases: (i) an ideal case where we assume that there is no temporary symbols and (ii) the general case.

In the following paragraphs, we define a cost function (Section 5.3.2). Then, we introduce an algorithm to compute the cost of actions in an ideal case where we assume that there is no temporary symbols (Section 5.3.3) and finally in the general case by resorting to heuristics (Section 5.3.4).

5.3.2 Cost Function

The cost function we use ignores the lexicalization of the symbols, both for the sake of simplicity and because standard evaluation metrics are only based on non-lexicalized constituents. Hence, we assume that there is a single binary reduction type called REDUCE-X. We define the cost function in several steps. First of all, we define a cost function for a predicted tree with respect to a gold tree (Section 5.3.2.1). Then, we use it to define a cost function for individual actions (Section 5.3.2.2).

5.3.2.1 Cost for a Tree

We adopt a representation of trees as sets of constituents. For example, the tree in Figure 5.1, where nodes are annotated with their spans, is represented by the set:

{(S,0,7), (NP,0,3), (NP:,1,3), (VP,3,7), (VP:,3,5), (VP,5,7), (VP,6,7)}

The transition system constructs incrementally a set of constituents: every REDUCE action (unary or binary) adds a new constituent to γ. We define the cost of a predicted set of constituents $\hat{\gamma}$, with respect to a gold set γ^* as the number of constituents in γ^* which are not in $\hat{\gamma}$ penalized by the number of predicted unary constituents which are not in the gold set:

$$\mathcal{L}_r(\hat{\gamma}, \gamma^*) = |\gamma^* - \hat{\gamma}| + |\{(X, i, i+1) \in \hat{\gamma} \text{ st } (X, i, i+1) \notin \gamma^*\}| \quad (5.2)$$

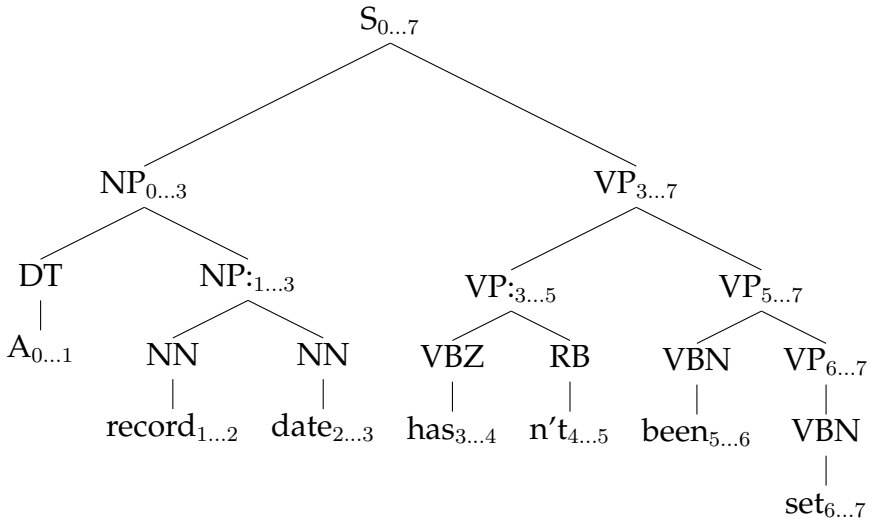


Figure 5.1: Binarized tree from the Penn Treebank annotated with spans (punctuation removed).

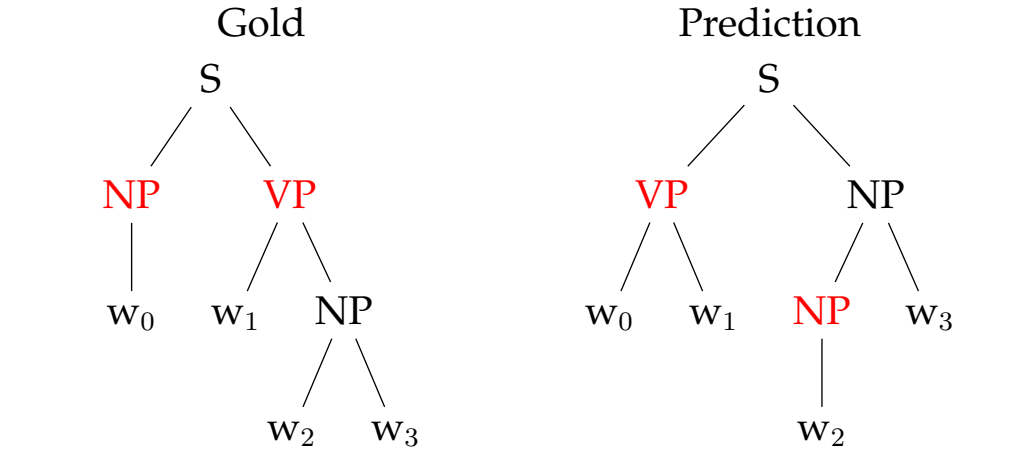
The first term penalizes false negatives and the second one penalizes unary false positives. Since the number of binary constituents in γ^* and $\hat{\gamma}$ depends only on the sentence length n, binary false positives are implicitly taken into account by the first term.

We illustrate the cost calculation for a tree in Table 5.2. In this example, the quantity $|\gamma^* - \hat{\gamma}|$ is 2 as the parser failed to predict (NP, 0, 1) and (VP, 1, 4), and $|\{(X, i, i+1) \in \hat{\gamma} \text{ st } (X, i, i+1) \notin \gamma^*\}|$ evaluates to 1, as the predicted unary constituent (NP, 2, 3) is a false positive. Therefore, the predicted set of constituents has cost 3.

5.3.2.2 Cost for a Transition

The cost function in Equation 5.2 is easily decomposable (as a sum of costs of transitions) whereas F1 measure is not.

The cost of a transition and that of a configuration are based on **constituent reachability**. The relation $c \vdash c'$ holds iff c' can be deduced from c by performing a transition. Let $\vdash^*$ denote the reflexive transitive closure of $\vdash$. A set of constituents γ (possibly a singleton) is reachable from a configuration c iff there is a configuration $c' = \langle S, j, \gamma' \rangle$ such that $c \vdash^* c'$ and $\gamma \subseteq \gamma'$, which we write $c \rightsquigarrow \gamma$.



$\gamma^* = \{$ (S, 0, 4), (NP, 0, 1), (VP, 1, 4), (NP, 2, 4) $\}$

$\hat{\gamma} = \{$ (S, 0, 4), (VP, 0, 2), (NP, 2, 3), (NP, 2, 4) $\}$

Table 5.2: Dynamic oracle cost calculation for a tree (artificial example).

The cost of an action a for a configuration c is the cost difference between the best tree reachable from $a(c)$ and the best tree reachable from c:

$$\mathcal{L}_r(a, c, \gamma^*) = \min_{\gamma : a(c) \rightsquigarrow \gamma} \mathcal{L}(\gamma, \gamma^*) - \min_{\gamma : c \rightsquigarrow \gamma} \mathcal{L}(\gamma, \gamma^*)$$

By definition, for each configuration, there is at least one transition with cost 0 with respect to the gold parse. Otherwise, it would entail that there is a tree reachable from c but unreachable from $a(c)$, for any a. Therefore, we reformulate equation 5.1:

$$o(c, \gamma^*) = \{a | \mathcal{L}_r(a, c, \gamma^*) = 0\} \tag{5.3}$$

Actions with a cost of 0 are actions that maintain the reachability of at least one of the best trees. We illustrate the computation of the cost of transitions in Table 5.3. After two SHIFTS, there are three gold reachable constituents. Performing another SHIFT maintains reachability for the three constituents, whereas performing a reduction with the two tokens in the stack jeopardizes the reachability of (VP, 1, 4), because this constituent is not compatible with (X, 0, 2).

In transition systems, the grammar is left implicit. A reduction of two nodes X and Y to Z may be performed even though the grammar rule Z $\rightarrow$ X Y does not appear in the treebank. Nevertheless, there are constraints related to temporary symbols. These constraints make it difficult to test the

Configuration:	$\langle (w_0, 0, 1) \vert (w_1, 1, 2), 2, \emptyset \rangle$	
	Gold reachable set	Action cost
Now	{ (S, 0, 4), (NP, 2, 4), (VP, 1, 4) }	
After SHIFT	{ (S, 0, 4), (NP, 2, 4), (VP, 1, 4) }	0
After REDUCE-X (any X)	{ (S, 0, 4), (NP, 2, 4) } ~~(VP, 1, 4)~~	1

Table 5.3: Cost of transitions from a given configuration, with respect to the gold tree in Figure 5.2 (left-hand side).

reachability of constituents. For this reason, we instantiate two transition systems. We call SR-TMP the transition system formulated above that enforces the constraints in Table 5.1, and SR-BIN, the same transition system without any of such constraints. SR-BIN assumes an idealized case where the grammar contains no temporary symbols, whereas SR-TMP is the actual system we use in our experiments.

In Section 5.3.3, we derive a correct dynamic oracle for SR-BIN. In Section 5.3.4, we present heuristics to use the oracle with SR-TMP.

5.3.3 Finding 0-Cost Actions

SR-BIN transition system provides no guarantees that predicted trees are unbinarizable. The only condition for a binary reduction to be allowed is that the stack contains at least two symbols. If so, any nonterminal in the grammar could be used. In such a case, we can define a simple necessary and sufficient condition for constituent reachability, which makes it very easy to find 0-cost actions.

5.3.3.1 Constituent Reachability

Let γ^* be a tree-consistent constituent set, and $c = \langle S, j, \gamma \rangle$ a parsing configuration, such that:

$$S = (X_1, i_0, i_1) \dots (X_p, i_{p-1}, i) \vert (A, i, k) \vert (B, k, j)$$

A binary constituent (X, m, n) is reachable iff it satisfies one of the following three conditions:

1. $(X, m, n) \in \gamma$
2. $j < m < n$

3. $m \in \{i_0, \dots i_{p-1}, i, k\}$, $n \geq j$ and $(m, n) \neq (k, j)$

The first two cases are trivial and correspond respectively to a constituent already constructed and to a constituent spanning words which are still in the buffer.

In the third case, (X, m, n) can be constructed by performing $n-j$ times the transitions SHIFT (possibly with some in-between REDUCE-UNARY), and then a sequence of binary reductions ended by a reduction to X. As the index j in the configuration is non-decreasing during a derivation, the constituents whose span end is smaller than j are not reachable if they are not already constructed. For a unary constituent, the condition for reachability is straightforward: a constituent $(X, l-1, l)$ is reachable from configuration $c = \langle S, j, \gamma \rangle$ if one of these conditions holds:

1. $(X, l-1, l) \in \gamma$
2. $l > j$
3. $l = j$ and $c = \text{SHIFT}(c')$ (the last action is SHIFT)

Algorithm 8 Oracle algorithm for SR-BIN.

```
function ORACLE(c = ⟨S|(A,i,k)|(B,k,j), j, γ⟩, γ*)
    if c = SHIFT(c') then                          ▷ Last action was SHIFT
        if (X, j−1, j) ∈ γ* then
            return {REDUCE-UNARY-X}
    if ∃n > j, (X, k, n) ∈ γ* then
        return {SHIFT}
    if (X, i, j) ∈ γ* then
        return {REDUCE-X}
    if ∃m < i, (X, m, j) ∈ γ* and (X, m, j) is reachable then
        return {REDUCE-Y, ∀Y}
    return {a|a is a possible action}
```

5.3.3.2 Constituent Decomposability

SR-BIN is constituent decomposable. In this paragraph, we show why this holds. Reasoning by contradiction, let us assume that every constituent of a tree-consistent set γ^* is reachable from $c = \langle S|(A, i, k)|(B, k, j), j, \gamma \rangle$ and that γ^* is not reachable (contraposition). This entails that at some

point during a derivation, there is no possible transition which maintains reachability for all constituents of γ^*. Let us assume c is in such a case. If some constituent of γ^* is reachable from c, but not from SHIFT(c), its span must have the form (m, j), where $m \leq i$. If some constituent of γ^* is reachable from c, but not from REDUCE-X(c), for any label X, its span must have the form (k, n), where $n > j$. If both conditions hold, γ^* contains incompatible constituents (crossing brackets), which contradicts the assumption that γ^* is tree-consistent.

5.3.3.3 Computing the Cost of a Transition

The conditions on constituent reachability makes it easy to compute the cost of a transition a for a given configuration $c = \langle S|(A, i, k)|(B, k, j), j, \gamma\rangle$ and a gold set γ^*:

- The cost of a SHIFT is the number of constituents not in γ, reachable from c and whose span ends in j.
- The cost of a binary reduction REDUCE-X is a sum of two terms. The first one is the number of constituents of γ^* whose span has the form (k, n) with $n > j$. These are no longer compatible with (X, i, j) in a tree. The second one is one if $(Y, i, j) \in \gamma^*$ and $Y \neq X$, and zero otherwise. It is the cost of mislabelling a constituent with a gold span.
- The cost of a REDUCE-UNARY-X is zero if $(X, j - 1, j) \in \gamma^*$ and one otherwise.

Algorithm 8 is derived from these observations.

5.3.4 An Oracle for SR-TMP

We now turn to the SR-TMP system. When there are constraints on the REDUCE actions (Table 5.1), the conditions for constituent reachability for SR-BIN do not hold any longer. The consequence is that Algorithm 8 is not correct for SR-TMP.

In Figure 5.2, we give an illustration of a prototypical case in which Algorithm 8 will fail. The constituent (C:, i, j) is in the gold set of constituents and could be constructed with REDUCE-C:. Since the third symbol in the stack is the temporary symbol D:, the reduction to C:, another temporary symbol, will jeopardize the reachability of (C, m, j) because reductions are not possible when the two symbols at the top of the stack are temporary

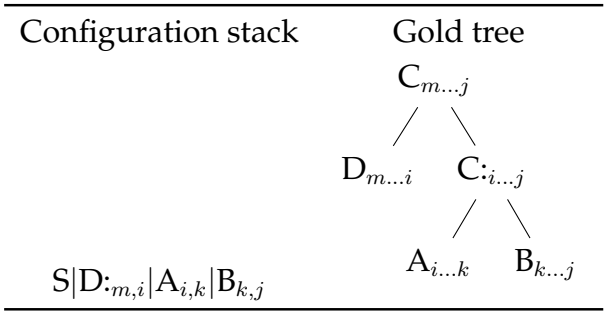


Figure 5.2: Difficult case for SR-TMP. Due to the temporary symbol constraints enforced by SR-TMP, the algorithm in Figure 8 will fail on this example.

symbols (Table 5.1). The best course of action is a reduction to any non-temporary symbol, so as to keep (C, m, j) reachable. Note that in this case, the cost of REDUCE-C: cannot be smaller than that of a single mislabelled constituent. Labelling errors are local, whereas structural errors may have global consequences.

In fact, this example shows that the constraints inherent to SR-TMP makes it non constituent-decomposable. In the example in Figure 5.2, both constituents in the set $\{(\mathrm{C}, m, j), (\mathrm{C}:, i, j)\}$, a tree-consistent constituent set, are reachable. However, the whole set is not reachable, as REDUCE-C: would make (C, m, j) unreachable.

In dependency parsing, several exact dynamic oracles have been proposed for non arc-decomposable transition systems (Goldberg et al., 2014), including systems for non-projective parsing (Gómez-Rodríguez et al., 2014). These oracles rely on tabular methods to compute the cost of transitions and have (high-degree) polynomial worst case running time. Instead, to avoid resorting to more computationally expensive exact methods, we adapt Algorithm 8 to the constraints involving temporary symbols using the following heuristics:

- **Prefer correct structure to correct labelling**. If the standard oracle predicts a reduction, make sure to choose its label so that every reachable constituent $(X, m, j) \in \gamma^*$ $(m < i)$ is still reachable after the transition. Practically, if such a constituent exists and if the third symbol on the stack is a temporary symbol, then do not predict a temporary symbol.

- **Avoid false positives**. When reductions to both temporary symbols and non-temporary symbols have cost zero, only predict temporary

symbols. A non-temporary symbol corresponds to a constituent in the unbinarized tree, whereas a temporary symbol does not. Therefore, predicting a non-temporary symbol would decrease precision on n-ary constituents.

Algorithm 9 is the resulting oracle based on these heuristics. It is the oracle we use in our experiments.

Head Choice In some cases, namely when reducing two non-temporary symbols to a new constituent (X, i, j), the oracle must determine the head position in the reduction (REDUCE-RIGHT or REDUCE-LEFT). We use the following heuristic: if (X, i, j) is in the gold set, choose the same head position; otherwise, predict both REDUCE-RIGHT-X and REDUCE-LEFT-X to keep the non-determinism.

Algorithm 9 Dynamic oracle algorithm (SR-TMP).

function ORACLE-TMP($c = \langle S|(C, l, i)|(A, i, k)|(B, k, j), j, \gamma\rangle$, γ^*)
  if $c = \text{SHIFT}(c')$ **then** ▷ Last action was SHIFT
    if $(X, j-1, j) \in \gamma^*$ **then**
      return {REDUCE-UNARY-X}
  if $\exists n > j, (X, k, n) \in \gamma^*$ **then**
    return {SHIFT}
  if A and B are temporary symbols **then**
    return {SHIFT}
  if $(X, i, j) \in \gamma^*$, X non-temporary **then**
    return {REDUCE-X}
  if $(X, i, j) \in \gamma^*$, X temporary **then**
    if C is temporary and $\exists m < i, (Y, m, j) \in \gamma^*$ is reachable **then**
      return {REDUCE-Z| Z non temporary }
    return {REDUCE-X}
  if $\exists m < i, (X, m, j) \in \gamma^*$ and (X, m, j) is reachable **then**
    if C is temporary **then**
      return {REDUCE-Z| Z non temporary }
    return {REDUCE-Y|Y temporary }
  return {$a|a$ is a possible action}

5.4 Experiments

The experiments we conducted aimed at assessing whether the dynamic oracle training improved parsing. We compare two experimental settings. In the 'static' setting, the parser is trained only on gold configurations; in the 'dynamic' setting, we use the dynamic oracle and the training method in Algorithm 7 to explore non-gold configurations.

Before discussing the results (Section 5.4.2) we briefly present our experimental protocol (Section 5.4.1).

5.4.1 Experimental setting

5.4.1.1 Datasets

We used both the SPMRL dataset (Seddah et al., 2013) and the Penn Treebank (Marcus et al., 1993) to assess the effect of the dynamic oracle over a static oracle, and to compare the resulting parser with other published results. The SPMRL dataset contains constituency treebanks for nine languages: Arabic, Basque, French, German, Hebrew, Hungarian, Korean, Polish and Swedish. In these treebanks, each token is annotated with a language-specific number of morphological attributes, such as case, number, tense, mood, in addition to standard POS tags. These attributes have been shown to be very important for parsing (Björkelund et al., 2013; Crabbé, 2015). Languages with a rich inflectional morphology are more subject to lexical data sparsity, because a single lexeme may usually appear with lots of different word forms in the corpus. Since the corpora typically exhibit high out-of-vocabulary rates, morphological attributes provide important information.

The POS tags and morphological attributes were predicted using MarMoT (Mueller et al., 2013), with 10-fold jackknifing for the training corpora. For the SPMRL dataset, the head annotation was carried out with the procedures described in Crabbé (2015), using the alignment between dependency treebanks and constituency treebanks. For English, we used Collins' head annotation rules (Collins, 1999).

5.4.1.2 Classifier: Feed-Forward Neural Network

Feature Templates We used the feed-forward network described in Section 4.5.2 with a single hidden layer. The input to the network is a concatenation of embeddings of symbols extracted from the configuration.

Nonterminals						
$s_0.c_t$	$s_0.c_l$	$s_0.c_r$	$s_1.c_t$	$s_1.c_l$	$s_1.c_r$	$s_2.c_t$
Tokens						
$s_0.w_t.form$	$s_0.w_l.form$	$s_0.w_r.form$	$s_1.w_t.form$	$s_1.w_l.form$	$s_1.w_r.form$	$s_2.w_t.form$
$s_0.w_t.tag$	$s_0.w_l.tag$	$s_0.w_r.tag$	$s_1.w_t.tag$	$s_1.w_l.tag$	$s_1.w_r.tag$	$s_2.w_t.tag$
$s_0.w_t.m \forall m \in M$			$s_1.w_t.m \forall m \in M$			
$b_0.tag$	$b_1.tag$	$b_2.tag$	$b_3.tag$			
$b_0.form$	$b_1.form$	$b_2.form$	$b_3.form$			
$b_0.m \forall m \in M$		$b_1.m \forall m \in M$				

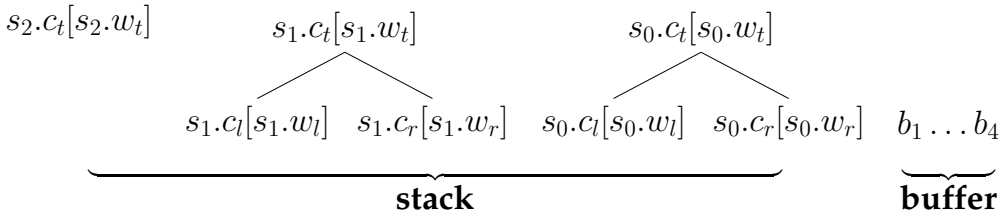


Table 5.4: Specification of the input to the neural network as feature templates (upper part). Schematic representation of a configuration (lower part).

The complete list of templates is presented in Table 5.4. Each type of symbol has its own embedding matrix. The types include nonterminals, word forms, POS tags, and any of the available morphological attributes (represented by the set M in the table).

Every embedding was initialized randomly (uniformly) in the interval $[-0.01, 0.01]$. Word embeddings have 32 dimensions, tags and nonterminal embeddings have 16 dimensions. The dimensions of the morphological attributes depend on the number of values they can have (Table 5.5). When a feature template addresses an empty position in a configuration, for example $b_0.form$ when the buffer is empty, it is valued by a special vector whose coefficients are learned parameters. Finally, the hidden layer has 512 units.

Training Recall that a hyperparameter p controls the probability to follow the parser's prediction when training on a sentence, instead of following the best action. The value of p may have a significant effect on the training of the parser: if p is too high, the gradient descent may converge too slowly. If p is too low, the training accuracy can be very high in the first iterations, which prevents the exploration of the search space. In order to avoid these two extreme situations, for the 'dynamic' setting, we trained

Number of possible values	≤ 8	≤ 32	> 32
Dimensions for embedding	4	8	16

Table 5.5: Size of morphological attributes embeddings.

'static' and 'dynamic' setting			'dynamic' setting	
learning rate	α	iterations	k	p
$\{0.01, 0.02\}$	$\{0, 10^{-6}\}$	$[1, 24]$	$\{8, 16\}$	$\{0.5, 0.9\}$

Table 5.6: Hyperparameters. α is the decrease constant used for the learning rate (Bottou, 2010).

every other k sentence with the dynamic oracle and the other sentences with the static oracle. This method, used by Straka et al. (2015), allows for high values of p, without slowing or preventing convergence. It is also more stable than training only with the dynamic oracle.

We used several hyperparameters combinations (see Table 5.6). For each language, we present the model with the combination which maximizes the development set F-score. We used the Averaged Stochastic Gradient Descent (Polyak and Juditsky, 1992) algorithm to minimize the negative log likelihood of the training examples. We shuffled the sentences in the training set before each iteration.

5.4.2 Results

In this section, we discuss the effect of the dynamic oracle (Section 5.4.2.1), and the combined effect of beam search and dynamic oracle training (Section 5.4.2.2).

5.4.2.1 Effect of the Dynamic Oracle

Results for English are shown in Table 5.7. The use of the dynamic oracle improves the F-score by 0.4 on the development set and 0.6 on the test set. The resulting parser, despite using greedy decoding and no additional data, is fairly accurate. For example, it compares well with the span-based model of Hall et al. (2014), and is much faster. Current state-of-the-art methods (Cross and Huang, 2016a; Liu and Zhang, 2017a) are based on bi-

Dev F1 (EVALB)		Method	Decoding	tokens/sec
Static (this work)	88.6	transition, neural	greedy	
Dynamic (this work)	89.0	transition, neural	greedy	
Test F1 (EVALB)				
Hall et al. (2014)	89.2	chart, CRF	CKY	12
Petrov et al. (2006)	90.1	chart, PCFG-LA	CKY	169
Durrett and Klein (2015)†	91.1	chart, neural CRF	CKY	-
Zhu et al. (2013)†	91.3	transition, perceptron	beam=16	1,290
Crabbé (2015)	90.0	transition, perceptron	beam=8	2,150
Wang et al. (2015)†	89.4	transition, neural	beam=8	-
Sagae and Lavie (2006)	85.1	transition, MaxEnt	greedy	-
Cross and Huang (2016a)	91.3	transition, bi-LSTM	greedy	-
Liu and Zhang (2017a)	91.8	transition, bi-LSTM	greedy	-
Static (this work)	88.0	transition, neural	greedy	3,820
Dynamic (this work)	88.6	transition, neural	greedy	3,950

Table 5.7: Results on the Penn Treebank (Marcus et al., 1993). † Uses clusters or word vectors learned on unannotated data.

LSTM models that compute global features, mitigating the locality biases of greedy search.

For the SPMRL dataset, we report results on the development sets and test sets in Table 5.8. The metrics take punctuation and unparsed sentences into account (Seddah et al., 2013). We compare our results with the SPMRL shared task baselines (Seddah et al., 2013) and several other parsing models. The model of Björkelund et al. (2014) is based on a product of PCFG-LA grammars and a discriminative reranker, together with morphological features and word clusters learned on unannotated data. Durrett and Klein (2015) use a neural CRF based on CKY decoding algorithm, with word embeddings pretrained on unannotated data. Fernández-González and Martins (2015) use a parsing-as-reduction approach, based on a dependency parser with a label set rich enough to reconstruct constituent trees from dependency trees. Finally, Crabbé (2015) uses a structured perceptron with rich morphological features and beam search decoding. Both Crabbé (2015) and Björkelund et al. (2014) use MARMOT-predicted morphological tags (Mueller et al., 2013), as is done in our experiments.

Our results show that, despite using a very simple greedy inference and being strictly supervised, our base model (static oracle training) is competitive with other single parsers on this dataset (excluding the recent bi-LSTM parsers).

		Arabic	Basque	French	German	Hebrew	Hungarian	Korean	Polish	Swedish	Avg
	Decoding	Development F1 (EVALBSPMRL)									
Durrett and Klein (2015)†	CKY	80.68	84.37	80.65	**85.25**	89.37	**89.46**	82.35	92.10	**77.93**	84.68
Crabbé (2015)	beam=8	**81.25**	84.01	**80.87**	84.08	**90.69**	88.27	83.09	**92.78**	77.87	84.77
Static (this work)	greedy	80.25	84.29	79.87	83.99	89.78	88.44	84.98	92.38	76.63	84.51
Dynamic (this work)	greedy	80.94	**85.17**	80.31	84.61	90.20	88.70	**85.46**	92.57	77.87	**85.09**
		Test F1 (EVALBSPMRL)									
Björkelund et al. (2014)†		81.32*	**88.24**	**82.53**	**81.66**	**89.80**	**91.72**	83.81	90.50	**85.50**	**86.12**
Berkeley (Petrov et al., 2006)	CKY	79.19	70.50	80.38	78.30	86.96	81.62	71.42	79.23	79.18	78.53
Berkeley-Tags	CKY	78.66	74.74	79.76	78.28	85.42	85.22	78.56	86.75	80.64	80.89
Durrett and Klein (2015)†	CKY	80.24	85.41	**81.25**	**80.95**	88.61	**90.66**	82.23	**92.97**	**83.45**	**85.09**
Crabbé (2015)	beam=8	**81.31**	84.94	80.84	79.26	**89.65**	90.14	82.65	92.66	83.24	84.97
Fernández-González and Martins (2015)		-	85.90	78.75	78.66	88.97	88.16	79.28	91.20	82.80	(84.22)
Static (this work)	greedy	79.77	85.91	79.62	79.20	88.64	90.54	84.53	92.69	81.45	84.71
Dynamic (this work)	greedy	80.71	**86.24**	79.91	80.15	88.69	90.51	**85.10**	**92.96**	81.74	**85.11**
Dynamic (this work)	beam=2	81.14	86.45	80.32	80.68	89.06	90.74	85.17	**93.15**	82.65	85.48
Dynamic (this work)	beam=4	81.59	86.45	80.48	80.69	89.18	90.73	85.31	93.13	82.77	85.59
Dynamic (this work)	beam=8	**81.80**	**86.48**	80.56	80.74	89.24	**90.76**	**85.33**	93.13	82.80	**85.64**

Table 5.8: Results on development and test corpora of the SPMRL dataset. Metrics are provided by `evalb_spmrl` with `spmrl.prm` parameters (`http://www.spmrl.org/spmrl2013-sharedtask.html`).
†Use clusters or word vectors learned on unannotated data.
*Björkelund et al. (2013).

Furthermore, we observe that the dynamic oracle improves training by up to 0.6 F-score (averaged over all languages). The improvement depends on the language. For example, Swedish, Arabic, Basque and German are the languages with the most important improvement. In terms of absolute score, the parser also achieves very good results on Korean and Basque, and even outperforms the reranker of Björkelund et al. (2014) on Korean.

5.4.2.2 Combined Effect of Beam Search and Dynamic Oracle

Although initially, dynamic oracle training was designed to improve parsing without relying on more complex search methods (Goldberg and Nivre, 2012), we tested the combined effects of dynamic oracle training and beam search decoding. In Table 5.8, we also provide results with beam search decoding with the models trained in the 'dynamic' setting. The transition from greedy search to a beam of size two brings an improvement comparable to that of the dynamic oracle. Further increase in beam size does not seem to have any noticeable effect, except for Arabic. These results show that the effects of dynamic oracle training and beam decoding are complementary and suggest that a good tradeoff between speed and accuracy is already achieved in a greedy setting or with a very small beam size.

5.5 Conclusion

In this chapter, we have introduced a dynamic oracle algorithm for lexicalized constituency parsing. We used the dynamic oracle to train a greedy parser to learn how to search by letting it explore the search space. Experiments have shown that the dynamic oracle training improves parsing over a static oracle consistently for almost every dataset we used.

Chapter 6

Modelling Morphological and Functional Interfaces

Contents

6.1 Introduction **108**
6.2 Multitask Learning **110**
6.3 A Multitask Learning Architecture for Parsing and Tagging **113**
6.3.1 A Shared Hierarchical Bidirectional LSTM Encoder . 114
6.3.2 Tagging Component 115
6.3.3 Parsing Component 115
6.3.4 Objective Function and Training 116
6.4 Experiments **117**
6.4.1 Datasets 118
6.4.2 Protocol 118
6.4.3 Results and Discussion 120
6.5 Conclusion **123**

6.1 Introduction

This chapter addresses the issue of the interfaces of phrase structure parsing. We focus on two interfaces: inflectional morphology and functional structure.

Morphological information is very important for constituency parsing. In the case of languages with a rich inflectional morphology, tagsets are often very large, and structured into a coarse part-of-speech tag and a set of morphological attributes (such as tense, mood, case). For the SPMRL dataset for example, there are from 7 morphological attributes (German) to 24 (Basque). Adding this information as input to a parser is very beneficial to parsing (Björkelund et al., 2013; Crabbé, 2015).

The most common approach to use morphological information in parsing is the pipeline approach: the output of a morphological tagger is used by the parser as input features (Crabbé, 2015; Björkelund et al., 2013). This method works well in practice. For example, Crabbé (2015) reports an absolute improvement of 2.2 % F1 in average on the SPMRL dataset, thanks to the use of morphological features.

Yet, the pipeline approach has limitations. First, resorting to an external tagger can be costly. Secondly, the pipeline is prone to error propagation. Although it is possible to keep some ambiguity by providing the k-best analyses from the external tagger to the parser (Bohnet et al., 2013; Mi and Huang, 2015), most works use only the best single morphological analysis, preventing interaction between structural information and morphological information. Finally, designing feature templates when a lot of morphological attributes are available is a difficult problem, given the number of possible combinations (Crabbé, 2015).[1] In this chapter, we address jointly morphological analysis and parsing. In order to construct representations suitable both for parsing and tagging, we incorporate character-based embeddings in the architecture, as was done in dependency parsing (Ballesteros et al., 2015) and POS tagging (Santos and Zadrozny, 2014; Labeau et al., 2015; Plank et al., 2016).

The grammatical functions of the constituents (e.g. subject, object) are rarely predicted by constituency parsers, and ignored by the standard evaluator (EVALB), in contrast with dependency parsing where functional labels are part of the evaluation. However, this information is generally available in treebanks and might be useful both for parsing and its applications. For example, subject and object NPs do not have the same properties nor the same distributions (in English, subjects NP are more likely to be shorter and to rewrite as a pronoun). Such linguistic knowledge has been used successfully with annotated PCFG (Klein and Manning, 2003; Petrov et al., 2006). On the other hand, the prediction of functional tags is important for, e.g. extracting a predicate argument semantic structure from the syntactic tree. Depending on the annotation guidelines, the fact that

[1] At least for linear classifiers, neural networks being less subject to this issue.

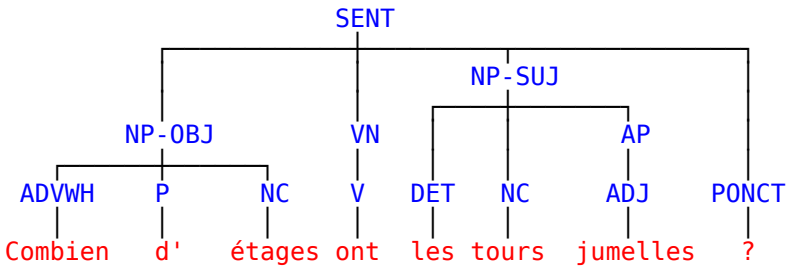


Figure 6.1: Tree from the French Question Bank (Seddah and Candito, 2016). The French Question Bank follows the same annotation guidelines as the French Treebank. *How many floors do the twin towers have?*

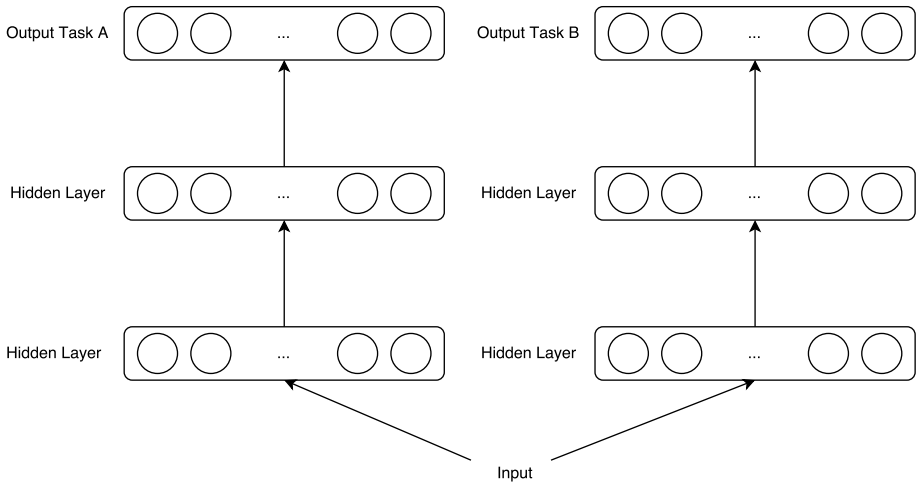
an NP is a subject or an object cannot be inferred from the tree structure. In the French Treebank annotation style, for example, subject and object NPs are both sibling nodes of the verbal nucleus, as illustrated in the tree in Figure 6.1. Seddah and Candito (2016) noticed that phrase structure is rather easy to recover on the French Question Bank, but that functional labels are hard to predict, due to word order variations. Depending on whether they took into account functional labels during evaluation, they observed an 18 F1 difference. Thus, functional labelling is challenging, in particular in sentences with non-canonical word order.

In this chapter, we introduce a neural network architecture based on multitask learning to model interactions between constituency syntax and two of its important interfaces: morphology and functional labelling. Our objectives are two-fold. At training time, we use morphological and functional labels as an additional training signal to improve constituency parsing. At test time, our model is also able to output a morphological analysis and a functional label for each token, thus providing a more complete syntactic analysis than a single parser.

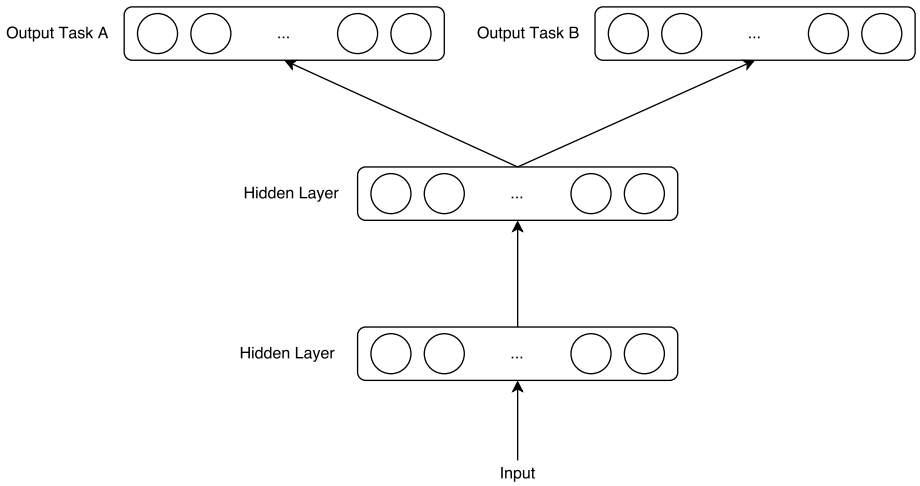
We first introduce the framework of Multitask Learning (MTL, Section 6.2). In Section 6.3 we present an MTL neural architecture for joint parsing and tagging. In Section 6.4, we evaluate our model on the SPMRL dataset.

6.2 Multitask Learning

Multitask learning is a transfer learning method introduced by Caruana (1997) that aims at improving the generalization ability of a classifier for



(a) Single-task learning.



(b) Multitask learning.

Figure 6.2: Single-task learning vs multitask learning.

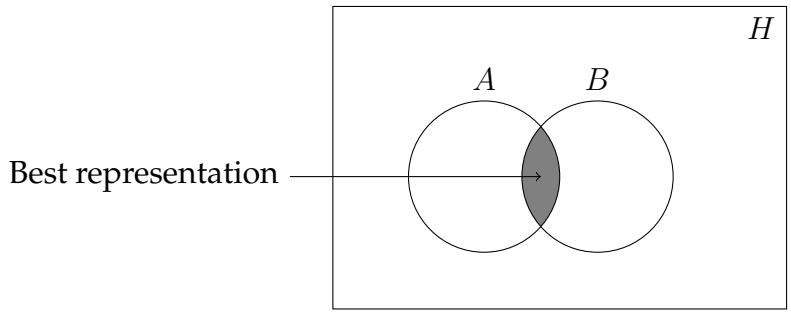


Figure 6.3: Representations learned by multitask learning (Caruana, 1997). H is the representation search space.

a target task by using an additional training signal from a related task, sometimes called auxiliary task.

When used with neural network classifiers,[2] multitask learning usually works by learning, often in parallel, several tasks with shared representations (Figure 6.2). In the case of single task learning (6.2a), the two tasks with the same input are completely distinct. Instead, in a multitask learning setup, the two tasks share hidden layers as well as the parameters to compute them from the inputs (6.2b).

Consider a main task A and a related auxiliary task B, with their respective loss functions L_A and L_B, sets of parameters $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$, and datasets D_A and D_B. In a single task setting, $\boldsymbol{\theta}_A \cap \boldsymbol{\theta}_B = \emptyset$, and the two loss functions are optimized separately. In contrast, in a multitask setting, $\boldsymbol{\theta}_A \cap \boldsymbol{\theta}_B \neq \emptyset$, and the objective function L jointly optimizes task A and B:

$$L(D_A, D_B; \boldsymbol{\theta}) = \alpha \cdot L_A(D_A; \boldsymbol{\theta}_A) + \beta \cdot L_B(D_B; \boldsymbol{\theta}_B) \tag{6.1}$$

where α and β are hyperparameters. Usually, the trainer samples a task at each stochastic training step to optimize either L_A or L_B.

Parameter sharing works as an inductive bias: in order to fit task A, the network will prefer a hypothesis that learns a good representation for task B. Improvements in generalization may come from different effects (Caruana, 1997):

- **Regularization**: the term L_B added to the objective function prevents overfitting to task A by constraining the set of hypotheses.
- **Data amplification**: the data from task B might contain information useful for task A not available in the dataset for task A.

[2]Multitask learning can also be used with other types of classifiers, e.g. linear classifiers. See Ruder (2017) for an overview.

token	morphology							function
	POS tag	mood	number	person	tense	subcat	gender	
Combien	ADVWH	NA	NA	NA	NA	int	NA	mod
d'	P	NA	NA	NA	NA	NA	NA	det
étages	NC	NA	p	NA	NA	c	m	obj
ont	V	ind	p	3	pst	NA	NA	root
les	DET	NA	p	NA	NA	def	NA	det
tours	NC	NA	p	NA	NA	c	f	suj
jumelles	ADJ	NA	p	NA	NA	qual	f	mod
?	PONCT	NA	NA	NA	NA	s	NA	ponct

Table 6.1: Word label matrix for a full sentence.

- **Eavesdropping**: a feature useful for task A but hard to learn, might be easy to learn when learning task B. In such cases, task A eavesdrops on the feature representation learned by task B in the shared layers.
- **Representation bias**: The optimizer will learn representations that are at the intersection of good representations for task A and good representations for task B (Figure 6.3). The auxiliary tasks add a bias in the search for suitable representations.

6.3 A Multitask Learning Architecture for Parsing and Tagging

In this section, we introduce a parser based on the standard lexicalized transition system already described in Chapter 3. We focus on the formulation of a scoring function which models jointly parsing, full morphological analysis and functional labelling at the word level.

The architecture is motivated by the fact that the sentence bi-LSTM of parsers (Cross and Huang, 2016b) is very similar to a tagging architecture (Plank et al., 2016). The full architecture is illustrated in Figure 6.4. In what follows, we assume that, at training time, each token w_i is associated with a vector of typed symbols $M_i = (M_{i,1}, M_{i,2}, \ldots, M_{i,m})$. The word-level labels for a single sentence form a matrix $\mathbf{M} = [M_1; \ldots; M_n]$, as illustrated in Table 6.1.

The objective of our architecture is to model both

$$p(\mathbf{M} = (M_1, M_2, \ldots, M_m) | w_1^n; \boldsymbol{\theta}_t),$$

the probability of every label for every token in the sentence and

$$p(T|w_1^n; \boldsymbol{\theta}_p),$$

the probability of a syntactic tree T conditioned on the whole sentence. The first term decomposes as

$$p(\mathbf{M}|w_1^n; \boldsymbol{\theta}_t) = \prod_{i=1}^{n} \prod_{j=1}^{m} p(M_{i,j}|w_1^n; \boldsymbol{\theta}_t)$$

under the assumption that labels are independent. The second term is defined as the probability of the sequence of actions $\mathbf{a} = (a_1, a_2, \dots a_k)$ used to construct T:

$$\begin{aligned} p(T|w_1^n; \boldsymbol{\theta}_p) &= p(a_1^k|w_1^n; \boldsymbol{\theta}_p) \\ &= \prod_{i=1}^{k} p(a_i|a_1^{i-1}, w_1^n; \boldsymbol{\theta}_p) \end{aligned}$$

The two models are based on a common representation computed with shared parameters $\boldsymbol{\theta}_s = \boldsymbol{\theta}_p \cap \boldsymbol{\theta}_t \neq \emptyset$. We first introduce a hierarchical bi-LSTM model used to build a common representation with parameters $\boldsymbol{\theta}_s$, then we present the models to compute respectively the distributions $p(M_{i,j} = \cdot\ |w_1^n; \boldsymbol{\theta}_t)$ and $p(a_i = \cdot\ |a_1^{i-1}, w_1^n; \boldsymbol{\theta}_p)$.

In contrast with the model presented in Chapter 5, this statistical model is a global model: at each parsing step, the probability of an action is conditioned on the whole sentence and the whole sequence of previous actions.

6.3.1 A Shared Hierarchical Bidirectional LSTM Encoder

The shared component of the architecture consists of a hierarchical bi-LSTM encoder that computes context-aware vector representations for each word in the sentence.

A single character is represented by a character embedding. A word form w is represented by the concatenation of a word embedding $\mathbf{w}$ and of the output of a single layer word-level bi-LSTM encoder:

$$\mathbf{c} = [\text{LSTM}_f(w); \text{LSTM}_b(w)].$$

A sentence-level bi-LSTM encoder uses the representations $[\mathbf{w}; \mathbf{c}]_i$ at each position i in the sentence to construct context-aware representations $\mathbf{d}_i^{(1)}$ for each token. All these representations and the corresponding parameters (word embeddings, character embeddings, bi-LSTM parameters) are shared across all tasks.

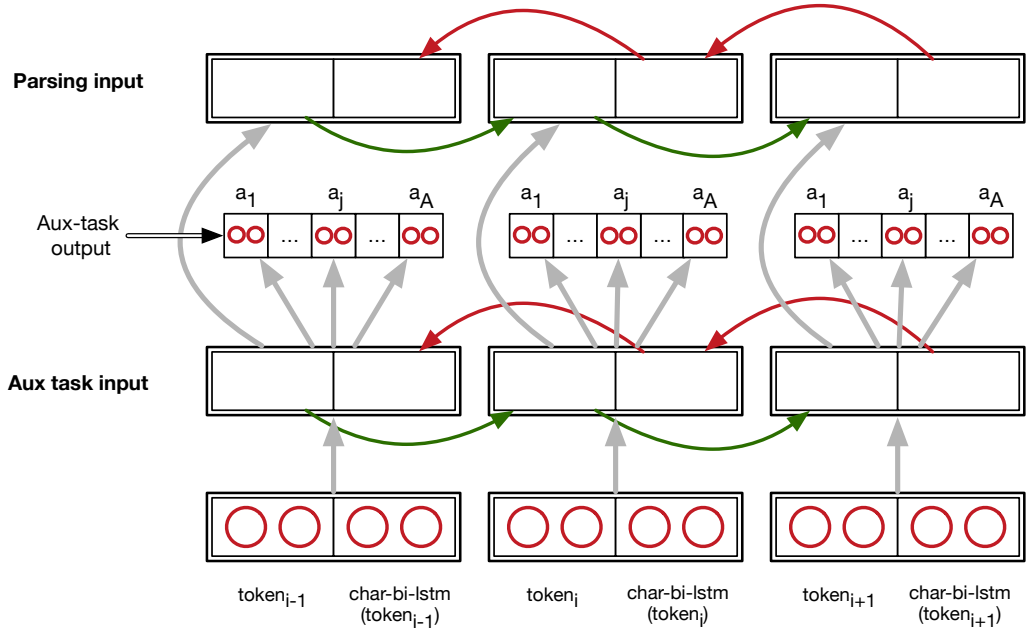


Figure 6.4: Deep bi-LSTM encoder with auxiliary tasks supervised at the first layer.

6.3.2 Tagging Component

We connect the representation $\mathbf{d}_i^{(1)}$ of each token to as many softmax output layers as there are labels to predict. The distribution for each typed $M_{i,j}$ of token w_i is defined as:

$$p(M_{i,j} = \cdot | w_1^n; \boldsymbol{\theta}_t) = \text{Softmax}(\mathbf{W}_t^{(j)} \cdot \mathbf{d}_i^{(1)} + \mathbf{b}_t^{(j)}) \tag{6.2}$$

where $\mathbf{W}_t^{(j)}$ and $\mathbf{b}_t^{(j)}$ are parameters for label j.

6.3.3 Parsing Component

A second bi-LSTM layer uses the sequence $[\mathbf{d}_1^{(1)}, \mathbf{d}_2^{(1)}, \ldots, \mathbf{d}_n^{(1)}]$ as input and computes higher-level representations $[\mathbf{d}_1^{(2)}, \mathbf{d}_2^{(2)}, \ldots, \mathbf{d}_n^{(2)}]$. Then, to represent a configuration, we extract typed features from the stack and buffer. The features used in practice are illustrated in Table 6.5. They consist of the left and right corners and the heads of constituents in the stack, as well as the labels of the first three constituents in the stack, and the first element in the buffer. Types are either nonterminal symbols or tokens. The nonterminal features are valued by an embedding. The token features

are valued by the corresponding representations $\mathbf{d}_i^{(2)}$. The concatenation of these valued features forms a vector $\mathbf{h}^{(0)}$ that is input to a feed-forward neural network with a softmax output layer that computes a distribution over possible actions:

$$\begin{aligned}
\mathbf{h}^{(1)} &= f(\mathbf{W}_p^{(1)} \cdot \mathbf{h}^{(0)} + \mathbf{b}_p^{(1)}) && \text{(hidden layer)} \\
\mathbf{h}^{(2)} &= f(\mathbf{W}_p^{(2)} \cdot \mathbf{h}^{(1)} + \mathbf{b}_p^{(2)}) && \text{(hidden layer)} \\
p(a_i = \cdot\,|a_1^{i-1}, w_1^n; \boldsymbol{\theta}_p) &= \text{Softmax}(\mathbf{W}_p^{(3)} \cdot \mathbf{h}^{(2)} + \mathbf{b}_p^{(3)}) && \text{(output layer)}
\end{aligned}$$

where $\mathbf{W}_t^{(l)}$ and $\mathbf{b}_t^{(l)}$ are parameters and f is a non-linear activation function.

In practice, the number of hidden layers for the feed-forward network and the choice of an activation function are hyperparameters. In our experiments, we used a network with 2 hidden layers, as presented above, and a rectifier (ReLU: $x \mapsto \max\{0, x\}$) as the activation function.

Several variants of this multitask architecture are possible (number of layers for the sentence level bi-LSTM). In particular, it is possible to share the two layers of the sentence-level bi-LSTM between the tagger and the parser, instead of just the first layer. Supervising different tasks at different levels of the hierarchical neural net has shown benefits in previous works (Søgaard and Goldberg, 2016).

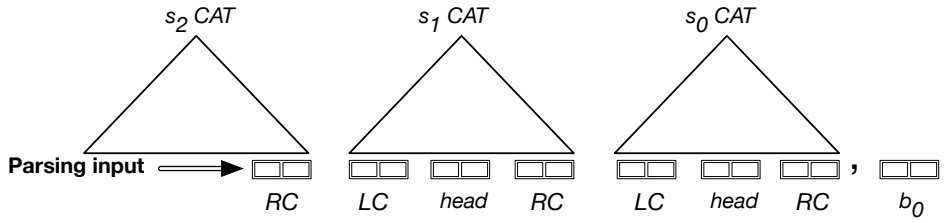
6.3.4 Objective Function and Training

To train the model, we optimize the negative log-likelihood of the data. The loss function for a single sentence w_1^n, with the corresponding gold sequence of actions a_1^k and gold labels M_1^n is defined as:

$$L(a_1^k, w_1^n, M_1^n; \boldsymbol{\theta}) = -\sum_{i=1}^{k} \log p(a_i|a_1^{i-1}, w_1^n; \boldsymbol{\theta}_p) - \sum_{i=1}^{n}\sum_{j=1}^{m} \log p(M_{i,j}|w_1^n; \boldsymbol{\theta}_t)$$

where $\boldsymbol{\theta} = \boldsymbol{\theta}_t \cup \boldsymbol{\theta}_p$. The first term is the objective function for the parser and the second term is the objective function for the tagger. The loss for the whole dataset is the sum of L for every sentence in the dataset. Although, we assume that each sentence has gold annotations for every task (parsing, morphological analysis, functional labelling), it is also possible to use a different dataset for each task.

Parser configuration:



Template set: s_0.*CAT*, s_0.*LC*, s_0.*RC*, s_0.*head*, s_1.*CAT*, s_1.*LC*, s_1.*RC*, s_1.*head*, s_2.*CAT*, s_2.*RC*, b_0

Figure 6.5: Feature templates for bi-RNN parsing. s and b respectively address symbols in the stack and the buffer.

	Input	Auxiliary tasks
TOK+CLSTM	token, character bi-LSTM	
TOK+CLSTM+M	token, character bi-LSTM	morphology
TOK+CLSTM+M+D	token, character bi-LSTM	morphology, functional labels
TOK	token	
TOK+MMT	token, predicted morphology	
TOK+MMT+D	token, predicted morphology	functional labels

Table 6.2: Summary of models.

6.4 Experiments

The experiments we conduct have several objectives. First, we assess to what extent the tagging auxiliary tasks can improve constituency parsing. Secondly, we evaluate the accuracy of the output of the auxiliary tasks. Finally, we compare our multitask model to a pipeline approach, where predicted morphological attributes are given as the input to the parser at test time.

In a first set of experiments, we use the model we described with a character-level bi-LSTM, and either no auxiliary task (TOK+CLSTM), morphological analysis as an auxiliary task (TOK+CLSTM+M) or morphological analysis and functional labelling as auxiliary tasks (TOK+CLSTM+M+D).

In a second set of experiments, the input to the sentence-level bi-LSTM does not include a character-based embedding. Instead, it is either a standard word-embedding (TOK), or the concatenation of a word embedding and embeddings for each available morphological tag (TOK+MMT). For

example, the token *étages* from the sentence above will be represented as the concatenation $[\mathbf{w}_{\text{étages}}; \mathbf{w}_{\text{g=m}}; \mathbf{w}_{\text{number=p}}; \mathbf{w}_{\text{tense=NA}}; \mathbf{w}_{\text{mood=NA}}]$. Finally, the last model uses the same input as TOK+MMT, but predicts functional labels as an auxiliary task (TOK+MMT+D). The different parameters of these models are summed up in Table 6.2.

6.4.1 Datasets

We evaluate our models on the SPMRL dataset (Seddah et al., 2013). This dataset contains constituency and dependency treebanks aligned at the word level for 9 morphologically rich languages. Each token is annotated with a part-of-speech tag and a number of language-specific morphological attributes (case, mood, tense, number).

In the first set of experiments, where morphology is predicted as an auxiliary task, we use the gold tags and morphological annotations at training time. At test time, the only input to the parser is a sequence of word forms.

In the second set of experiments, we use the POS and morphological tags predicted by MARMOT (Mueller et al., 2013) for training and parsing.[3] MARMOT is a CRF tagger designed to output a structured morphological analysis for each token, and to use external morphological lexicons.

As the transition system is lexicalized, we need to know the head of each constituent in order to extract the gold derivation. The constituency trees were head-annotated using the method of Crabbé (2015). This method uses the alignment between constituency and dependency trees to determine the head of each constituent and uses heuristics to solve mismatch cases. Finally, we performed a head-outward binarization with an order-0 Markovization (see Section 3.2.3.1), and collapsed unary productions to single nodes, except those that produce preterminals.

6.4.2 Protocol

We trained every model with Averaged Stochastic Gradient Descent (Polyak and Juditsky, 1992) and shuffled the training set before each iteration. In a single stochastic optimization iteration, we optimize successively the two terms of the objective function in equation 6.2. First, we compute the gradient of $\boldsymbol{\theta}_t$ with respect to L, and update it accordingly. Then we do the same operation with $\boldsymbol{\theta}_p$. We used the same learning rate for both sets of parameters. Between these two steps, we assign POS tags to tokens, as

[3] We used the tags available on MARMOT website.

Hyperparameters	Values
Optimization	
Iterations	$\{4, 8, 12, \ldots 28, 30\}$
Initial learning rate	$\{0.01, 0.02\}$
Learning rate decay constant	10^{-6}
Hard gradient clipping	5.0
Gaussian noise σ	0.01
Parameter initialization	Xavier initialization
Embedding initialization	Uniform($[-0.01, 0.01]$)
Output layers	
Number of hidden layers	2
Size of hidden layers	128
Activation	rectifiers
Word-level bi-LSTM	
Depth	2
Size of LSTM states	128
Word embeddings	32
Nonterminal embeddings	16
Morphological embeddings[a]	4, 8 or 16[b]
Char-level bi-LSTM[a]	
Depth	1
Size of LSTM states	32
Character embeddings	32

Table 6.3: Hyperparameters.
[a] When applicable.
[b] Depending on the number of possible values for this attribute.

	Arabic	Basque	French	German	Hebrew	Hungarian	Korean	Polish	Swedish	Avg
Experimental conditions	Development F1 (EVALBSPMRL)									
TOK+CLSTM	82.97	86.88	81.97	87.91	88.43	89.91	86.12	92.13	77.08	85.93
TOK+CLSTM+M	83.03	87.93	82.0	88.32	89.42	89.98	86.71	92.8	78.4	86.51
TOK+CLSTM+M+D	83.04	87.93	82.19	88.7	89.64	90.52	86.78	93.23	79.14	**86.8**
TOK	80.97	76.28	79.93	85.52	85.82	81.88	72.97	82.8	72.95	79.9
TOK+MMT	82.75	88.25	82.5	88.5	90.31	91.22	86.53	93.53	79.39	87.0
TOK+MMT+D	83.07	88.35	82.35	88.75	90.34	91.22	86.55	94.0	79.64	**87.14**
Hall et al. (2014)	78.89	83.74	79.40	83.28	88.06	87.44	81.85	91.10	75.95	83.30
Durrett and Klein (2015)	80.68	84.37	80.65	85.25	89.37	89.46	82.35	92.10	77.93	84.68
Coavoux and Crabbé (2016)	80.94	85.17	80.31	84.61	90.20	88.70	85.46	92.57	77.87	85.09
Experimental Conditions	Test F1 (EVALBSPMRL)									
TOK+CLSTM+M+D	**82.92**	87.87	82.1	85.12	89.19	90.95	85.89	92.67	83.44	86.68
TOK+MMT+D	82.77	**88.81**	82.49	**85.34**	**89.87**	**92.34**	**86.04**	**93.64**	84.0	**87.26**
Björkelund et al. (2014)	81.32[a]	88.24	**82.53**	81.66	89.80	91.72	83.81	90.50	**85.50**	86.12

Table 6.4: Parsing results on development and test corpora (SPMRL evaluator). [a]Björkelund et al. (2013).

they are considered nonterminals once they are shifted onto the stack and used as features by the parser.[4]

We optimized hyperparameters on the development sets for each language with a grid search over a very small set of models (16 per language). The full list of hyperparameters is in Table 6.3. The initial states of bi-LSTM are learned parameters. Following Kiperwasser and Goldberg (2016), we stochastically replace a token in the training set by an UNKNOWN pseudoword with a probability $p_w = \frac{\alpha}{\#\{w\}+\alpha}$, where $\#\{w\}$ is the raw frequency of w in the training set and $\alpha = 0.8375$, as suggested by Cross and Huang (2016a). In the development and test sets, the unknown words are replaced by the same pseudoword. This replacing does not affect the character bi-LSTM that has still access to the sequence of characters of the unknown word. Finally, we used greedy decoding for all our experiments.

6.4.3 Results and Discussion

We first discuss the effect of the auxiliary tasks on constituency parsing, then we compare our best models to other published results on this dataset.

[4]The usual policy for training a multitask architecture is to sample a task randomly at each stochastic step. Instead, we consider all tagging auxiliary tasks as if they were only one task, in order to decrease training time, as running the forward and backward propagation on the sentence-level bi-LSTM is the most expensive part of training. However it is likely that other training policies might be better adapted to this architecture.

6.4.3.1 Effect of Tagging Auxiliary Tasks

Morphological Analysis Parsing results are presented in Table 6.4. We first discuss results on the development sets. The comparison between TOK+CLSTM and TOK shows that the simple addition of the character-level bi-LSTM improves parsing by a large margin (+ 6 on average). The most important improvements are for Basque, Hungarian, Korean (i.e. agglutinative languages) and Polish.

Adding the morphological auxiliary tasks improves the model by 0.6 on average (TOK+CLSTM vs TOK+CLSTM+M), which shows that adding morphological supervision during training to constrain representations to be good predictors of morphological information has a beneficial effect for parsing.

Still, the parser that models morphology as auxiliary tasks (TOK+CLSTM+M) underperforms the one that uses predicted morphological tags as input (TOK+MMT). Across languages, the performance difference between the two models can be partly explained by the difference in tagging accuracy (Table 6.5). The TOK+CLSTM+M model matches MARMOT tagging results for several languages, but is not as good overall. MARMOT uses morphological lexicons as an additional source of information, which might be crucial for languages such as Basque.

Functional Labelling Finally, results show that in the two settings (TOK+CLSTM+M vs TOK+CLSTM+M+D and TOK+MMT vs TOK+MMT+D) the additional auxiliary task of predicting the dependency labels brings a small but consistent improvement (+0.3 and +0.1 on average). We conclude that functional information is useful for constituency parsing.

6.4.3.2 Comparison with Existing Results

Constituency Trees On the test sets (Table 6.4), we compare our models to the parser of Björkelund et al. (2014) that is based on a product of PCFGs with latent annotations, and a discriminative reranker using morphological features. Our TOK+CLSTM+M+D model outperforms their reranker by 0.5 F_1, despite using a greedy decoding and requiring no other inputs than the raw sequence of tokens at test time.

Labelled Dependency Trees As our model is lexicalized, we can extract unlabelled dependency trees from its output. As a byproduct of the functional label auxiliary tasks, we can obtain labelled dependency trees instead. Thus, we also evaluate the output of our parser against the

		Arabic	Basque	French[c]	German	Hebrew[c]	Hungarian	Korean[c]	Polish[c]	Swedish[c]
Experimental Conditions	Decoding	Development results – POS-Tagging[d]								
TOK+CLSTM+M	greedy	97.66	95.7	97.58	98.39	95.71	98.06	94.42	97.02	96.88
MARMOT[a]	CRF+lexicons	97.38	97.02	97.61	98.10	97.09	98.72	94.03	98.12	97.27
		Test results – UAS/LAS								
TOK+CLSTM+M+D	greedy	81.5/78.7	75.8/68.9	88.0/83.1	67.1/64.1	84.5/75.3	74.5/69.5	89.9/87.3	88.2/80.0	86.3/76.5
TOK+MMT+D	greedy	81.3/78.6	76.8/71.2	**87.8/83.5**	67.2/64.7	**85.8/77.3**	75.9/72.0	**89.6/87.5**	**89.6/83.1**	**86.7/78.5**
Ballesteros et al. (2015)	greedy	**86.1/83.4**	**85.2/78.6**	86.2/82.0	**87.3/84.6**	80.7/72.7	**80.9/76.3**	88.4/86.3	87.1/79.8	83.4/76.4
Best published[b]	ens+reranker	88.3/86.2	90.0/85.7	89.0/85.7	91.6/89.7	87.4/81.7	89.8/86.1	89.1/87.3	91.8/87.1	88.5/82.8

Table 6.5: Dependency parsing and tagging results.
[a]Uses external morphological lexicons (Björkelund et al., 2013).
[b]Either Björkelund et al. (2013) or Björkelund et al. (2014).
[c]Languages with few head mismatches between the dependency and the constituency corpora (Crabbé, 2015).
[d]Tagging is evaluated with the dependency treebanks (the tagsets used in the constituency treebanks might differ).

dependency corpora using the evaluator provided with the SPMRL shared task (Seddah et al., 2013).

Dependency results are shown in Table 6.5. Our parser outperforms Ballesteros et al. (2015), the best published results with a greedy parser, on 5 languages out of 9. Unsurprisingly, these languages correspond to the corpora, identified by Crabbé (2015), which contain very few mismatch cases between the dependency and the constituency treebank. This result is in keeping with Cer et al. (2010) who have shown that constituency parsers are very good at recovering dependency structures for English. Our experiments confirm this finding in a multilingual setting where labelled dependency trees are directly predicted by the parser, rather than obtained by conversion of predicted constituency trees.

6.4.3.3 Limitation and Perspective

We plotted the learning curves for the TOK+CLSTM+M+D model trained of the French dataset in Figure 6.6, focusing on three tasks: the parsing task, the POS tagging task and the functional labelling task. The model learns these three tasks at different speeds. For POS tagging, the learning curve is nearly flat after 3 iterations for both the train and development sets, whereas on the parsing task, generalization improves fast until the fifteenth iteration, and we still observe very small improvements afterwards. The functional labelling is in-between and needs roughly eight iterations to reach its best performance.

This is not an ideal situation, as multitask learning works best when every task "learn at similar rates and reach best performance at roughly

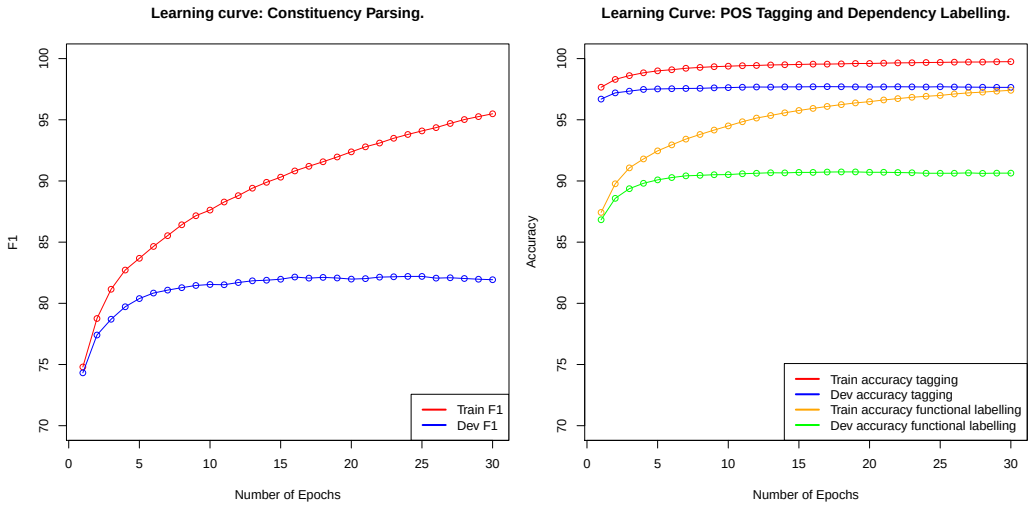


Figure 6.6: Learning curves for the TOK+CLSTM+M+D model trained on the French dataset.

the same time" (Caruana, 1997). This issue may be addressed by using different learning rates for the different tasks.

Finally, improving on POS tagging might lead to better results in constituency parsing. In this respect, the use of morphological lexicons seems promising, for example to constrain the possible tags and morphological attributes for each token.

6.5 Conclusion

In this chapter, we have presented a multitask method to model interfaces of constituency parsing, namely morphological analysis and functional labelling. We have investigated how to use morphology and functional labelling as additional training signal to improve constituency parsing, and to enrich the parser output. Our multitask model has obtained state-of-the art results on the SPMRL dataset despite using a very simple search algorithm and not requiring any other input than the raw sequences of tokens.

Part III

Discontinuous Constituency Parsing

Chapter 7

Discontinuous Constituency Parsing

Contents

7.1 Introduction **125**
7.2 The Shift-Reduce-Gap Transition System **127**
7.2.1 Algorithm 127
7.2.2 Oracle and Properties 133
7.2.3 Comparing Derivations with Different Lengths 136
7.3 Experiments **138**
7.3.1 Datasets 139
7.3.2 Classifier: Structured Perceptron 139
7.3.3 Results 140
7.4 Discussion: a Comparison of SR-SWAP and SR-GAP **145**
7.4.1 Derivation Length 145
7.4.2 Feature Semantics and Feature Locality 148
7.5 Conclusion **150**

7.1 Introduction

Allowing crossing branches in constituency trees is a way to directly model word order variation phenomena and extractions, as opposed to

additional annotation layers such as coindexed empty categories. In this chapter, we introduce a parsing algorithm that is incremental, efficient, and very accurate in predicting discontinuous constituency trees.

In the constituency parsing tradition, the vast majority of parsers makes the hypothesis that syntactic trees do not contain empty categories (Petrov et al., 2006; Zhu et al., 2013). As a matter of fact, the standard version of the Penn Treebank used for the training and evaluation of parsers has been stripped of empty categories and functional annotations. One of the initial reasons for this is the difficulty to predict empty categories.[1]

Early attempts at discontinuous parsing are based on probabilistic grammars (Plaehn, 2004; Kallmeyer and Maier, 2010, 2013) or on a reduction to dependency parsing with a reversible transformation of discontinuous constituency trees to labelled non-projective dependency trees (Hall and Nivre, 2008; Fernández-González and Martins, 2015). The parsers based on explicit generative grammars may in principle reject a non-grammatical sentence, but the drawback of these methods is that decoding is very expensive. For example, exact decoding with a binary probabilistic LCFRS of fan-out k has a time complexity in $\mathcal{O}(n^{3k})$ (Kallmeyer, 2010). They usually need to resort to heuristics (Kallmeyer and Maier, 2010; van Cranenburgh et al., 2016) to make parsing times reasonable, and scale hardly to long sentences. On the contrary, methods based on dependency parsing have achieved very strong results recently (Fernández-González and Martins, 2015).

Transition-based parsing for discontinuous constituents was first proposed by Versley (2014b,a) who used an easy-first approach (Goldberg and Elhadad, 2010b). In this transition system, discontinuities are handled with a swap action. Finally, Maier (2015) and Maier and Lichte (2016) introduced a shift-reduce transition system augmented by a swap action (shift-reduce-swap, SR-SWAP) as is done in dependency parsing (Nivre, 2009; Nivre et al., 2009). The main advantage of such methods is their efficiency. They have a complexity in $\mathcal{O}(k \cdot n^2)$ where k is the size of the beam and scale to whole corpora. Despite being much more accurate than Versley (2014b), shift-reduce-swap falls short of the parser of Fernández-González and Martins (2015). One of the reason for this, we argue, is that oracles for shift-reduce-swap tend to produce very long derivations and produce configurations in which local features are not informative enough (cf Section 7.4).

[1] However, there have been a few attempts at predicting empty categories either with a post-processing step (Johnson, 2002; Levy and Manning, 2004; Takeno et al., 2015) or by integrating empty element prediction into parsing (Schmid, 2006; Cai et al., 2011; Hayashi and Nagata, 2016).

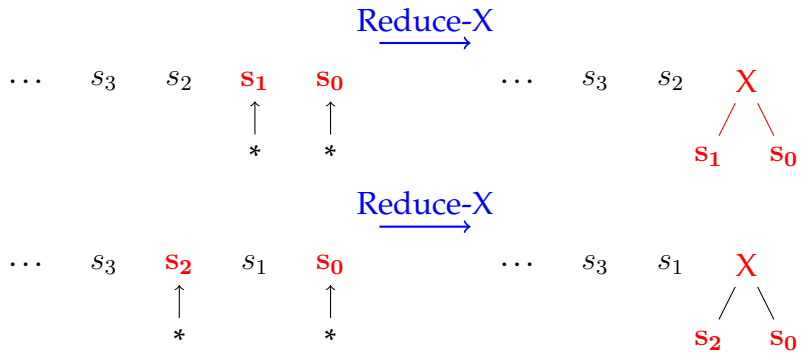


Figure 7.1: Illustration of a reduction with the standard shift-reduce transition system (upper part) and with SR-GAP (lower part).

This chapter introduces shift-reduce-gap (SR-GAP), a lexicalized transition system able to output discontinuous constituency trees. The algorithm relies on three data structures to define a parsing configuration (instead of two in the standard shift-reduce algorithm). In this respect, it is akin to Covington's dependency parsing algorithms (Covington, 2001) and their formulation as transition systems by Nivre (2008), who describes them as *list-based*.

The chapter is structured as follows. First, we introduce the shift-reduce-gap transition system (Section 7.2). Then, we evaluate SR-GAP in several experimental settings (Section 7.3). Finally, we provide a comparison of SR-GAP and SR-SWAP in order to understand the performance of SR-GAP (Section 7.4).

7.2 The Shift-Reduce-Gap Transition System

First, we introduce the SR-GAP transition system (Section 7.2.1). Then, we provide a static oracle for SR-GAP and investigate its formal properties (Section 7.2.2). Finally, we discuss and address the problem of comparing derivations with different lengths (Section 7.2.3).

7.2.1 Algorithm

The shift-reduce-gap transition system is an extension of the lexicalized shift-reduce algorithm presented in Section 3.2.1, that is able to output discontinuous trees. Like the standard shift-reduce algorithm, it outputs bi-

nary trees. In order to use it to predict n-ary trees, a parser relies on the reversible head-outward binarization algorithm presented in Section 3.2.3.1.

In the shift-reduce algorithm, reductions always apply to the two topmost elements in the stack. Consequently, derived trees are projective because constituents in the stack always have contiguous spans (Figure 7.1, upper part). In contrast, SR-GAP may perform reductions applied to constituents that can be arbitrarily distant in the stack. We illustrate such a reduction in the lower part of Figure 7.1. A reduction applied to the first and the third element in the stack produces a constituent whose yield is not contiguous.

The main difference between SR-GAP and a standard lexicalized shift-reduce system is that SR-GAP can choose dynamically an element in the stack and combine it with the item at the top of the stack in a reduction. To do so, SR-GAP splits the usual stack into two data structures:

- A stack S (upper part of the stack, stores the older constituents);
- A deque D (lower part of the stack, stores the later constituents).

Reductions always apply to the top elements of, respectively, S and D (corresponding to the two '*' in Figure 7.1).

In order to control the movement of items between S and D, SR-GAP uses an action called GAP. This action removes the topmost item of S and pushes it onto the bottom of D, making the new topmost item of S available for a potential reduction.

In SR-GAP, a parsing configuration consists of three data structures: S, D and the usual buffer B. The whole transition system is presented as a deduction system in Table 7.1. The available actions have the following effects:

- SHIFT flushes the content of D onto S, pops a token from the buffer and pushes it onto D
- REDUCE-UNARY-X pops the topmost element of D (d_0), creates a new node labelled X with d_0 as its single child and pushes it onto D.
- REDUCE-LEFT-X and REDUCE-RIGHT-X pop the topmost elements of S and D (d_0 and s_0), create a new node labelled X with s_0 and d_0 as its children, flush the content of D to S and pushes the new constituent on D. These actions also assign a lexical head to the new constituent.
- GAP pops the topmost item in S and pushes it at the bottom of D. It has to be noted that this action has no effect on d_0.

Input	$t_1[w_1]t_2[w_2]\ldots t_n[w_n]$
Axiom	$\langle \epsilon, \epsilon, t_1[w_1]t_2[w_2]\ldots t_n[w_n]\rangle$
Goal	$\langle \epsilon, A[w], \epsilon\rangle$
SHIFT	$\frac{\langle S, D, t[w]\|B\rangle}{\langle S\|D, t[w], B\rangle}$
REDUCE-UNARY-X	$\frac{\langle S, d_0[h], B\rangle}{\langle S, X[h], B\rangle}$
REDUCE-RIGHT-X	$\frac{\langle S\|s_0[h], D\|d_0[h'], B\rangle}{\langle S\|D, X[h'], B\rangle}$
REDUCE-LEFT-X	$\frac{\langle S\|s_0[h], D\|d_0[h'], B\rangle}{\langle S\|D, X[h], B\rangle}$
GAP	$\frac{\langle S\|s_0[h], D, B\rangle}{\langle S, s_0[h]\|D, B\rangle}$
IDLE	$\frac{\langle \epsilon, A[w], \epsilon\rangle}{\langle \epsilon, A[w], \epsilon\rangle}$

Table 7.1: Lexicalized Shift-Reduce-Gap transition system for discontinuous phrase structure parsing. $X[h]$ denotes a nonterminal X and its head h. s_0 and d_0 denote the topmost elements of respectively S and D. A is the axiom symbol. The IDLE action is introduced in Section 7.2.3.

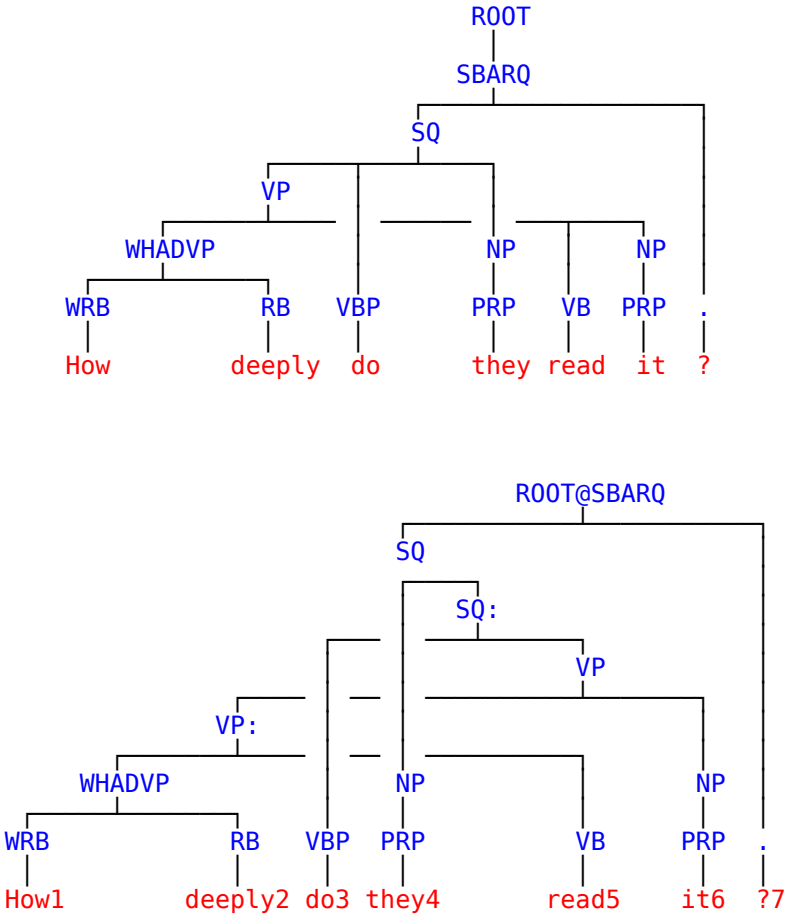


Figure 7.2: Example tree from the Discontinuous Penn Treebank (Evang and Kallmeyer, 2011). Lower part: preprocessed binarized tree. Temporary symbols are suffixed by ":". "@" indicates a merged unary chain.

The only case where D has more than one element is after a GAP action, because other types of actions flush D to S. To derive a projective tree, the parser does not perform any GAP, and D always contains a single element. Finally, the parser can perform several consecutive GAPS to access older constituents. Performing a GAP is possible as long as S contains at least two elements.

7.2.1.1 An Example Run

We present a full derivation for the tree in Figure 7.2 in Table 7.2. In this table, RR, RL, and RU denote respectively a right, left, and unary

Transitions	Configurations		
	S	D	B
			How deeply do they read it ?
SHIFT ⟹		WRB[How]	deeply do they read it ?
SHIFT ⟹	WRB[How]	RB[deeply]	do they read it ?
RL-WHADVP ⟹		WHADVP[How]	do they read it ?
SHIFT ⟹	WHADVP[How]	VBP[do]	they read it ?
SHIFT ⟹	WHADVP[How] VBP[do]	PRP[they]	read it ?
RU-NP ⟹	WHADVP[How] VBP[do]	NP[they]	read it ?
SHIFT ⟹	WHADVP[How] VBP[do] NP[they]	VB[read]	it ?
GAP ⟹	WHADVP[How] VBP[do]	NP[they] VB[read]	it ?
GAP ⟹	WHADVP[How]	VBP[do] NP[they] VB[read]	it ?
RR-VP: ⟹	VBP[do] NP[they]	VP:[read]	it ?
SHIFT ⟹	VBP[do] NP[they] VP:[read]	PRP[it]	?
RU-NP ⟹	VBP[do] NP[they] VP:[read]	NP[it]	?
RL-VP ⟹	VBP[do] NP[they]	VP[read]	?
GAP ⟹	VBP[do]	NP[they] VP[read]	?
RL-SQ: ⟹	NP[they]	SQ:[do]	?
RL-SQ ⟹		SQ[do]	?
SHIFT ⟹	SQ[do]	.[?]	
RL-ROOT@SBARQ ⟹		ROOT@SBARQ[do]	

Table 7.2: Full derivation for the tree in Figure 7.2.

reduction. Although the transition system assumes that the sentence is already tagged and that tokens consist of couples made of a word form and a POS tag, we do not include the POS tags in the buffer to improve legibility.

7.2.1.2 Interpretation of the GAP Action

In transition-based parsing, the stack is often interpreted as a memory. It stores items that have already been seen, but not completely processed. The newest items are always at the top of the stack, whereas the oldest are at the bottom. In projective parsing, the parser only needs to access the top of the stack, i.e. the most recent items. In discontinuous parsing, the parser needs to access older elements to construct discontinuous constituents. In that respect, the GAP action can be seen as a mechanism to access any element in the stack in linear time. The parser must perform $n - 1$ operations to access the n^{th} most recent item.

This might be an interesting property from a psycholinguistic perspective. If we were to measure cognitive cost in terms of the number of parsing operations, SR-GAP predicts that retrieving a long distance dependency incurs a higher cost than constructing local dependencies.

7.2.1.3 Relationship with Covington's Algorithm

Covington (2001) described a general algorithm for unrestricted dependency parsing that consists in iterating over every pair of words in a sen-

tence and deciding for each pair whether to link them with a dependency arc. In Algorithm 10, the function LINK performs one of the following operations:

- adding a right arc $i \rightarrow j$;
- adding a left arc $i \leftarrow j$;
- doing nothing.

This high-level algorithm can be refined to ensure that the predicted structure is a single tree and formulated as a transition system based on three data structures (Nivre, 2008; Gómez-Rodríguez and Fernández-González, 2015).

Algorithm 10 Covington's algorithm, as formulated by Nivre (2007).

```
function PARSE(s = (w_1, w_2, ..., w_n))
    for i = 1 to n do
        for j = i - 1 downto 1 do
            LINK(i, j)
```

The outer loop of the algorithm corresponds to successive SHIFTS, while the inner loop constructs every dependency relation involving token i and a preceding token. In fact, SR-GAP's strategy is rather similar: after having shifted token i, it constructs every constituent whose last token is i,[2] before shifting token $i + 1$.

7.2.1.4 Preconditions on Actions

Preconditions on actions are needed to ensure the termination of the algorithm and certain structural constraints on predicted trees. We present all preconditions in Table 7.3.

As the SHIFT action flushes the content of D to S before pushing a new token, it cancels the effect of preceding GAPS. To avoid useless GAPS, we impose that only a binary reduction or another GAP can follow a GAP. Furthermore, as the only unary constituents produce preterminals, a REDUCE-UNARY action can be performed only immediately after a SHIFT. Figure 7.4(a) shows an automaton that encodes these constraints. Any legal action sequence must be recognized by this automaton. State S is the state in which a REDUCE-UNARY is possible, and it is reached only with

[2]Following a particular order described in Section 7.2.2.3.

Action	Preconditions
SHIFT	B is not empty. The last action is not GAP.
GAP	S has at least 2 elements. If d_0 is a temporary symbol, there must be at least one non temporary symbol in $S_{1:}$.
REDUCE-UNARY-X	The last action is SHIFT. X is an axiom iff this is a one-word sentence.
REDUCE-RIGHT-X, REDUCE-LEFT-X	S is not empty. X is an axiom iff B is empty, and S and D both have exactly one element. If X is a temporary symbol and if B is empty, there must be a non-temporary symbol in either $S_{1:}$ or $D_{1:}$.
REDUCE-RIGHT-X	s_0 is not a temporary symbol.
REDUCE-LEFT-X	d_0 is not a temporary symbol.
IDLE	The configuration must be final, i.e. S and B are empty and the only element of S is the axiom.

Table 7.3: List of all preconditions for SR-GAP actions. The notation $S_{1:}$ is used to denote the elements of S without the first one. The IDLE action is introduced in Section 7.2.3.

a SHIFT action. State G is reached by a GAP and can only be left with a binary reduction.

Other preconditions make sure that the predicted lexicalized trees can be unbinarized. The two conditions that need to be satisfied is that no nonterminal rewrites as two temporary symbols and that a temporary symbol contains the lexical head of its parent. These issues were also discussed in Section 3.2.3.

7.2.2 Oracle and Properties

After introducing some definitions, we present an oracle for SR-GAP as well as formal properties of this algorithm.

7.2.2.1 Preliminary Definitions

We first introduce some definitions necessary to characterize the oracle in the next sections. Following Maier and Lichte (2016), we define a discontinuous tree as a rooted connected directed acyclic graph $T = (V, E, r)$ where

- V is a set of nodes;
- $r \in V$ is the root node;
- $E : V \times V$ is a set of (directed) edges and E^* is the reflexive transitive closure of E.

If $(u, v) \in E$, then u is the parent of v. Each node has a unique parent (except the root that has none). Nodes without children are *terminals*.

The *right index* (resp. *left index*) of a node is the index of the rightmost (resp. leftmost) terminal dominated by this node. For example, the left index of the node labelled VP: in Figure 7.2 is 1 and its right index is 5.

7.2.2.2 Oracle

In order to extract the gold derivation for a given tree, we repeat the following steps, starting with the initial configuration (and indexing items in S and D with s_i and d_i respectively) to derive new configurations:

- If the configuration is final, stop.
- If s_0 and d_0 have the same parent node in the gold tree, perform a reduction with the parent node label. Choose the head direction by looking at the gold tree.
- If s_0 and d_i have the same parent, perform i GAPS.
- Otherwise perform a SHIFT possibly followed by a REDUCE-UNARY-X if the shifted terminal's parent has a single child.

The derivation in Table 7.2 illustrates this oracle.

7.2.2.3 Determinism of the Oracle

Given the preconditions on actions presented above, the oracle is inherently deterministic. If we ignore lexicalization and consider only one type

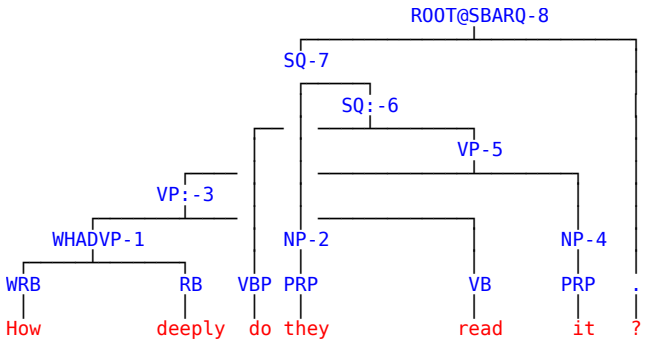


Figure 7.3: Illustration of the order $\prec$ on nodes.

of binary reduction,[3] there is a unique gold derivation. Let $\prec$ be an order relation on the internal nodes of a tree. For two nodes n and n', let $n \prec n'$ iff (i) $rindex(n) < rindex(n')$ or (ii) $(n', n) \in E^*$. This order is total: if n and n' have the same right index i, they both dominate terminal i, which implies that either n dominates n' or n' dominates n. The order $\prec$ is illustrated in Figure 7.3, where each node n is annotated with an integer i_n and $n \prec n'$ iff $i_n < i_{n'}$.

If a derivation predicts two nodes n and n' such that $n \prec n'$, then the reduction to n must necessarily precede the reduction to n' in a derivation. The right index of d_0 is non-decreasing during a derivation and corresponds exactly to the number of preceding SHIFTS. A node with right index i cannot be constructed if the number of preceding SHIFTS in the derivation is not i. Therefore, if $rindex(n) < rindex(n')$ (i), then n must be constructed before n'. Moreover, if n' is an ancestor of n (ii), it is necessary that n be constructed before n' in a derivation. As a consequence, all the nodes of the tree must be constructed in the order $\prec$, from which we conclude that there is a unique possible derivation for a given tree.

7.2.2.4 Completeness and Soundness of SR-GAP

The shift-reduce-gap transition system is sound and complete for the set of discontinuous binary trees labelled with a set of nonterminal symbols. Given the constraints shown in Table 7.3, this result also holds for the set of discontinuous n-ary trees (modulo unbinarization).

Completeness is a consequence of the correctness of the oracle which corresponds to a tree traversal in the order $\prec$. To prove soundness, we

[3]Otherwise, there can be several derivations that construct the same tree but with different head assignments.

need to show that any valid derivation sequence produces a discontinuous binary tree. It holds from the transition system that no node can have several parents, as parent assignation via REDUCE actions pops the children nodes and makes them unavailable to subsequent reductions. This implies that at any moment, the content of the stack is a forest of discontinuous trees. Preconditions in Table 7.3 ensure that at least one action is allowed at each parsing step.

The number of actions in a derivation for a sentence of size n is upper bounded: there can be at most n SHIFT, n unary REDUCE, $n-1$ binary REDUCE and less than $\frac{1}{2} \cdot n^2$ GAP (see Section 7.4.1 for a precise bound). As a consequence, the algorithm can always reach a final configuration, where the forest only contains one discontinuous tree.

The correctness of the shift-reduce-gap transition system holds for the set of all labelled discontinuous trees, and not for the set of all trees generated by an LCFRS grammar. Our transition system is not able to decide grammaticality. From the perspective of robust analysis, this is not a limitation: the ability to test grammaticality is traded off for parsing efficiency and accuracy. The nature of the relationship between shift-reduce-gap and automata designed explicitly for LCFRS parsing (de la Clergerie, 2002; Kallmeyer and Maier, 2015) requires further investigations.

7.2.3 Comparing Derivations with Different Lengths

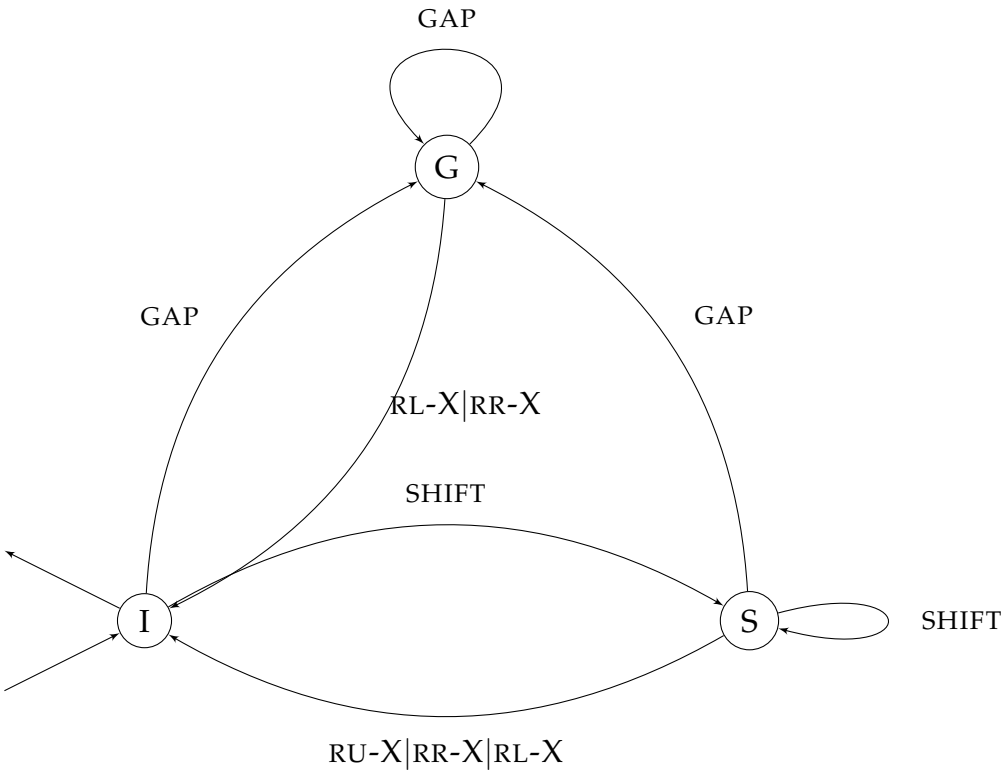
For a given sentence, different derivations corresponding to potential hypotheses may have different lengths. If the number of binary REDUCE and SHIFT depends only on the sentence length, the number of GAP and unary REDUCE may widely vary.

As stated in Section 3.2.1.2, this problem is recurrent in constituency parsing, because it harms comparability between hypotheses. Depending on the model, classifiers may be biased towards shorter or longer derivations. For example, Crabbé (2014) observes that the score given by a perceptron to a derivation is approximately linear in the number of actions in the derivation.

Existing strategies for dealing with this problem consist in including actions with no effect in the transition system (Zhu et al., 2013; Crabbé, 2014; Mi and Huang, 2015) and modifying the preconditions on actions to make sure that every derivation has the same length.

Following these authors, we tried two strategies. First, we use an additional IDLE action (Zhu et al., 2013) that has no effect and can only

(a) SHIFT-REDUCE-GAP



(a) SHIFT-REDUCE-COMPOUND-GAP

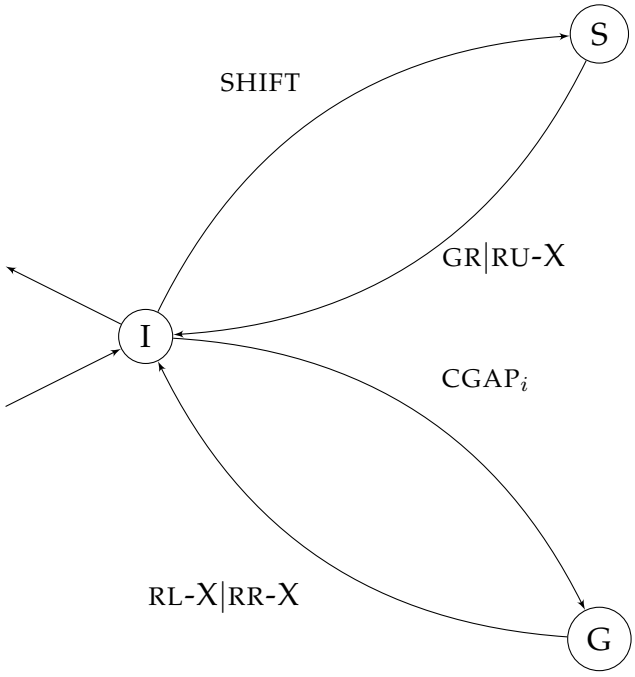


Figure 7.4: Automata of licit action sequences for SR-GAP and SR-CGAP.

be applied to a final configuration. This action is used to pad shorter derivations until every derivation in the beam has the same length.

The second strategy relies on a variant of the shift-reduce-gap transition system called shift-reduce-compound-gap (SR-CGAP), which addresses independently the two sources of varying derivation length: the unary REDUCE and the GAPS. For the former, SR-CGAP uses the strategy of Crabbé (2014): each SHIFT must mandatorily be followed by a unary REDUCE or a GHOST-REDUCE (transition with no effect). For the latter, SR-CGAP requires that every binary REDUCE is preceded by exactly one compound GAP_i action. Akin to the compound SWAP_i action of Maier (2015), a GAP_i bundles $i \in \{0, 1, \ldots, m\}$ GAPS in a single transition. For example, GAP_0 has no effect, GAP_2 corresponds to two consecutive GAPS. There are in total $m + 1$ distinct compound GAP actions. Figure 7.4(b) shows an automaton that recognizes any licit sequence of actions. From initial state I, the parser either performs a SHIFT, followed by GHOST-REDUCE or REDUCE-UNARY-X, or a COMPOUND-GAP_i followed by a binary reduction.

The set of transitions in SR-CGAP and the new constraints on them make sure that any derivation for a sentence of length n will have exactly $4n - 2$ actions: n SHIFTS, n unary or ghost REDUCE, $n - 1$ binary REDUCE and $n - 1$ compound GAP_i.

The SR-CGAP transition system has a $\mathcal{O}(n)$ time complexity (whereas SR-GAP is in $\mathcal{O}(n^2)$). However, SR-CGAP is not complete: it cannot derive a tree which requires a compound GAP_i with $i > m$ (corresponding in SR-GAP to more than m consecutive GAPS). In our experiments, we chose the maximum index m of a compound GAP to be the minimum possible m such that it is possible to derive all the trees in the training set.

7.3 Experiments

The experiments we carry out aim at evaluating our transition systems' performance in different settings and their ability to recover discontinuities in comparison with other approaches.

We first present the datasets (Section 7.3.1) as well as the classifier (Section 7.3.2) that we used. Then, we discuss our results (Section 7.3.3).

7.3.1 Datasets

7.3.1.1 Corpora

For evaluation, we used the Tiger (Brants et al., 2002) and the Negra (Skut et al., 1997) corpus. Maier and Lichte (2011) report that 27.5% of sentences in the Negra corpus contain at least one discontinuity and 29.0% of sentences in the Tiger corpus. For both corpora, approximately 3% of sentences have a gap-degree greater than 1.

To compare fairly with previous work, we used several instantiations of these corpora:

- NEGRA-30 contains sentences with fewer than 30 words and follows the train-dev-test split described by Maier (2015);
- NEGRA-ALL is the full corpus and uses the split of Dubey and Keller (2003);
- TIGERHN8 is the split described by Hall and Nivre (2008);
- TIGERM15 is the split described by Maier (2015), and previously used in the SPMRL shared task Seddah et al. (2013).

7.3.1.2 Preprocessing

We applied the following preprocessing steps to the trees. First, we removed functional labels on nonterminal nodes. In these corpora, punctuation is usually attached to the root symbol, which causes spurious discontinuity. As is standard practice, we reattached punctuation lower to avoid this type of discontinuity. Finally, we annotated the heads of constituents with headrules, using the rules included in the DISCODOP distribution (van Cranenburgh et al., 2016), and we binarized the trees with the left-first head-outward binarization algorithm presented in Section 3.2.3.1.

7.3.2 Classifier: Structured Perceptron

We used an averaged global perceptron (Collins, 2002) with early update (Collins and Roark, 2004). Although other statistical models based on neural networks may give much better results (see Chapter 8), in this chapter, we want to evaluate SR-GAP as a transition system. Therefore we decided on a structured perceptron to compare fairly with other transition systems (Maier, 2015; Versley, 2014b) that also use a perceptron. In every

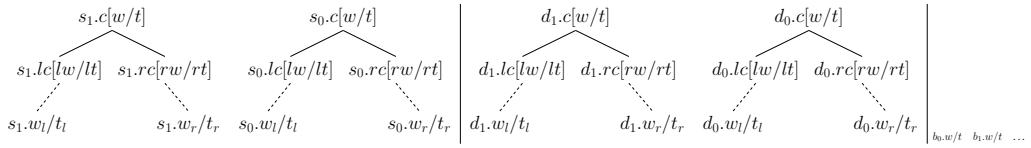


Figure 7.5: Abstract configuration: representation of the topmost elements of S, D and B, using the notations introduced in Table 7.4. Due to discontinuities, it is possible that both the left- and right- index of s_i are generated by the same child of s_i.

experiment, we trained the model for 30 epochs and shuffle the training set before each epoch.

We tried several feature template sets to inform the classifier, which we describe here. The full definitions are presented in Table 7.4.

- The BASELINE set contains the templates usually used by a (projective) constituency parser (Zhu et al., 2013);
- The +EXTENDED set adds information about D (d_1 and d_2), i.e. items that are inside a gap, as well as extended context on S;
- The +SPANS set adds templates based on constituent boundaries (Hall et al., 2014), following the intuition that they often contain functional words.

7.3.3 Results

We first compare our different models in various settings to understand which experimental choices give the best results. Then, we compare our best models to results published by other researchers.

In every setting, we evaluate the unbinarized predicted trees with the evaluation tool of DISCODOP using standard evaluation parameters that ignore punctuation and root symbols. This evaluator computes evalb-style evaluation metrics for discontinuous constituency trees. In a few cases, we used the SPMRL shared task evaluation parameters.[4] We report 2 different metrics (Tables 7.6 and 7.5): F1 is the standard F-measure and Disc. F1 is an F-measure computed only on discontinuous constituents.

[4]When we do so, we evaluate trees after reattaching the punctuation to the root node, because these parameters take punctuation into account.

BASELINE				
$b_0.tw$	$b_1.tw$	$b_2.tw$	$b_3.tw$	$d_0.tc$
$d_0.wc$	$s_0.tc$	$s_0.wc$	$s_1.tc$	$s_1.wc$
$s_2.tc$	$s_2.wc$	$s_0.lwlc$	$s_0.rwrc$	$d_0.lwlc$
$d_0.rwrc$	$s_0.wd_0.w$	$s_0.wd_0.c$	$s_0.cd_0.w$	$s_0.cd_0.c$
$b_0.wd_0.w$	$b_0.td_0.w$	$b_0.wd_0.c$	$b_0.td_0.c$	$b_0.ws_0.w$
$b_0.ts_0.w$	$b_0.ws_0.c$	$b_0.ts_0.c$	$b_0.wb_1.w$	$b_0.wb_1.t$
$b_0.tb_1.w$	$b_0.tb_1.t$	$s_0.cs_1.wd_0.c$	$s_0.cs_1.cd_0.c$	$b_0.ws_0.cd_0.c$
$b_0.ts_0.cd_0.c$	$b_0.ws_0.wd_0.c$	$b_0.ts_0.wd_0.c$	$s_0.cs_1.cd_0.w$	$b_0.ts_0.cd_0.w$
+ EXTENDED				
$s_3.tc$	$s_3.wc$	$s_1.lwlc$	$s_1.rwrc$	$d_1.tc$
$d_1.wc$	$d_2.tc$	$d_2.wc$	$s_0.cs_1.cd_0.c$	$s_2.cs_0.cs_1.cd_0.c$
$s_0.cd_1.cd_0.c$	$s_0.cd_1.cs_1.cd_0.c$			
+ SPANS				
$d_0.cw_lw_r$	$s_0.cw_lw_r$	$d_0.cw_ls_0.w_r$	$d_0.cw_rs_0.w_l$	$d_0.w_lw_rb_0.w$
$d_0.w_lw_rb_1.w$	$d_0.cw_rs_0.w_{lo}$	$d_0.ct_lw_r$	$d_0.cw_lt_r$	$d_0.ct_lt_r$
$s_0.ct_lw_r$	$s_0.cw_lt_r$	$s_0.ct_lt_r$	$d_0.ct_ls_0.w_r$	$d_0.cw_ls_0.t_r$
$d_0.ct_ls_0.t_r$	$d_0.ct_rs_0.w_l$	$d_0.cw_rs_0.t_l$	$d_0.ct_rs_0.t_l$	$d_0.w_lw_rb_0.t$
$d_0.w_lw_rb_1.t$	$d_0.cw_{lo}$	$d_0.ct_{lo}$	$s_0.cw_{ro}$	$s_0.ct_{ro}$

Table 7.4: Feature templates. s, d and b refer respectively to the data structures (S, D, B) presented in Section 7.2. The integers are indices on these data structures. $l$eft and $r$ight refer to the children of nodes. We use c, w and t to denote a node's label, its head and the part-of-speech tag of its head. When used as a subscript, l (r) refers to the left (right) index of a node. Finally lo (ro) denotes the token immediately left to the left index (right to the right index). See Figure 7.5 for a representation of a configuration with these notations.

Beam size	TIGERHN8		TIGERM15	
	F1	Disc. F1	F1	Disc. F1
2	81.86	48.49	84.28	49.04
4	83.27	53.00	85.43	53.14
8	83.61	54.42	85.93	55.00
16	83.84	54.81	86.13	56.17
32	84.32	56.22	86.10	55.50
64	84.14	56.01	86.30	56.90
128	84.05	55.76	86.13	57.04

Table 7.5: Results on development sets for different beam sizes (+SPANS feature set).

7.3.3.1 Internal Comparisons

We present parsing results with different beam sizes in Table 7.5. As expected, performance improves with the beam size until a ceiling is reached with a beam size of 32. An interesting result is that a larger beam gives small improvements for the F-measure ($+1$ from 4 to 32 on TIGERHN8) but substantial gains for the discontinuous F-measure ($+3.2$). We speculate that non-local information is very important for accurate parsing, and that keeping more hypotheses in the beam may compensate partially for the lack of global information. In fact, we will show in Chapter 8 that scoring models based on a bi-LSTM encoder, as presented in Chapter 4, may perform much better than a structured perceptron, even with greedy decoding.

We compare results with the three feature sets in Table 7.6. We observe that across the board, the +EXTENDED set improves over the BASELINE and that +SPANS yields another improvement. The improvements are larger for the discontinuous F1 metric. Both information about the gapped elements and about constituent boundaries are very useful for parsing discontinuous trees. We can perhaps expect further improvements by looking at elements further in S, D and B to capture even more extended context. But given that the number of templates is already high and that finding the right combination of bigram and trigram of features is rather costly, a much better strategy seems to shift to a neural model as that presented in Chapter 6.

Finally we compare SR-GAP to its variant SR-CGAP. The results of SR-CGAP either match or, more often, underperform those of SR-GAP. There

		NEGRA-30	NEGRA-ALL		TIGERHN08		TIGERM15
	Method	All	$L \leq 40$	All	$L \leq 40$	All	SPMRL / standard
Fernández-González and Martins (2015)	dep2const	82.56†	81.08	80.52	**85.53**	**84.22**	80.62 / -
Hall and Nivre (2008)	dep2const	-	-	-	79.93	-	-/-
van Cranenburgh (2012)	DOP	-	72.33	71.08	-	-	-/-
van Cranenburgh and Bod (2013)	DOP	-	76.8	-	-	-	-/-
Kallmeyer and Maier (2013)	LCFRS	75.75	-	-	-	-	-/-
Versley (2014b)	EAFI	-	-	-	74.23	-	-/-
Maier (2015) (baseline, b=(Ne=8/Ti=4))	SR-SWAP	75.17 (15.76)	-	-	-	-	-/-
Maier (2015) (best, b=(Ne=8/Ti=4))	SR-SWAP	76.95 (19.82)	-	-	79.52	-	- / 74.71 (18.77)
Maier and Lichte (2016) (best, b=4)	SR-SWAP				80.02		-/ 76.46 (16.31)
This work, beam=4		F1 (Disc. F1)					
GAP, BASELINE	SR-GAP	79.31 (38.66)	79.29 (39.78)	78.53 (38.64)	82.84 (47.13)	81.67 (44.83)	78.77 / 78.86 (41.36)
GAP, +EXTENDED	SR-GAP	80.44 (41.13)	80.34 (43.42)	79.79 (43.56)	83.57 (50.91)	82.43 (48.81)	79.42 / 79.51 (43.76)
GAP, +SPANS	SR-GAP	81.64 (42.94)	81.70 (47.17)	81.28 (46.85)	84.40 (51.98)	83.16 (49.76)	80.30 / 80.40 (46.50)
CGAP, BASELINE	SR-CGAP	79.61 (41.06)	79.32 (43.49)	78.64 (42.13)	82.90 (47.86)	81.68 (45.55)	78.32 / 78.41 (39.99)
CGAP, +EXTENDED	SR-CGAP	80.26 (40.52)	80.48 (43.42)	79.98 (42.60)	83.23 (50.57)	82.00 (48.28)	79.32 / 79.42 (44.66)
CGAP, +SPANS	SR-CGAP	81.16 (42.39)	81.41 (44.73)	80.89 (44.13)	83.92 (50.83)	82.79 (48.84)	80.38 / 80.48 (46.17)
This work, beam=32		F1 (Disc. F1)					
GAP, BASELINE	SR-GAP	80.57 (42.16)	80.20 (43.87)	79.75 (42.80)	83.53 (51.91)	82.41 (49.63)	79.60 / 79.69 (44.77)
GAP, +EXTENDED	SR-GAP	81.61 (45.75)	81.13 (47.52)	80.54 (46.89)	84.33 (53.84)	83.17 (51.88)	80.50 / 80.59 (46.45)
GAP, +SPANS	SR-GAP	**82.46** (**47.35**)	**82.76** (**51.82**)	**82.16** (**50.00**)	85.11 (**55.99**)	84.01 (**54.26**)	**81.50** / **81.60** (**49.17**)

Table 7.6: Final test results (gold POS tags). For TIGERM15, we report metrics computed with the SPMRL shared task parameters (see Section 7.3.3), as well as the standard parameters. †Trained on NEGRA-ALL.

are two likely explanations for these results. First, the padding method of Zhu et al. (2013) might be already very effective to handle derivation length biases. Secondly, we speculate that compound GAP actions are hard to predict when the parser has a bounded view on S.

7.3.3.2 External Comparisons

We now compare our models with other approaches, namely chart parsers based on explicit grammars (van Cranenburgh, 2012; van Cranenburgh and Bod, 2013; van Cranenburgh et al., 2016; Kallmeyer and Maier, 2013), parsers based on dependency parsing and reversible tree conversions (Hall and Nivre, 2008; Fernández-González and Martins, 2015) and transition-based parsers (Versley, 2014b; Maier, 2015; Maier and Lichte, 2016). Results with gold POS tags are presented in Table 7.6.[5] We also give results with predicted POS tags in Table 7.7.

Our best model outperforms the state of the art (Fernández-González and Martins, 2015) in every setting except one. When we compare our model to the previous best transition-based results (Maier, 2015) in the same experimental conditions (beam size of 4), we observe that SR-GAP has much better results (+4) and is noticeably more than twice as accurate

[5]New results have been published since the work presented in this chapter has been done (Corro et al., 2017; **?**). We omit them here, but include them in the experiment section of Chapter 8.

TIGERM15	F1 (spmrl.prm)	
	≤ 70	All
Versley (2014b)	73.90	-
Fernández-González and Martins (2015)	77.72	77.32
SR-GAP, beam=32, +SPANS	**79.44**	**79.26**

Table 7.7: Final results with predicted POS tags on the SPMRL split.

	Dev			Test			Sentence Lengths
	F1	Disc. F1	POS	F1	Disc. F1	POS	
SR-GAP, +SPANS, beam=32	90.92	73.35	100	89.91	70.25	100	All
Corro et al. (2017)				90.09		100	All
Evang and Kallmeyer (2011)	-	-	-	79.0	-	100	< 25
van Cranenburgh et al. (2016)	86.9	-	96.1	87.0	-	96.7	≤ 40

Table 7.8: Parsing results on the discontinuous Penn Treebank.

on discontinuous constituents. This result suggests that the representation of a parsing configuration by three data structures is very well suited to handle discontinuities.

Finally, we ran our model on the discontinuous version of the Penn Treebank (Evang and Kallmeyer, 2011), in the best experimental setting (+SPANS, beam=32). The results are presented in Table 7.8. Other published results are not all comparable to ours, due to different experimental settings (gold or predicted POS tags, restriction on the length of sentences), except that of Corro et al. (2017). Our model almost matches the parser of Corro et al. (2017) who use a more complex scoring system (bi-LSTM). We will give more thorough evaluation on this dataset in Chapter 8.

Model Analysis A constant in every experiment we have done is that there is usually a discrepancy between the recall and the precision of discontinuous constituents. We report detailed results in Table 7.9. In each setting, precision is more than 10 points above recall (almost 20 for English).[6] Our parser is rather conservative in the prediction of discontinuous constituents. We hypothesize that this discrepancy might be due to the rarity of discontinuous constituents in treebanks (and of the GAP action in the training data).

[6]This observation also holds for experiments with predicted tags (not reported here).

Dataset	Recall	Precision	F1	Δ Prec. / Rec.
NEGRA-30	41.5	55.2	47.4	13.7
NEGRA-ALL	43.2	59.4	50.0	16.2
TIGERHN8	49.1	60.6	54.3	11.5
TIGERM15	42.7	58.0	49.2	15.7
DPTB	64.8	84.5	73.4	19.7

Table 7.9: Detailed results for discontinuous constituents (+SPANS, SR-GAP, beam=32, gold POS tag setting).

7.4 Discussion: a Comparison of SR-SWAP and SR-GAP

In order to gain more insights into the factors explaining SR-GAP performance in comparison with SR-SWAP, we investigate two key differences between both algorithms: (i) for the same tree, SR-GAP tends to produce a much shorter derivation than SR-SWAP, (ii) SR-GAP configurations have arguably better access to relevant features.

7.4.1 Derivation Length

Generally, deriving a tree with SR-GAP will be less costly (in terms of the number of actions) than with SR-SWAP. To derive discontinuous trees, both systems implicitly predict an order on terminals with which the tree would be projective. An example of such an order is given in Figure 7.6. However, reordering is more efficient with SR-GAP which swaps whole subtrees, whereas the SWAP action only targets terminals.

Moreover, each time a SWAP is performed, the swapped terminal needs to be shifted again. As an illustration, we present a full derivation of the tree in Figure 7.2 with SR-SWAP in Table 7.10: the tokens *they* and *do* are swapped twice, and shifted three times during the derivation. 23 transitions are necessary to derive the tree whereas only 18 are needed with SR-GAP.

Let us consider first the longest possible derivation for a sentence of length n. With SR-GAP, there are exactly n shifts and $n-1$ binary reductions in a derivation (we ignore unary reductions temporarily). The longest derivation maximizes the number of GAP actions, by performing as many GAP actions as possible before each binary reductions. When S

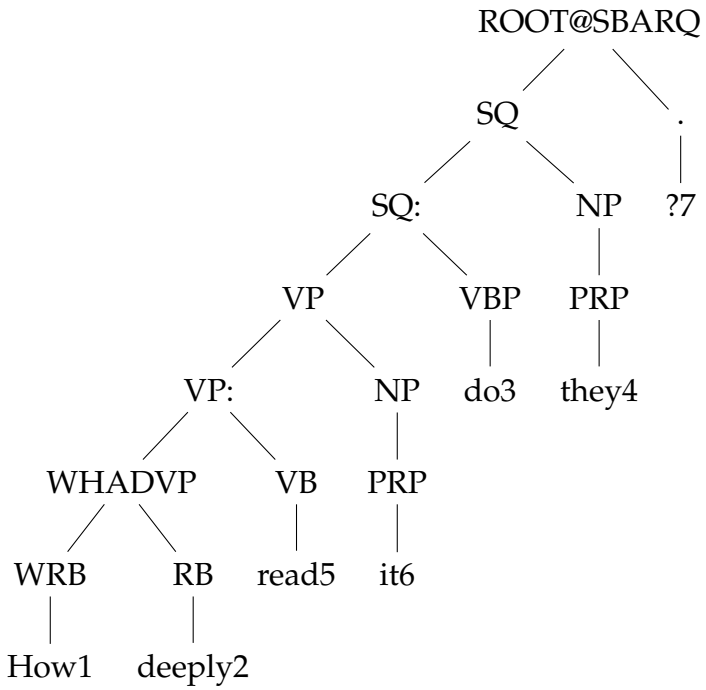


Figure 7.6: Projective tree obtained from the tree in Figure 7.2 by reordering terminals.

contains k elements, there are $k-1$ possible consecutive GAP actions. So the longest derivation starts by n SHIFTS, followed by $n-2$ GAPS, one binary reduction, $n-3$ GAPS, one binary reduction, and so on:

$$L_{\text{SR-GAP}}(n) = n + ((n-2)+1) + \cdots + 1 \tag{7.1}$$
$$= 1 + 2 + \cdots + n \tag{7.2}$$
$$= \frac{n(n+1)}{2} \tag{7.3}$$

The tree on the left-hand side of Figure 7.7 is derived in 15 steps ($\frac{5\cdot 6}{2}$). Since the parser may also perform up to n unary reductions, the longest possible derivation for a sentence of length n has $\frac{n(n+3)}{2}$ steps.

With SR-SWAP, the longest derivation for a sentence of length n maximizes the number of SWAPS. This derivation can be constructed as follows. After the first SHIFT, the parser performs repeatedly:

- $n-i$ SHIFTS (where i is the total number of tokens in the stack),
- $n-i-1$ SWAPS,

Transitions	Configurations S	B
	How deeply do they read it ?	
SHIFT $\Longrightarrow$	WRB[How]	deeply do they read it ?
SHIFT $\Longrightarrow$	WRB[How] RB[deeply]	do they read it ?
RL-WHADVP $\Longrightarrow$	WHADVP[How]	do they read it ?
SHIFT $\Longrightarrow$	WHADVP[How] VBP[do]	they read it ?
SHIFT $\Longrightarrow$	WHADVP[How] VBP[do] PRP[they]	read it ?
SHIFT $\Longrightarrow$	WHADVP[How] VBP[do] PRP[they] VB[read]	it ?
SWAP $\Longrightarrow$	WHADVP[How] VBP[do] VB[read]	they it ?
SWAP $\Longrightarrow$	WHADVP[How] VB[read]	do they it ?
RR-VP: $\Longrightarrow$	VP:[read]	do they it ?
SHIFT $\Longrightarrow$	VP:[read] VBP[do]	they it ?
SHIFT $\Longrightarrow$	VP:[read] VBP[do] PRP[they]	it ?
SHIFT $\Longrightarrow$	VP:[read] VBP[do] PRP[they] PRP[it]	?
RU-NP $\Longrightarrow$	VP:[read] VBP[do] PRP[they] NP[it]	?
SWAP $\Longrightarrow$	VP:[read] VBP[do] NP[it]	they ?
SWAP $\Longrightarrow$	VP:[read] NP[it]	do they ?
RL-VP $\Longrightarrow$	VP[read]	do they ?
SHIFT $\Longrightarrow$	VP[read] VBP[do]	they ?
RR-SQ: $\Longrightarrow$	SQ:[do]	they ?
SHIFT $\Longrightarrow$	SQ:[do] PRP[they]	?
RU-NP $\Longrightarrow$	SQ:[do] NP[they]	?
RL-SQ $\Longrightarrow$	SQ[do]	?
SHIFT $\Longrightarrow$	SQ[do] .[?]	
RL-ROOT@SBARQ $\Longrightarrow$	ROOT@SBARQ[do]	

Table 7.10: Derivation for the tree in Figure 7.2 with SR-SWAP (Maier, 2015).

- 1 binary reduction.

In such a derivation, the number of steps is:

$$L_{\text{SR-SWAP}}(n) = 1 + \sum_{i=1}^{n-1} ((n-i) + (n-i-1) + 1) \tag{7.4}$$

$$= 1 + 2\sum_{i=1}^{n-1} (n-i) \tag{7.5}$$

$$= 1 + 2\frac{n(n-1)}{2} \tag{7.6}$$

$$= n^2 - n + 1 \tag{7.7}$$

The tree on the right-hand side of Figure 7.7 can be derived in $5^2-5+1 = 21$ steps, using this strategy. Finally, if we take into account unary reductions, the maximum length of a derivation for a sentence of size n is $n^2 + 1$.

In the worst case, SR-GAP is asymptotically twice as economical as SR-SWAP ($\frac{n(n+3)}{2}$ vs $n^2 + 1$). To see if the difference in derivation lengths is confirmed empirically, we computed some statistics on the Tiger corpus

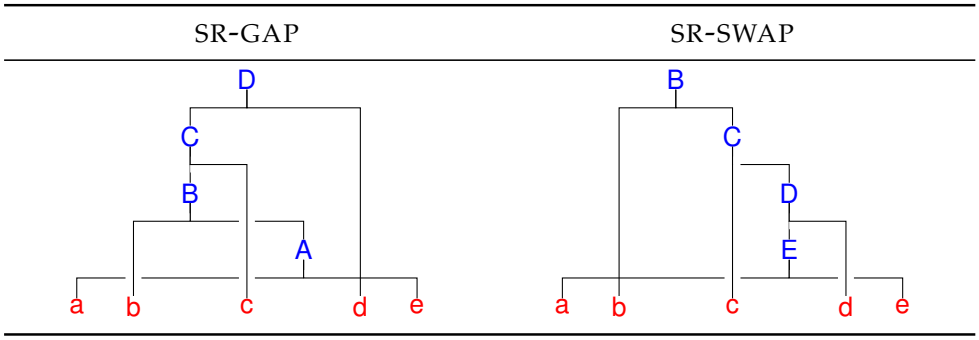


Figure 7.7: Example trees corresponding to the longest derivation for a sentence of length 5 with, respectively, SR-GAP and SR-SWAP.

	TIGERSPMRL (train)			DPTB (train)		
	SR-GAP	SR-SWAP	SR-CSWAP	SR-GAP	SR-SWAP	SR-CSWAP
Theoretical longest derivation	$\frac{n^2+3n}{2}$	n^2+1				
Longest derivation	264	2,081	1,163	294	1,459	835
Total number of gaps/swaps	40,905	361,577	105,393	33,341	215,793	55,609
Max consecutive gaps/swaps	10	68	1	9	30	1
Average derivation length wrt n	$2.0n$	$3.0n$	$2.6n$	$2.1n$	$2.5n$	$2.3n$

Table 7.11: Statistics about derivations on the training sets of the Tiger and DPTB corpora. n is the length of a sentence. SR-CSWAP is a variant of SR-SWAP introduced by Maier (2015).

and the Discontinuous Penn Treebank. We present them in Table 7.11. On average, $2.0 \cdot n$ actions are necessary for SR-GAP to derive the gold tree for a sentence of length n from the Tiger corpus (training set), and $3.0 \cdot n$ actions to derive the same tree with SR-SWAP.[7] As shorter derivations are arguably less prone to error propagation, we think that derivations produced by SR-GAP are easier to learn.

7.4.2 Feature Semantics and Feature Locality

In SR-SWAP the buffer stores both unprocessed terminals and terminals that have been swapped. Feature functions on B are ambiguous because they do not distinguish the two situations whereas this cue seems important. In contrast, with SR-GAP, B only stores unprocessed terminals and

[7] As Maier and Lichte (2016) points out, better oracles than that presented in Maier (2015) might produce shorter derivations.

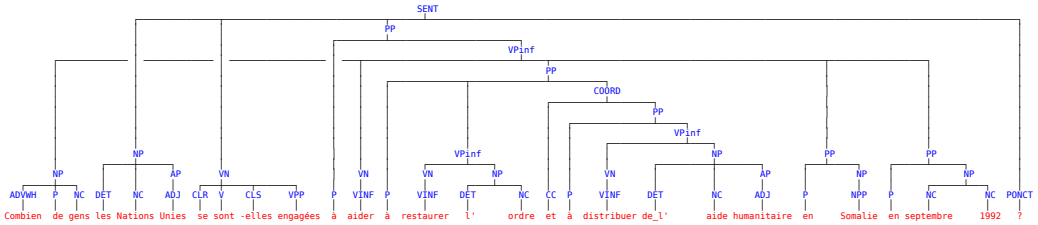


Figure 7.8: Tree from the French Question Bank (Seddah and Candito, 2016). *How many people has the UN committed itself to help restore order and to deliver humanitarian aid in Somalia in September 1992?*

the gapped elements are in D: there is no such ambiguity and thus the semantics of features is arguably better defined.

The second advantage of SR-GAP over SR-SWAP is that it has a better access to relevant information when making decisions. Thanks to the use of three data structures to define parsing configurations, SR-GAP has a larger domain of locality. Moreover, as SR-SWAP only allows terminals to be swapped, it must delay the construction of certain constituents that are inside a gap, leading to very long sequences of consecutive SHIFTS followed by consecutive SWAPS (Table 7.11).

For example, for the tree in Figure 7.8, the oracle of SR-SWAP (Maier, 2015) delays the construction of the NP *les Nations Unies* and that of the VN *se sont-elles engagées* until the full VPinf has been constructed.

Let us compare the configurations SR-GAP and SR-SWAP are in when they need to predict the first GAP or SWAP action when deriving the tree in Figure 7.8. The parser needs to make the NP *Combien de gens* and the verb *aider* contiguous to reduce them together.

The SR-GAP system is in the following configuration:

SR-GAP						
			S	D	B	
s_3	s_2	s_1	s_0	d_0	b_0	…
NP	NP	VN	à	aider	…	
Combien de gens	les Nations Unies	se sont-elles engagées				

It can access the NP *Combien de gens* with three consecutive GAP and then perform a reduction. The NP is close enough to the top of S to be accessed by the feature extractor (the templates we used include features on s_3). In contrast, SR-SWAP is in the following situation:

SR-SWAP											
									S	B	
s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0	b_0	...
NP	les	Nations	Unies	se	sont	-elles	engagées	à	aider	à ...	
Combien de gens											

It has to perform eight SWAPS before the reduction (and to repeat sequences of SHIFTS and SWAPS to get the terminals dominated by the VPinf node to be moved before the string *les Nations Unies se sont-elles engagées à*). Moreover, when the parser must predict the first SWAPS, it has only a limited view on the stack: feature templates are rarely defined on elements in the stack further than s_4. Therefore, the information that there is a potential object NP for *aider* in the stack is inaccessible.

In this type of situation, a sequence of related decisions (perform nine SHIFTS then eight SWAPS then REDUCE) happens over time when going through completely different configurations, which makes SR-SWAP very prone to error propagation.

7.5 Conclusion

In this chapter, we have introduced a new transition system for discontinuous constituency parsing. We have shown that the SR-GAP algorithm has very desirable properties: it produces shorter derivations than previous approaches, it is based on three data structures which makes relevant information easily accessible for parsing, and it is efficient enough to scale to whole corpora. Finally, it obtained state-of-the-art accuracies on several treebanks.

A direct extension of the parsing algorithm presented in this chapter is to use a different scoring model, e.g. a neural statistical model like the one presented in Chapter 6, to predict jointly morphology and discontinuous trees. Following recent proposals in projective constituency parsing (Dyer et al., 2016; Kuncoro et al., 2017), the design of a generative model that handles discontinuities might also prove fruitful. Finally, a remaining question is to investigate the importance of lexicalization in SR-GAP, as unlexicalized transition-based models have achieved very high results recently in projective constituency parsing (Cross and Huang, 2016a), at odds with the tradition of lexicalization in transition-based constituency parsing (Sagae and Lavie, 2005; Zhu et al., 2013; Mi and Huang, 2015; Crabbé, 2015). We address this question in the next chapter.

Chapter 8

Unlexicalized Constituency Parsing

Contents

8.1 Lexicalization in Statistical Parsing **152**
8.1.1 Lexicalized Chart Parsers 153
8.1.2 Unlexicalized Chart Parsers 153
8.1.3 Unlexicalized Transition-based Parsers 154
8.1.4 Contributions 154
8.2 Structure-Label Transitions Sytems **155**
8.2.1 Merge-Label-Gap 155
8.2.2 Lexicalized Merge-Label-Gap 159
8.2.3 Properties 160
8.3 Experiments **166**
8.3.1 Statistical Model 167
8.3.2 Feature Templates 167
8.3.3 Data 168
8.3.4 Results 169
8.4 Error Analysis **174**
8.4.1 Methodology 174
8.4.2 Observations 176
8.4.3 Discussion 186
8.5 Dynamic Programming **186**

8.5.1 Equivalent Parsing States 187
8.5.2 Tree Structured Stack 187
8.5.3 Encoding a TSS in the Parsing States 189
8.5.4 Towards Exhaustive Search with Dynamic Programming 192
8.6 Conclusion **193**

This chapter investigates the issue of lexicalization in the context of projective and non-projective constituency parsing. We use the term **lexicalized** to refer to a transition system that explicitly models bilexical relations, and thus constructs a dependency tree together with a constituency tree. A system that is not lexicalized is **unlexicalized**.[1]

Throughout this dissertation, we have used transition systems that assign the lexical head of a new constituent (REDUCE-LEFT and REDUCE-RIGHT), and statistical models that use the head of a constituent as a cue to guide parsing decisions. In this chapter, we introduce an unlexicalized transition system for discontinuous constituency parsing and compare it with its lexicalized counterparts. The main result of this chapter is that unlexicalized models lead to better parsing results than lexicalized models.

After briefly reviewing the question of lexicalization in constituency parsing (Section 8.1), we introduce several transition systems for discontinuous parsing and investigate their relevant properties (Section 8.2). We evaluate them with both projective and discontinuous multilingual parsing experiments (Section 8.3). In Section 8.4, we provide an analysis of the mistakes of the best model on discontinuous constituents in English. Finally, in Section 8.5, we argue that an unlexicalized transition system based on the GAP action and on minimal feature templates is particularly well suited to dynamic programming decoding. We outline a dynamic programming decoding algorithm for the discontinuous case.

8.1 Lexicalization in Statistical Parsing

Traditionally, transition-based constituency parsers almost exclusively use lexicalized transition systems (Sagae and Lavie, 2005, 2006; Zhu et al.,

[1] *Unlexicalized* parsing should not be confused with *delexicalized* parsing (McDonald et al., 2011), a transfer method for dependency parsing.

2013; Zhang and Clark, 2009; Maier, 2015; Liu and Zhang, 2017b). In contrast, in chart parsing, both lexicalized models (Collins, 1997) and unlexicalized models (Klein and Manning, 2003; Petrov et al., 2006) have been successful. We review them in the following paragraphs.

8.1.1 Lexicalized Chart Parsers

Parsing with lexicalized PCFG was popularized by Collins (1997). In order to limit the effect of data sparsity, Collins (1997) decomposes the probability of lexicalized grammar rules as products of factors that include explicitly the probability of a bilexical dependency between the heads of the nonterminals in the right-hand side of the grammar rule.

However, Gildea (2001) and Bikel (2004) observed that Collins' model performance only decreased marginally without bilexical probabilities. In fact, due to the data sparsity, the estimated bilexical probability is only used in a very small proportion of candidate rules. This observation suggests that the strength of the model does not rely in an adequate model of bilexical attachments, but rather on other features, including backoff strategies to POS tags.

The results on the SPMRL datasets presented in Table 6.2 of Chapter 6 seem to confirm this observation in another setting, namely transition-based constituency parsing. Without any information about the morphological tags, the parser performs poorly. There is a 7 F1 difference between the TOK+MMT and the TOK models.

8.1.2 Unlexicalized Chart Parsers

Unlexicalized chart parsers are based on the idea that learning a distribution of bilexical dependencies is too hard given the data sparsity problem. Instead, they rely on structural information. Klein and Manning (2003) used a refined grammar, where symbols are annotated with vertical and horizontal context in order to capture linguistic regularities that are not explicit in the treebank grammar and alleviate the effects of strong independence assumptions in standard PCFG. For example, subject and object NPs have the same label but subject NPs are more likely to rewrite as a personal pronoun than object NPs. This distinction can be encoded by annotating symbols with their parent (NP^S vs NP^VP). These finer-grained distribution distinctions can be learned automatically (Matsuzaki et al., 2005; Petrov et al., 2006; Shindo et al., 2012). Other types of cues used by chart parsers include features about the boundaries of constituents (Hall

et al., 2014; Durrett and Klein, 2015; Stern et al., 2017), which have also been shown to improve lexicalized constituency parsing (Crabbé, 2015; Coavoux and Crabbé, 2017a).

8.1.3 Unlexicalized Transition-based Parsers

Recently, two unlexicalized transition systems were introduced for projective constituency parsing.

- Dyer et al. (2016) proposed a generative parsing model called Recurrent Neural Network Grammar (RNNG), and based on a transition system with a top-down parsing strategy and an RNN that computes compositional representations of constituents. Kuncoro et al. (2017) observed that RNNG implicitly learned some notion of headedness. The compositional representation computed for a constituent depends mostly on a single child constituent, or on several constituents in cases such as coordination. Kuncoro et al. (2017) concluded that their model captures a fuzzy notion of headedness where multiple-head constituents are possible.

- Cross and Huang (2016a) introduced an unlexicalized transition system that distinguishes two types of actions: the **structural actions** that are used to construct an unlabelled constituent tree, and the **labelling actions** that assign nonterminal labels to constituents. The parser alternates one structural action and one labelling action. Their model obtains state-of-the-art results on English and French (Cross and Huang, 2016a). These results may be explained by several factors: the properties of the transition system (that does not require binarization) and the scoring system based on a bi-LSTM with very few feature templates.

8.1.4 Contributions

This chapter makes several contributions. First of all, we introduce the first unlexicalized transition system for discontinuous constituency parsing (ML-GAP, Section 8.2.1) which extends the structure-label system of Cross and Huang (2016a). Secondly, we formulate a lexicalized version of ML-GAP (Section 8.2.2), that maintains the distinction between structure and label actions. Thirdly, we compare these transition systems in several experimental settings and show that lexicalization is not necessary to achieve very high results (Section 8.3). Finally, we provide an error analysis of the best model (Section 8.4).

Input	$(1, 2, 3, \ldots, n)$
Axiom	$\langle \epsilon, 0, \emptyset \rangle$
Goal	$\langle (0, n), n, C \rangle$
Structural actions	
SHIFT	$\dfrac{\langle S, i, C \rangle}{\langle S \vert (i, i+1), i+1, C \rangle}$
MERGE	$\dfrac{\langle S \vert (j, k) \vert (k, i), i, C \rangle}{\langle S \vert (j, i), i, C \rangle}$
Labelling actions	
LABEL-X	$\dfrac{\langle S \vert (j, i), i, C \rangle}{\langle S \vert (j, i), i, C \cup \{(X, j, i)\} \rangle}$
NO-LABEL	$\dfrac{\langle S, i, C \rangle}{\langle S, i, C \rangle}$

Table 8.1: A reformulation of the structure-label transition system of Cross and Huang (2016a).

8.2 Structure-Label Transitions Sytems

In this section, we introduce two transition systems for discontinuous constituency parsing: an unlexicalized transition system called merge-label-gap (ML-GAP, Section 8.2.1) and its lexicalized counterpart ML-GAP-LEX (Section 8.2.2). We describe their relevant properties in Section 8.2.3.

8.2.1 Merge-Label-Gap

The transition system proposed by Cross and Huang (2016a) is presented as a deduction system in Table 8.1. The tokens are identified by their indices in the sentence. A parsing configuration is a triple $\langle S, j, C \rangle$ where

S is a stack of integer couples (l_i, r_i) representing spans, j is the index of the first token in the buffer, and C is a set of constituents. As in Chapter 5, a constituent is defined as a triple (X, l, r) where X is a nonterminal, and (l, r) is its span.

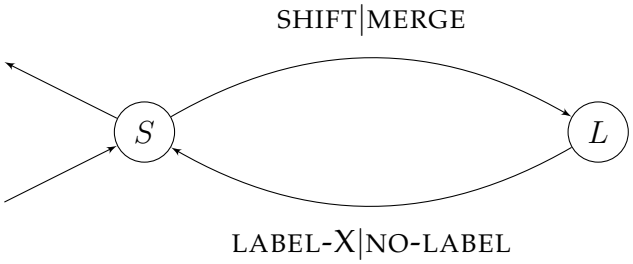
The parser alternates structural actions and labelling actions. At step n, if n is even, it must take a structural action, and a labelling action if n is odd. As a consequence, any derivation has length $4n - 2$, where n is the length of the sentence (n SHIFTS, $n - 1$ MERGES, $2n - 1$ LABEL-X and NO-LABEL). Any valid action sequence must be recognized by the automaton in Figure 8.1(a). Importantly, this transition system can derive directly n-ary trees and does not rely on binarization algorithms.

In order to generalize this transition system to the case of discontinuous constituency parsing, we need (i) to add a mechanism to predict discontinuous constituents and (ii) to adopt a definition of constituents that includes discontinuous constituents. For condition (i), we use the GAP action described in Chapter 7. For condition (ii), we define a constituent as a couple (X, s) where X is a nonterminal symbol and $s \in 2^{\mathbb{N}}$ is the set of indices of terminals dominated by the constituent. Alternatively, s could be defined as a range vector, i.e. a vector $\rho = \langle (l_1, r_1), \ldots, (l_n, r_n) \rangle$ where each couple (l_i, r_i) represents a span (Kallmeyer, 2010). Range vectors are used to describe instantiated rules in the LCFRS and RCG literature. We adopt a set definition instead to simplify the description of the transition system.

Like the SR-GAP transition system, ML-GAP is based on three data structures: a stack S, a deque D and a buffer B. We define a parsing configuration as a quadruple $\langle S, D, i, C \rangle$, where S and D are sequences of index sets, i is the index of the first token in the buffer, and C is a set of discontinuous constituents. The ML-GAP transition system is defined as a deduction system in Table 8.2. The available actions are defined as follows:

- The SHIFT action pushes the singleton $\{i + 1\}$ on D.
- The MERGE action pops I_{s_0} and I_{d_0}, the index sets at the top of S and D, computes their union $I = I_{s_0} \cup I_{d_0}$, flushes the content of D to S and pushes I onto D.
- The GAP action removes the top element from S and pushes it at the beginning of D, making the next element in S available for a MERGE operation.
- LABEL-X creates a new constituent labelled X whose yield is the set I_{d_0} at the top of D.

(a) MERGE-LABEL



(b) MERGE-LABEL-GAP

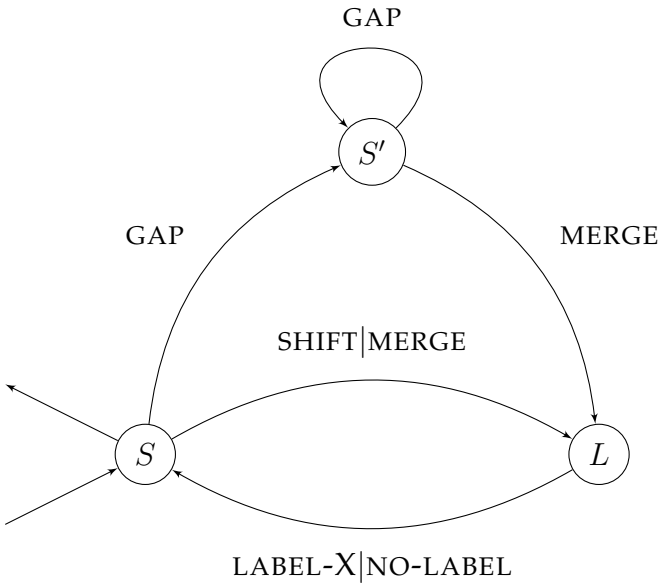


Figure 8.1: Action sequences allowed in transition systems. Any derivation must be recognized by the automaton.

- NO-LABEL has no effect.

As the semantics of the GAP action, a structural action, is not to modify I_{d_0}, but to make an index set in S available for a MERGE, it must not be followed by a labelling action. Any GAP must be followed either by another GAP or by a MERGE. We illustrate this constraint with an automaton in Figure 8.1(b). Any valid action sequence must be recognized

Input	$(1, 2, 3, \ldots, n)$
Axiom	$\langle \epsilon, \epsilon, 0, \emptyset \rangle$
Goal	$\langle \epsilon, \{1, 2, \ldots n\}, n, C \rangle$
Structural actions	
SHIFT	$\frac{\langle S, D, i, C \rangle}{\langle S \vert D, \{i+1\}, i+1, C \rangle}$
MERGE	$\frac{\langle S \vert I_{s_0}, D \vert I_{d_0}, i, C \rangle}{\langle S \vert D, I_{s_0} \cup I_{d_0}, i, C \rangle}$
GAP	$\frac{\langle S \vert I_{s_0}, D, i, C \rangle}{\langle S, I_{s_0} \vert D, i, C \rangle}$
Labelling actions	
LABEL-X	$\frac{\langle S, I_{d_0}, i, C \rangle}{\langle S, I_{d_0}, i, C \cup \{(X, I_{d_0})\} \rangle}$
NO-LABEL	$\frac{\langle S, I_{d_0}, i, C \rangle}{\langle S, I_{d_0}, i, C \rangle}$

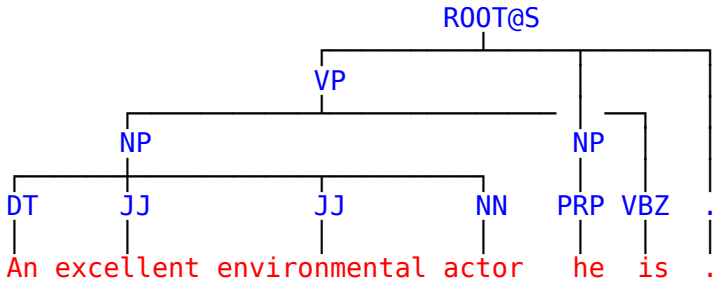
Table 8.2: The ML-GAP transition system, an unlexicalized transition system for discontinuous constituency parsing.

by this automaton.[2] When predicting a discontinuous constituent, the parser may perform several structural actions in a row, with the conditions that they constitute a sequence of GAP actions followed by a MERGE. We illustrate the transition system with a full derivation in Table 8.3.

[2]However the automata over-generate since all preconditions on actions are not taken into account.

8.2.2 Lexicalized Merge-Label-Gap

In order to assess the role of lexicalization in parsing in the context of structure-label transition systems, we introduce a hybrid transition system, ML-GAP-LEX, which is based on a distinction between structural and labelling actions, but is nonetheless lexicalized. An instantiated **lexicalized discontinuous constituent** is defined as a triple (X, I, h) where X is a nonterminal label, I is the set of terminals that are in the yield of the constituent, and $h \in I$ is the lexical head of the constituent.



Action	Stack	Deque	Configuration Buffer	Constituents
			0	{ }
SHIFT $\Longrightarrow$		{1}	1	{ }
NO-LABEL $\Longrightarrow$		{1}	1	{ }
SHIFT $\Longrightarrow$	{1}	{2}	2	{ }
NO-LABEL $\Longrightarrow$	{1}	{2}	2	{ }
MERGE $\Longrightarrow$		{1, 2}	2	{ }
NO-LABEL $\Longrightarrow$		{1, 2}	2	{ }
SHIFT $\Longrightarrow$	{1, 2}	{3}	3	{ }
NO-LABEL $\Longrightarrow$	{1, 2}	{3}	3	{ }
MERGE $\Longrightarrow$		{1, 2, 3}	3	{ }
NO-LABEL $\Longrightarrow$		{1, 2, 3}	3	{ }
SHIFT $\Longrightarrow$	{1, 2, 3}	{4}	4	{ }
NO-LABEL $\Longrightarrow$	{1, 2, 3}	{4}	4	{ }
MERGE $\Longrightarrow$		{1, 2, 3, 4}	4	{ }
LABEL-NP $\Longrightarrow$		{1, 2, 3, 4}	4	{(NP, {1,2,3,4})}
SHIFT $\Longrightarrow$	{1, 2, 3, 4}	{5}	5	{(NP, {1,2,3,4})}
LABEL-NP $\Longrightarrow$	{1, 2, 3, 4}	{5}	5	{(NP, {1,2,3,4}), (NP, {5})}
SHIFT $\Longrightarrow$	{1, 2, 3, 4}|{5}	{6}	6	{(NP, {1,2,3,4}), (NP, {5})}
NO-LABEL $\Longrightarrow$	{1, 2, 3, 4}|{5}	{6}	6	{(NP, {1,2,3,4}), (NP, {5})}
GAP $\Longrightarrow$	{1, 2, 3, 4}	{5} | {6}	6	{(NP, {1,2,3,4}), (NP, {5})}
MERGE $\Longrightarrow$	{5}	{1, 2, 3, 4, 6}	6	{(NP, {1,2,3,4}), (NP, {5})}
LABEL-VP $\Longrightarrow$	{5}	{1, 2, 3, 4, 6}	6	{(NP, {1,2,3,4}), (NP, {5}), (VP, {1,2,3,4,6})}
MERGE $\Longrightarrow$		{1, 2, 3, 4, 5, 6}	6	{(NP, {1,2,3,4}), (NP, {5}), (VP, {1,2,3,4,6})}
NO-LABEL $\Longrightarrow$		{1, 2, 3, 4, 5, 6}	6	{(NP, {1,2,3,4}), (NP, {5}), (VP, {1,2,3,4,6})}
SHIFT $\Longrightarrow$	{1, 2, 3, 4, 5, 6}	{7}	7	{(NP, {1,2,3,4}), (NP, {5}), (VP, {1,2,3,4,6})}
MERGE $\Longrightarrow$		{1, 2, 3, 4, 5, 6, 7}	7	{(NP, {1,2,3,4}), (NP, {5}), (VP, {1,2,3,4,6})}
LABEL-ROOT@S $\Longrightarrow$		{1, 2, 3, 4, 5, 6, 7}	7	{(NP, {1,2,3,4}), (NP, {5}), (VP, {1,2,3,4,6}), (ROOT@S, {1,2,3,4,5,6,7})}

Table 8.3: Example derivation for the tree above with the ML-GAP transition system.

In ML-GAP-LEX, a parsing configuration is a 5-tuple $\langle S, D, i, C, A\rangle$. S and D are sequences of couples (I, h), where I is a set of indices and $h \in I$ is a distinguished element of I. The integer i has the same semantics as in ML-GAP, C is a set of constituents and A is a set of dependency arcs. The ML-GAP-LEX transition system is presented in Table 8.4 as a deduction system. The main difference with ML-GAP is that there are two MERGE actions, MERGE-LEFT and MERGE-RIGHT, and that each of them creates a new directed dependency arc.

8.2.3 Properties

This section illustrates key differences between the transition systems we introduced and more standard transition systems, both in the case of discontinuous and projective constituency parsing. We first briefly describe the oracles we used, then we discuss several metrics computed on treebanks: the number of action types, the number of GAP actions and the incrementality of each transition system.

8.2.3.1 Oracles

The oracles we used are static oracles. For the ML-GAP-LEX transition system, the oracle is very similar to the SR-GAP oracle presented in Chapter 7. A derivation in ML-GAP-LEX can be straightforwardly computed from a derivation in SR-GAP transition system by the following operations:

- REDUCE-LEFT-X (resp. REDUCE-RIGHT-X) actions are replaced by a MERGE-LEFT (resp. MERGE-RIGHT) action followed by
  - LABEL-X if X is a non-temporary nonterminal
  - NO-LABEL otherwise
- The REDUCE-UNARY-X actions are replaced by LABEL-X.
- NO-LABEL actions are inserted to make sure the derivation satisfy the structure-label alternation.

This oracle attaches the left dependents of a governor first. In practice, other oracle strategies are possible as long as constituents are constructed from their head outward.

For the ML-GAP transition system, we use an oracle that implicitly corresponds to the left-first binarization (see Section 3.2.3.1): n-ary constituents are built in a left-to-right fashion.

Input	$(1, 2, 3, \ldots, n)$
Axiom	$\langle \epsilon, \epsilon, 0, \emptyset, \emptyset \rangle$
Goal	$\langle \epsilon, \{1, 2, \ldots n\}, n, C, A \rangle$
	Structural actions
SHIFT	$\frac{\langle S, D, i, C, A \rangle}{\langle S \vert D, (\{i+1\}, i+1), i+1, C, A \rangle}$
MERGE-LEFT	$\frac{\langle S \vert (I_{s_0}, h_{s_0}), D \vert (I_{d_0}, h_{d_0}), i, C, A \rangle}{\langle S \vert D, (I_{s_0} \cup I_{d_0}, h_{s_0}), i, C, A \cup \{h_{s_0} \rightarrow h_{d_0}\} \rangle}$
MERGE-RIGHT	$\frac{\langle S \vert (I_{s_0}, h_{s_0}), D \vert (I_{d_0}, h_{d_0}), i, C, A \rangle}{\langle S \vert D, (I_{s_0} \cup I_{d_0}, h_{d_0}), i, C, A \cup \{h_{d_0} \rightarrow h_{s_0}\} \rangle}$
GAP	$\frac{\langle S \vert I_{s_0}, D, i, C, A \rangle}{\langle S, I_{s_0} \vert D, i, C, A \rangle}$
	Labelling actions
LABEL-X	$\frac{\langle S, I_{d_0}, i, C, A \rangle}{\langle S, I_{d_0}, i, C \cup \{(X, I_{d_0})\}, A \rangle}$
NO-LABEL	$\frac{\langle S, I_{d_0}, i, C, A \rangle}{\langle S, I_{d_0}, i, C, A \rangle}$

Table 8.4: A lexicalized structure-label transition system for discontinuous parsing: ML-GAP-LEX.

Corpus (train sets)	Number of action types SR-GAP	ML-GAP-LEX	ML-GAP
SPMRL			
Arabic	195	101	100
Basque	88	32	31
French	181	95	94
German	923	580	579
Hebrew	286	202	201
Hungarian	79	44	43
Korean	45	19	18
Polish	157	94	93
Swedish	129	69	68
Discontinuous treebanks			
English (Disco PTB)	294	157	156
French (French Treebank)	190	95	94
German (Tiger)	191	94	93
German (Negra)	136	60	59

Table 8.5: Action type statistics per transition system and corpus.

8.2.3.2 Number of Action Types

The numbers of action types for each transition system and corpus are reported in Table 8.5. Structure-label systems have much fewer action types, at least 30% fewer actions (Hebrew) and at most 65% (Basque). There are two reasons for this. They have fewer nonterminal symbols because they do not need temporary symbols. They have a single labelling action LABEL-X for each nonterminal, where SR-GAP can have up to three actions (REDUCE-RIGHT-X, REDUCE-LEFT-X and REDUCE-UNARY-X) for the same nonterminal X.

Given the reduced number of action types and the distinction between structural and labelling actions, structure-label systems have arguably easier decisions to make. On the other hand, derivations are longer with structure-label systems. Indeed, in shift-reduce systems, REDUCE actions perform both a structural operation and a labelling operation in a single parsing step. Assuming there is a limited number of unary constituents and discontinuous constituents in a corpus, the length of a derivation for

	SR-GAP	ML-GAP-LEX	ML-GAP
English (Disco PTB)			
Max number of consecutive GAPS	9	9	8
Average number of consecutive GAPS	1.78	1.78	1.34
Total number of GAPS	33,341	33,341	18,421
French (French Treebank)			
Max number of consecutive GAPS	7	7	6
Average number of consecutive GAPS	1.57	1.57	1.33
Total number of GAPS	578	578	408
German (Tiger)			
Max number of consecutive GAPS	10	10	5
Average number of consecutive GAPS	1.40	1.40	1.12
Total number of GAPS	40,905	40,905	25,852
German (Negra)			
Max number of consecutive GAPS	11	11	5
Average number of consecutive GAPS	1.47	1.47	1.11
Total number of GAPS	20,149	20,149	11,181

Table 8.6: GAP action statistics in the training sets of different treebanks.

a sentence size n is around $4n$ for a structure-label system,[3] and $2n$ for a shift-reduce system.[4]

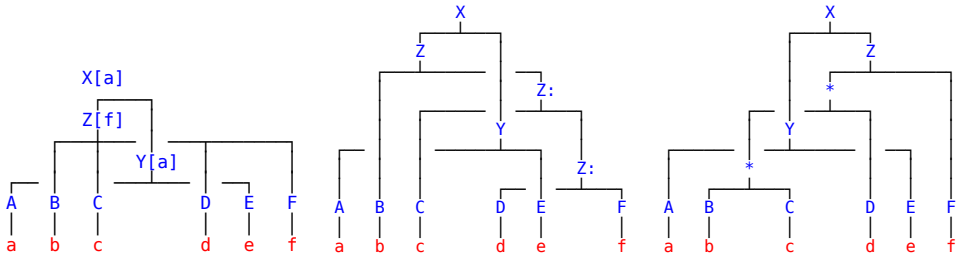
8.2.3.3 Number of GAP Actions

The GAP actions are supposedly more difficult to predict, because they involve long distance information. They also increase the length of a derivation and make the parser more prone to error propagation. We expect a transition system that is able to predict a discontinuous tree more efficiently, in terms of number of GAP actions, to be a better choice.

We report in Table 8.6 the number of GAP actions necessary to derive the discontinuous trees for several corpora and for several transition systems. We also report the average and maximum number of consecutive

[3] n SHIFTS, $n-1$ MERGES, $2n-1$ LABEL-X and NO-LABEL.

[4] n SHIFTS, $n-1$ REDUCES.



- SR-GAP: SH, SH, SH, SH, GAP, GAP, GAP, RR-Y, SH, GAP, RR-Z:, GAP, RR-Z:, GAP, RR-Z, RL-X
- ML-GAP: SH, SH, SH, MERGE, SH, MERGE, SH, GAP, MERGE, LABEL-Y, SH, GAP, MERGE, LABEL-Z, MERGE, LABEL-X

Figure 8.2: A lexicalized tree and the binarizations respectively used by a lexicalized system (center part) and an unlexicalized system (right part), along with the corresponding derivations (NO-LABEL actions are ignored).

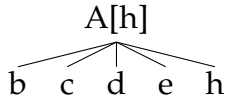
GAP actions in each case. For English and German, the unlexicalized transition system ML-GAP needs much fewer GAP actions to derive discontinuous trees (approximately 45% fewer). The average number of consecutive GAP actions is also smaller (as well as the maximum for German corpora). In average, the elements in the stack (S) that need to combine with the top of the deque (D) are closer to the top of S with the ML-GAP transition system than with lexicalized systems. This observation is not surprising, since ML-GAP can start constructing constituents before having access to their lexical head, it can construct larger structures before having to GAP them.

We illustrate this difference between oracles with an artificial example in Figure 8.2. We present a lexicalized tree (left-hand part), with its binarization used by lexicalized transition systems (center part), and the implicit binarization used by ML-GAP (right-hand part). In the latter tree, non-constituents are indicated with a star, corresponding to NO-LABEL actions. We also report the corresponding derivations for SR-GAP and ML-GAP. The SR-GAP transition system must construct constituent Y first and need to perform three GAP actions to do so, as terminals *b*, *c* and *d* must be combined with *f*, which has not been shifted yet. In contrast, the ML-GAP can start constructing Z by combining b, c and d and needs a single GAP to build Y. Such structures explain the difference in average

and maximum number of GAP actions for lexicalized and non-lexicalized transition systems.

8.2.3.4 Incrementality

Finally, we compare the incrementality of the three transition systems. We adopt the definition of incrementality of Nivre (2004): an incremental algorithm minimizes the number of connected components in the stack during parsing. An unlexicalized system can construct a new constituent by incorporating each new component immediately whereas a lexicalized system waits until it has shifted the head of a constituent before starting building the constituent. For example, to construct the following subtree,



a lexicalized system must shift every token before starting reductions in order to be able to predict the dependency arcs between h and its dependents.[5] In contrast, an unlexicalized system can construct partial structures as soon as there are two elements with the same parent node in the stack.[6]

We report the average number of connected components during a derivation for each transition system in Table 8.7. Overall, the unlexicalized transition system ML-GAP is more incremental than lexicalized transition systems, except for the Korean treebank. An explanation of this is that in the Korean treebank, nearly all constituents are binary. The three transition systems differ in their strategy to construct n-ary constituents but follow the same single possible strategy for binary constituents.

8.2.3.5 Interim Conclusion

In this section, we have discussed differences between three transition systems in different settings (discontinuous or projective treebanks). The ML-GAP transition system has much fewer action types than SR-GAP and is more incremental, producing simpler configurations. Finally, ML-GAP-LEX is a hybrid system with properties from both other systems: it yields derivations similar to those of SR-GAP whereas it has much fewer action types. The goal of the parsing experiments of the next section with

[5] SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, REDUCE-RIGHT-A:, REDUCE-RIGHT-A:, REDUCE-RIGHT-A:, REDUCE-RIGHT-A.

[6] SHIFT, SHIFT, MERGE, SHIFT, MERGE, SHIFT, MERGE, SHIFT, MERGE.

Projective corpora (SPMRL)	Average length of stack		
	SR	ML-LEX	ML
Arabic	6.57	6.48	6.24
Basque	2.57	2.58	2.16
French	4.41	4.39	4.13
German	3.14	3.19	2.33
Hebrew	4.85	4.73	4.53
Hungarian	3.05	3.01	2.42
Korean	2.76	2.84	2.79
Polish	2.87	2.77	2.59
Swedish	3.38	3.37	3.23
Discontinuous corpora	Average length of stack (S+D)		
	SR-GAP	ML-GAP-LEX	ML-GAP
English (Disco-PTB)	5.57	5.62	4.86
French (Disco-FTB)	4.41	4.39	4.13
German (Negra)	3.66	3.69	2.88
German (Tiger)	3.53	3.56	2.98

Table 8.7: Incrementality measure per transition system and corpus, measured by the average size of the stack during derivations. The average is calculated across all configurations (not across all sentences).

these three transition systems is to assess the contributions of different properties of ML-GAP.

8.3 Experiments

The experiments we have carried out aim at assessing the importance of lexicalization in the context of both projective and discontinuous constituency parsing. To this end, we compare the three transition systems we have presented in this chapter (and their projective counterparts), as well as several feature templates. As a side empirical question, we evaluate the role of nonterminal feature templates.

8.3.1 Statistical Model

The base statistical model is nearly identical to the TOK+CLSTM+M+D multitask model of Chapter 6. A hierarchical bi-LSTM builds character-aware word embeddings and uses them to construct context-aware representations for each token. A tagger uses each token representation to predict its POS tag, its dependency label and its morphological attributes. Then, at each parsing step, a feed-forward classifier extracts lexical and nonterminal symbols from the configuration to predict what transition should be taken next.

There are two differences with the TOK+CLSTM+M+D model of Chapter 6. First of all, we experiment with different feature templates (see next section). Secondly, following Cross and Huang (2016a), for the structure-label systems, we use two distinct classifiers for the structural actions and the labelling actions.

Let $\mathbf{h}^{(0)}$ be the input of an action classifier, i.e. the concatenation of symbol representations extracted from the configuration. The parser will use different sets of parameters depending on the type of the next action:

$$\begin{array}{ll} \text{Structure} & \text{Label} \end{array} \tag{8.1}$$

$$\mathbf{h}^{(1)} = f(\mathbf{W}^{(s^1)} \cdot \mathbf{h}^{(0)} + \mathbf{b}^{(s^1)}) \qquad \mathbf{h}^{(1)} = f(\mathbf{W}^{(l^1)} \cdot \mathbf{h}^{(0)} + \mathbf{b}^{(l^1)}) \tag{8.2}$$

$$\mathbf{h}^{(2)} = f(\mathbf{W}^{(s^2)} \cdot \mathbf{h}^{(1)} + \mathbf{b}^{(s^1)}) \qquad \mathbf{h}^{(2)} = f(\mathbf{W}^{(l^2)} \cdot \mathbf{h}^{(1)} + \mathbf{b}^{(l^1)}) \tag{8.3}$$

$$\mathbf{h}^{(3)} = \text{Softmax}(\mathbf{W}^{(s^3)} \cdot \mathbf{h}^{(2)} + \mathbf{b}^{(s^2)}) \qquad \mathbf{h}^{(3)} = \text{Softmax}(\mathbf{W}^{(l^3)} \cdot \mathbf{h}^{(2)} + \mathbf{b}^{(l^2)}) \tag{8.4}$$

where $\mathbf{W}^{(s^i)}$ and $\mathbf{b}^{(s^i)}$ are the parameters used to predict a structural action, whereas $\mathbf{W}^{(l^i)}$ and $\mathbf{b}^{(l^i)}$ are used for a label action. Thanks to the distinction between labelling and structural actions, parsing can be viewed as two subtasks. The use of two classifiers is related to the question of what parameters need to be shared in a multitask setting: either just the hierarchical bi-LSTM, or also the hidden layers of the action classifier.

8.3.2 Feature Templates

We experimented with four sets of feature templates. They are specified in Table 8.8, using notations from Chapter 7.

- The BASE feature templates only includes span features: the tokens that are at the boundaries of constituents in the stack (and deque in the discontinuous case), as well as b_0 to represent the buffer. The

	Projective	Discontinuous
BASE	$b_0, s_0.w_l, s_0.w_r, s_1.w_l, s_1.w_r, s_2.w_r$	$b_0, d_0.w_l, d_0.w_r, d_1.w_r, s_0.w_l, s_0.w_r, s_1.w_r$
+LEX	BASE+ $s_0.w, s_1.w$	BASE+ $d_0.w, d_1.w, s_0.w, s_1.w$
+NT	BASE+ $s_0.c, s_1.c, s_2.c$	BASE+ $d_0.c, d_1.c, s_0.c, s_1.c$
+LEX+NT	BASE+ $s_0.w, s_1.w, s_0.c, s_1.c, s_2.c$	BASE+ $d_0.w, d_1.w, s_0.w, s_1.w, d_0.c, d_1.c, s_0.c, s_1.c$

Table 8.8: Feature template set descriptions.

BASE templates are meant to be minimal (6 or 7 templates vs 11 for the template set used in Chapter 6).

- The +LEX templates add information about the heads of constituents to assess whether these are useful for a lexicalized transition system.
- The +NT templates add features about nonterminal labels.
- Finally the +LEX+NT templates include every template in the BASE, +LEX and +NT sets.

The absence of nonterminal symbol features in the BASE template set might seem odd as these features encode grammar rules that are supposedly important for accurate constituency parsing. However, Cross and Huang (2016a) showed that they were not necessary to achieve high accuracy in projective constituency parsing on the Penn Treebank and the French Treebank. A possible explanation is that nonterminal features may be inaccurate due to labelling errors.

On the training set, tagging accuracy reaches quickly 99%: the parser will have access to near-gold features. Moreover, as it uses a static oracle, it will have access to gold nonterminal features. In contrast, on the development and test sets, tagging is not as accurate and the parser will make nonterminal labelling mistakes. Due to error propagation, the parser will not have access to gold features, but to noisy, possibly wrong, features. In other words, there is a mismatch between the training and the testing situations, which may harm learning. Removing nonterminal features is the easiest way to avoid this situation. Morphosyntactic information is still available in the representations computed by the sentence-level bi-LSTM as they are trained to be good predictors of POS tags, as in the stack-propagation architecture of Zhang and Weiss (2016).

8.3.3 Data

Discontinuous Treebanks For discontinuous parsing experiments, we used the Negra corpus (Skut et al., 1997), the Tiger corpus (Brants et al.,

2002), the discontinuous version of the Penn Treebank (Evang and Kallmeyer, 2011; Marcus et al., 1993) and a discontinuous version of the French Treebank (Coavoux and Crabbé, 2017; Abeillé et al., 2003).

For the Tiger corpus, we use the SPMRL split. We obtained the dependency labels and the morphological information for each token with the dependency treebank versions of the SPMRL release.

We converted the Negra corpus to labelled dependency trees with the DEPSY tool[7] in order to annotate each token with a dependency label. We do not predict morphological attributes for this corpus (only POS tags) as only a small section is annotated with a full morphological analysis. We use the standard split (Dubey and Keller, 2003) and no limit on sentence length.

We used the Stanford parser to convert the Penn Treebank to a labelled dependency corpus and annotate its tokens with dependency labels.

Following standard practice and to compare fairly with other published results, we ignore punctuation for the evaluation, except for French, for which we use the SPMRL evaluation parameters.

Projective Treebanks For projective parsing experiments, we used the SPMRL dataset (Seddah et al., 2013) in the same instantiation as in Chapter 6. We used the SPMRL shared task evaluator that takes punctuation and unparsed sentences into account.

8.3.4 Results

In each setting, the only hyperparameters that are tuned are the learning rate $\{0.01, 0.02\}$ and the number of iterations $\{4, 8, 12, \ldots, 28, 30\}$. They are tuned for the F1 measure on the development sets.[8] The other hyperparameters have the same values as in the experiments of Chapter 6.

8.3.4.1 Multilingual Discontinuous Constituency Parsing

Internal Comparisons Discontinuous parsing results on the development set are shown in Table 8.9. First of all, we observe that all models perform very well, which suggests that the differences between transition systems have rather marginal effects compared to the bi-LSTM statistical model.

[7] `https://nats-www.informatik.uni-hamburg.de/pub/CDG/DownloadPage/cdg-2006-06-21.tar.gz` We modified DEPSY to keep the same tokenization as the original corpus.

[8] In order to maximize performance on the discontinuous constituents, another possibility would have been to optimize them on the discontinuous F1 measure.

		English		German (Tiger)		German (Negra)		French	
Transition System	Features	F	Disc. F	F	Disc. F	F	Disc. F	F	Disc. F
SR-GAP	BASE	90.66	**72.12**	85.88	55.96	82.27	47.56	81.77	10.53
SR-GAP	+NT	90.85	71.76	87.10	58.09	83.24	**54.26**	81.88	18.60
SR-GAP	+LEX	90.80	70.94	86.24	55.86	81.85	47.51	82.10	7.27
SR-GAP	+NT+LEX	90.86	69.31	86.47	58.20	82.72	51.38	81.96	4.65
ML-GAP	BASE	**91.20**	72.00	**87.61**	60.48	**83.66**	53.78	**82.59**	22.22
ML-GAP	+NT	91.07	68.63	87.37	**60.95**	83.61	53.19	82.42	**27.27**
ML-GAP-LEX	BASE	91.06	68.24	86.48	56.31	82.36	46.95	81.89	15.38
ML-GAP-LEX	+NT	90.69	68.66	87.22	58.20	83.49	55.20	82.22	17.39
ML-GAP-LEX	+LEX	90.92	68.37	86.69	57.92	82.59	50.06	81.70	25.00
ML-GAP-LEX	+NT+LEX	91.04	71.09	87.33	60.53	82.76	51.74	81.78	20.00

Table 8.9: Discontinuous parsing results on the development sets.

The ML-GAP transition system performs consistently better for the F1 measure, and most of the time, also the discontinuous F1. We conclude from this result that lexicalization is not necessary to achieve very strong discontinuous parsing results.

As regards feature templates, the effect of nonterminal features is either slightly beneficial to parsing (SR-GAP) or slightly detrimental (ML-GAP). Finally, for lexicalized transition systems, the use of head information has in fact very little effect, and is even detrimental is some cases (SR-GAP+NT+LEX vs SR-GAP+NT on the Tiger Corpus). From these observations, we conclude that the simplest template set must be preferred, and focus now on the ML-GAP model with BASE features.

Morphological Analysis We report results for morphological analysis with the selected model (ML-GAP with BASE features) in Table 8.10. For each morphological attribute, we report an accuracy score computed over every tokens. However, most morphological attributes are only relevant for specific part-of-speech tags. For instance, TENSE is only a feature of verbs. The accuracy metric is somewhat misleading, since the fact that the tagger predicts correctly that a token does not have an attribute is considered as a correct answer. Therefore, if only 5% of tokens bore a specific morphological attribute, a 95% accuracy is a baseline score. For this reason, we also report a coverage metric (Cov.) that indicates the proportion of tokens in the corpus that possess an attribute, and an F1 measure that uses the following definitions to compute the precision and recall:

- True positives: the system predicts the correct attribute-value pair.

Attribute	Acc.	F1	Cov.
English			
POS	97.25	-	100
French			
POS	97.61	-	100
Complete match	89.84	-	100
Gender	97.31	97.36	50.32
Mood	99.54	97.88	10.89
Number	98.47	98.66	57.07
Person	99.67	98.36	9.95
Subcat	97.57	98.31	71.2
Tense	99.55	97.45	8.7
MWE	96.53	80.68	8.88
MWE-head	97.65	78.02	5.29

Attribute	Acc.	F1	Cov.
German (Negra)			
POS	97.93	-	100
German (Tiger)			
POS	98.37	-	100
Complete match	92.94	-	100
Case	96.87	96.85	48.24
Degree	99.7	98.04	7.53
Gender	96.89	96.83	47.73
Mood	99.85	99.07	7.82
Number	98.45	98.67	57.77
Person	99.9	99.46	9.45
Tense	99.89	99.27	7.83
German (Tiger) Björkelund et al. (2013)			
POS	98.10		
Complete match	91.80		

Table 8.10: Morphological analysis results on the development sets.

- True negatives: the system predicts correctly that an attribute is not relevant for a token, for instance that a noun has no tense.
- False positives: the system predicts an attribute-value pair:
  - for a token that has no such attribute (for example, prediction of a tense for a noun);
  - for a token that has such attribute but with another value (for example, `case=nom` instead of `case=acc`).
- False negatives: when the system fails to see that the token has the attribute.

In French, the tagger performs rather well except on the two attributes related to multiword expression (MWE) detection. The attribute `MWE` has a boolean value that specifies whether a token is part of an MWE and `MWE-head` is assigned to lexical heads of MWE and specifies the POS tag of the MWE. For example, in the compound *motion de censure* (*motion of censure*), the three tokens have the attribute-value pair `MWE=true` and *motion* has the additional pair `MWE-head=NC+`. The low performance on these attributes is likely due to the fact that MWE recognition is more

Transition System	Features	English F	English Disc. F	German (Tiger) F	German (Tiger) Disc. F	German (Negra) F	German (Negra) Disc. F	French F	French Disc. F
Predicted POS tags									
ML-GAP	BASE	**90.99**	**71.31**	**82.73**	**55.86**	**82.62**	**54.07**	**82.91**	**20.29**
Stanojević and Garrido Alhama (2017), SWAP*, bi-LSTM				76.96					
Coavoux and Crabbé (2017a)				79.26					
Corro et al. (2017)		89.17							
Fernández-González and Martins (2015)				77.32					
Versley (2016)				79.50					
van Cranenburgh et al. (2016), ≤ 40		87				74.8			
Gold POS tags									
Maier (2015), SR-SWAP, perceptron				74.71	18.77	76.95	19.82		
Stanojević and Garrido Alhama (2017), SWAP*, bi-LSTM				81.64		**82.87**			
Evang and Kallmeyer (2011)†, PLCFRS, < 25		79							

Table 8.11: Discontinuous parsing results on the test sets.
*Uses a variant of the lexicalized shift-reduce system (Cross and Huang, 2016b) with an additional SWAP action.
†Does not discount root symbols and punctuation.

difficult than the other auxiliary tasks, as it involves lexical knowledge that is hard to learn. Moreover, the representation of MWE with attributes is not ideal. It has to be noted that MWEs are also taken into account by parsing evaluation as any other constituent.

In German, the tagger also achieves high results, with slightly lower scores for case and gender. Overall, it slightly outperforms previous results published by Björkelund et al. (2013) who used the MARMOT tagger (Mueller et al., 2013).

External Comparisons We report the results of the selected model on the test sets of the four corpora in Table 8.11. They are compared to other published results. All metrics are not fully comparable as some papers only report results with gold POS tags, or with a limit on sentence lengths. However, our setting is the most restrictive, as the parser performs joint tagging and scale to full corpora. Despite these differences, ML-GAP outperforms other parsers, including the bi-LSTM parser of Stanojević and Garrido Alhama (2017)[9] that is based on a SWAP action to predict discontinuities. This observation confirms in another setting the results of Chapter 7, namely that GAP transition systems have more desirable properties than SWAP transition systems.

[9]On Negra, they have similar results but our parser has no access to gold tags whereas their parser does.

trans. syst.	features	Arabic	Basque	French	German	Hebrew	Hungarian	Korean	Polish	Swedish	Avg.
					DEV						
SR-LEX	BASE	83.42	88.06	82.29	88.03	90.01	90.34	86.91	93.25	78.74	86.78
SR-LEX	+NT	83.35	87.72	82.52	88.37	89.62	90.70	86.69	92.88	79.09	86.77
SR-LEX	+LEX	83.29	87.67	82.30	88.14	90.08	90.28	86.86	93.37	78.16	86.68
SR-LEX	+NT+LEX	83.18	88.16	82.39	88.64	89.5	90.69	86.68	92.83	**79.38**	86.83
ML	BASE	**83.88**	88.13	**82.64**	**89.35**	90.30	**90.9**	86.92	**93.43**	78.84	**87.15**
ML	+NT	82.67	88.12	82.41	89.02	89.94	90.72	86.80	93.35	78.7	86.86
ML-LEX	BASE	83.37	**88.61**	82.43	88.58	90.14	90.39	86.94	93.29	78.21	86.88
ML-LEX	+NT	83.09	88.20	82.23	88.87	**90.34**	90.47	86.94	93.40	78.27	86.87
ML-LEX	+LEX	82.91	88.04	81.83	88.70	90.17	90.18	**87.25**	93.09	78.09	86.7
ML-LEX	+NT+LEX	82.43	88.01	81.93	88.88	89.99	90.58	86.89	92.96	78.07	86.64
TOK+CLSTM+M+D, Chapter 6		83.04	87.93	82.19	88.70	89.64	90.52	86.78	93.23	79.14	**86.8**
TOK+MMT+D, Chapter 6		83.07	88.35	82.35	88.75	**90.34**	**91.22**	86.55	**94.0**	**79.64**	**87.14**
					TEST						
ML	BASE	**83.56**	**88.81**	**82.55**	**85.66**	89.90	91.58	**86.19**	93.33	83.54	**87.24**
TOK+CLSTM+M+D, Chapter 6		82.92	87.87	82.1	85.12	89.19	90.95	85.89	92.67	83.44	86.68
TOK+MMT+D, Chapter 6		82.77	**88.81**	82.49	85.34	89.87	**92.34**	86.04	**93.64**	84.0	**87.26**
Fernández-González and Martins (2015), dep		-	85.90	78.75	78.66	88.97	88.16	79.28	91.20	82.80	(84.22)
Crabbé (2015), tb+perceptron, beam=8		81.31	84.94	80.84	79.26	89.65	90.14	82.65	92.66	83.24	84.97
Durrett and Klein (2015), neural, CKY		80.24	85.41	81.25	80.95	88.61	90.66	82.23	92.97	83.45	85.09
Coavoux and Crabbé (2016), FFNN, greedy		80.71	86.24	79.91	80.15	88.69	90.51	85.10	92.96	81.74	85.11
Legrand and Collobert (2016), RNN		80.4	87.5	80.8	82.0	**91.6**	90.0	84.8	93.0	80.5	85.6
Björkelund et al. (2014), ens+rerank		81.32	88.24	82.53	81.66	89.80	91.72	83.81	90.50	**85.50**	86.12

Table 8.12: Projective parsing results (development and test sets).

8.3.4.2 Multilingual Projective Constituency Parsing

We report results for projective parsing on the SPMRL dataset in Table 8.12. The ML transition system with BASE features is the best performing system. Overall, the selected model (ML with BASE features) matches the result of the TOK+MMT+D pipeline model from Chapter 6, that used a highly accurate morphological tagger (MARMOT Mueller et al., 2013) informed with morphological lexicons.

As observed in the discontinuous case, the different feature templates have very little effect on parsing accuracy (± 0.2). Lexicalized systems perform no better when they have access to head features (+LEX). The addition of nonterminal features (+NT) has hardly any effect.

We report UAS results for the ML-LEX transition system in Table 8.13 to assess the effect of lexicalized features on the predicted dependency structures. Surprisingly enough, the lexicalized features bring no improvement and are even detrimental in several cases (Polish, Basque). We believe that these results suggest that even lexicalized models do not actually model bilexical relations but rely on other surface features.

ML-LEX	Unlabelled Attachment Score (Dev)								
	Arabic	Basque	French	German	Hebrew	Hungarian	Korean	Polish	Swedish
BASE	82.78	74.65	88.26	71.80	85.63	74.69	90.34	90.15	83.23
+LEX	82.65	74.07	88.16	71.97	85.37	74.62	90.68	90.22	82.87
+NT	82.72	75.00	88.42	72.04	85.13	75.02	90.34	90.31	82.82
+NT+LEX	82.52	74.59	88.14	72.15	84.91	75.09	90.57	89.64	82.54

Table 8.13: UAS results with ML-LEX (dev).

8.4 Error Analysis

In this section, we provide an error analysis focused on the discontinuous constituents for the best performing model. After describing the methodology, we discuss the main findings of this analysis.

8.4.1 Methodology

All analyses are performed on the development set of the Discontinuous Penn Treebank (Evang and Kallmeyer, 2011), using the predictions of the ML-GAP model with BASE features. We used DISCODOP (van Cranenburgh et al., 2016) to extract the precision and recall on discontinuous constituents for each sentence. Out of 278 sentences containing a discontinuity (excluding those in which the discontinuity is only due to the punctuation), 165 were exact matches for discontinuous constituents and 113 contained at least one error.

Following Evang (2011), we categorized errors according to the phenomena producing the discontinuities. We used the following classification, where the main discontinuous constituent is in bold:

- Wh-extractions (questions, relative clauses):
  - ***What else*** *is one to* ***make of the whacky save-the-earth initiative just proposed by several major environmental groups and organized by the state's attorney general****?*
- Fronted quotations:
  - ***Currently, average pay for machinists is $13.39 an hour,*** *Boeing* ***said****.*
- It-extraposition:

  - *Were **it** true **that a weak currency paves the way for trade surpluses**, then presumably Argentina would be the center of today's global economy.*

- Circumpositioned quotations:

  - ***"We wanted to highlight the individual, not the environment,"** he says, **"and black and white allows you to do that better than color."***

- Extraposed modifiers or complements:

  - *"I had **calls** all night long **from the States**," he said.*

- Subject-verb inversions:

  - ***Grinned** Griffith Peck, a trader in Shearson Lehman Hutton Inc.'s OTC department: **"I tell you, this market acts healthy."***

For each phenomenon occurrence, we sub-categorized the parser's prediction in one of the following category:

1. **Perfect match**: the parser correctly identified every discontinuous constituent involved (discontinuous F1=100).

2. **Non detection** (false negative): the parser failed to detect a discontinuity (discontinuous F1=0).

3. **Partial recognition**: the parser correctly identified the phenomenon involved but did not obtain a 100 discontinuous F1 due to some mistakes. This category includes labelling mistakes, errors in the scope of the phenomenon, and errors in the internal structure of a discontinuous constituent.

Finally, we also report false positives for each phenomenon: cases where the parser predicted a discontinuity to model a phenomenon that does not occur in the reference tree.

	Recall	Precision	F1
All	91.15	91.25	91.20
Discontinuous constituents	66.59	78.36	72.00

Table 8.14: Precision and recall (development) for the selected model.

8.4.2 Observations

The precision and recall of the model on the whole corpus are reported in Table 8.14. As was observed in Chapter 7 with the structured perceptron model, the precision on discontinuous constituents is much higher than the recall, meaning that the model is rather conservative when predicting discontinuities.

Results per phenomenon are reported in Table 8.15, ordered by number of raw occurrences. We discuss them in the next sections. For convenience, we repeat the example corresponding to each phenomenon at the beginning of each section.

8.4.2.1 Wh-extractions

Example: ***What else** is one to **make of the whacky save-the-earth initiative just proposed by several major environmental groups and organized by the state's attorney general**?*

Non Detections For wh-extractions, a recurrent ambiguity accounts for 7 non-detected occurrences: *that* clauses are sometimes relative clauses (with extraction) and sometimes a complement clause (without extraction) as illustrated in the following sentences:

1. (NP the place (SBAR **that** world opinion has been celebrating **over**))
2. (NP the consensus . . . (SBAR that the Namibian guerrillas were above all else the victims of suppression by neighboring South Africa.))

Sentence (1) is annotated with a discontinuous PP containing only *that* and *over* to model an extraction. In contrast, there is no extraction in sentence (2) where the *that* clause is a complement clause of the preceding noun.

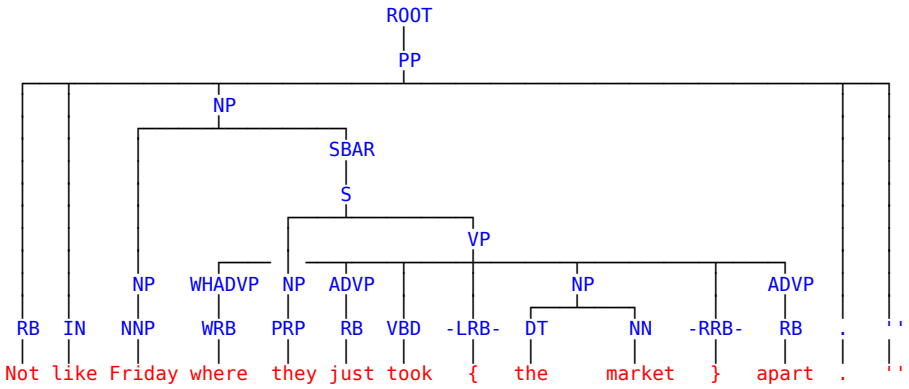
In the 7 cases, the parser predicts a complement clause instead of a relative clause with extraction of the object of a verb (5 cases) or the object of a stranded preposition (2 cases).

Partial recognitions The occurrences of partial recognition of a wh-extraction break down into several prototypical patterns. In 10 cases, the parser found the extracted phrase and attached it at the correct extraction site, but made a mistake in the scope of the discontinuous phrase (typically a VP). In 4 cases, the mistake consists of just an adverb attached to the sentence instead of the VP, as illustrated in Figure 8.3 (upper part). In the

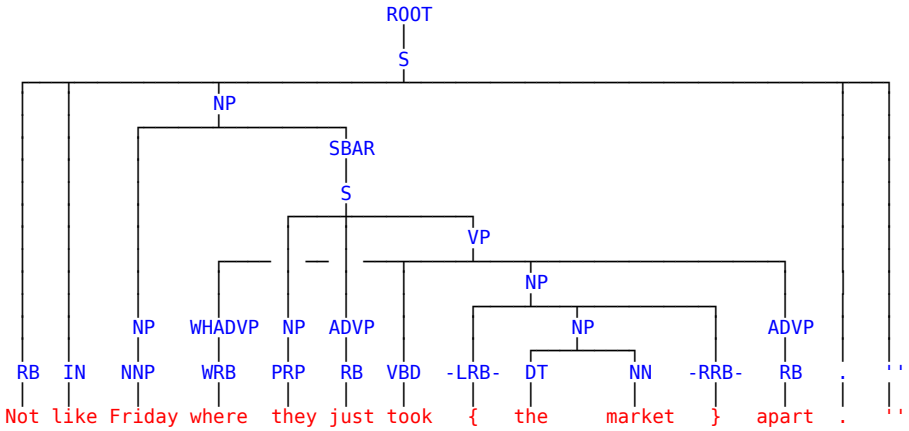
Wh-extractions	Gold	122	100%
	Perfect match	87	71.3
	Partial recognition	19	15.6
	Non detection	16	13.1
	False positives	8	NA
Fronted quotations	Gold	81	100%
	Perfect match	77	95.1
	Partial recognition	3	3.7
	Non detection	1	1.2
	False positives	0	NA
Extrapositions	Gold	44	100%
	Perfect match	10	22.7
	Partial recognition	1	2.27
	Non detection	33	75
	False positives	3	NA
Circumpositioned quotations	Gold	22	100%
	Perfect match	11	50
	Partial recognition	10	45.4
	Non detection	1	4.5
	False positives	3	NA
It-extrapositions	Gold	16	100%
	Perfect match	6	37.5
	Partial recognition	2	12.5
	Non detection	8	50
	False positives	2	NA
Subject-verb inversion	Gold	5	100%
	Perfect match	4	80
	Non detection	1	20
	False positives	1	NA

Table 8.15: Evaluation statistics per phenomenon.

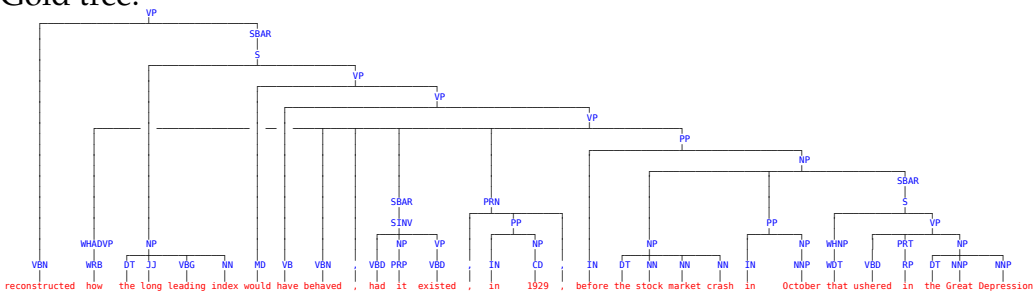
Gold tree:



Predicted tree:



Gold tree:



Predicted tree:

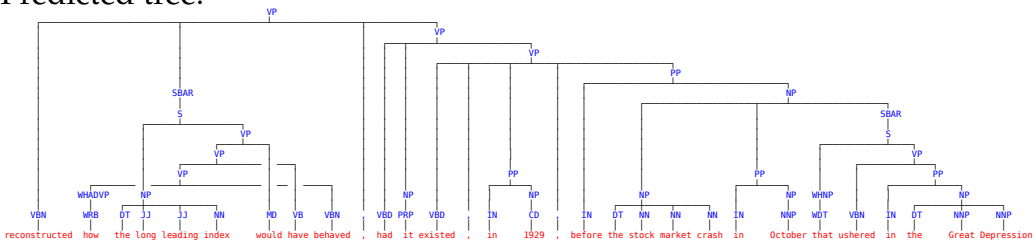


Figure 8.3: Partial recognition cases for wh-extractions: VP scope errors.

remaining cases, the parser failed to attach correctly other dependents of the VP (Figure 8.3, lower part).

In 8 cases, the parser found an incorrect attachment site for the extracted phrase. The difficulty arises when there are several candidate sites, several embedded verbs or prepositions. The parser either attaches the extracted phrase too high or too low in the tree. Trees in Figure 8.4 illustrate two cases with an ambiguity between the extraction of the object of a verb or of a preposition, that the parser failed to solve. The first tree exhibits an extraction from a stranded preposition. The parser predicted an extraction of the object of *know*. In the second case, the object of the verb is extracted, wheras the parser attached the extracted element as an object of *before*.

The last case of partial recognition of a wh-extraction is a case of multi-site extraction. When a phrase is extracted from the object positions of two coordinated phrases, it is attached to the coordination phrase, which in most cases does not produce a discontinuity.[10] In the trees in Figure 8.5, the parser interpreted the extracted *where* as a modifier of the first coordinated VP instead of attaching it higher to the VP coordination.

False positives Wh-extractions exhibit the highest number of false positives. There are two recurrent types of errors. In the case of multi-site extraction (3 occurrences), the parser tends to attach the extracted element to the first coordinated phrase (Figure 8.6, upper part), whereas the gold tree does not usually contain discontinuities in these cases (there are exceptions as in Figure 8.5).

In the 5 remaining cases, the parser predicted a relative clause with the extraction of the object of a verb. The gold reference trees either display a relative clause with an extracted subject with no discontinuity (Figure 8.6, lower part), or a complement clause introduced by *that*.

8.4.2.2 Fronted Quotations

Example: ***Currently, average pay for machinists is $13.39 an hour,*** *Boeing* ***said.***

Fronted quotations are the second most frequent phenomenon producing discontinuous constituents in the data. They exhibit fairly regular syntactic patterns and are very well predicted by the parser (95% perfect

[10] If the constraint that a phrase has a single parent is relaxed, the extracted phrase could be attached to both extraction sites. However, the resulting graph would no longer be seen as an LCFRS derivation, but rather a Range Concatenation Grammar (RCG) derivation.

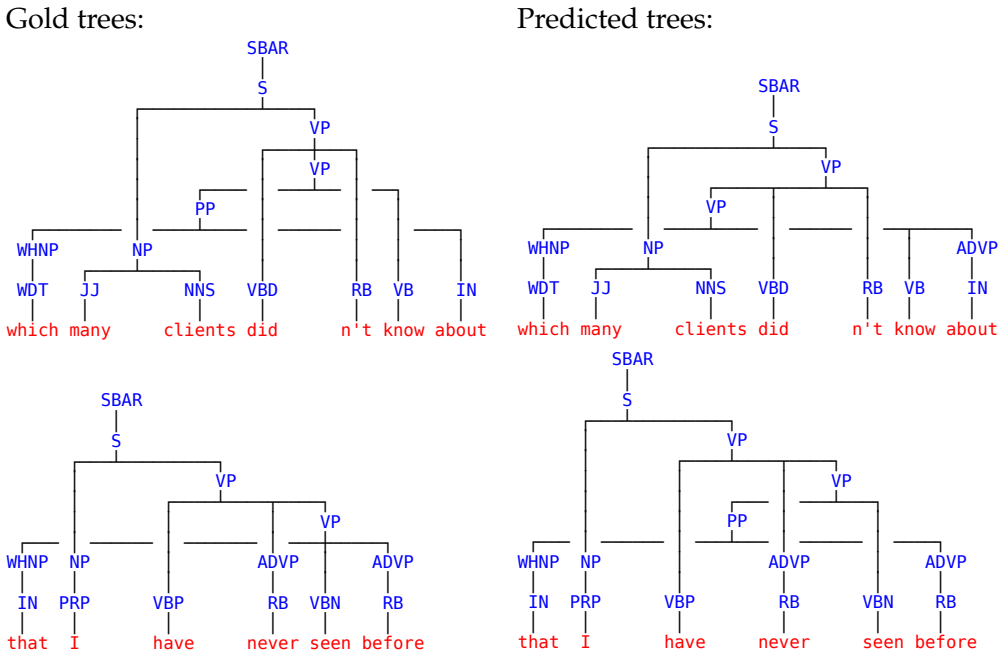


Figure 8.4: Partial recognition cases for wh-extractions: extraction site errors.

match). The only non-detection case is a sentence that contains no quotation marks to delimit the quotation, which may explain the mistake. In the partial match cases, the parser predicted a fronted quotation larger than it was actually.

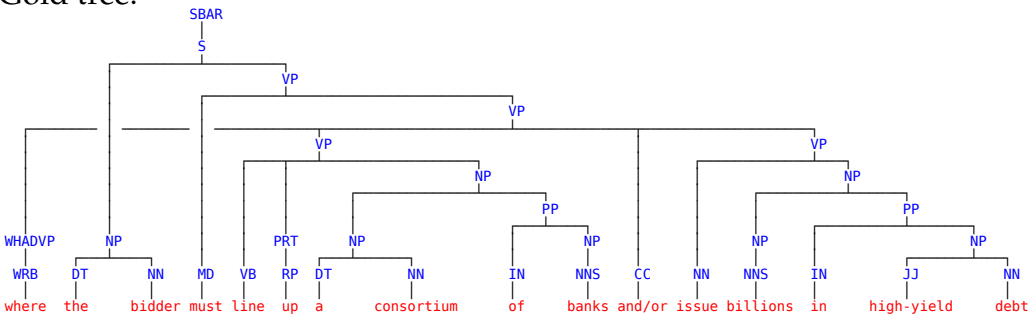
8.4.2.3 Extrapositions

Example: *"I had **calls** all night long **from the States**," he said.*

On extrapositions, the parser scored relatively low, compared to other phenomena. This is perhaps due to the fact that there is no clear recurrent pattern that makes the phenomenon easy to detect: the attachment ambiguities to solve are as hard as prepositional attachments and often require some semantic or world knowledge to be solved.

Except for 4 occurrences, the non-detected cases were extraposed adjuncts or complements of nouns. The two most frequent situations are illustrated in Figure 8.7. In the first case, the modifier of an object NP occurs after a dependent of the verb (either a modifier or an argument). The parser misattached the PP to the ADVP. In the second case, a modifier for an NP occurs after the verb. The parser has a tendency to attach these post-verbal modifiers locally to the verb.

Gold tree:



Predicted tree:

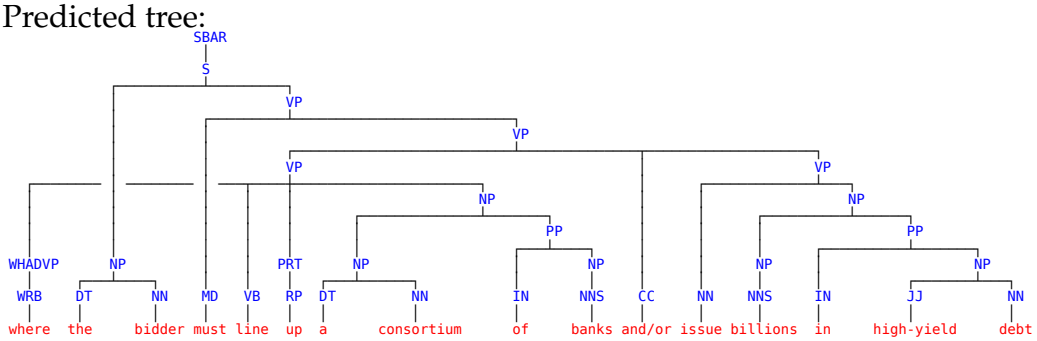
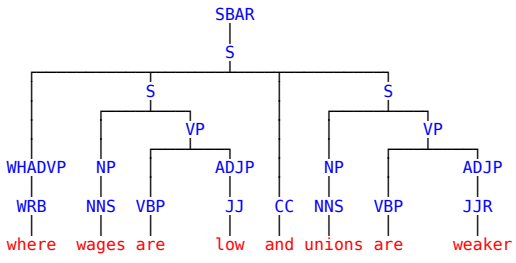
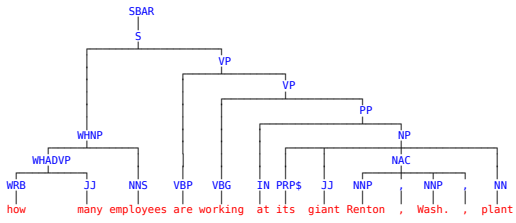


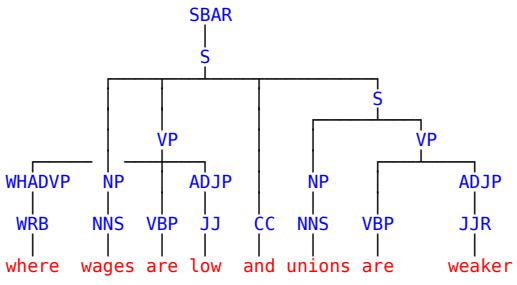
Figure 8.5: Partial recognition cases for wh-extractions: shared dependent cases (multi-site extraction).

Gold trees:





Predicted trees:



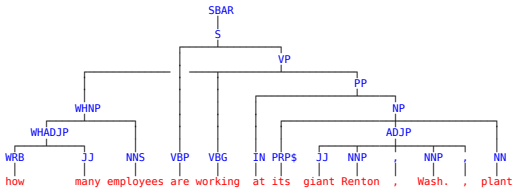


Figure 8.6: False positive wh-extractions.

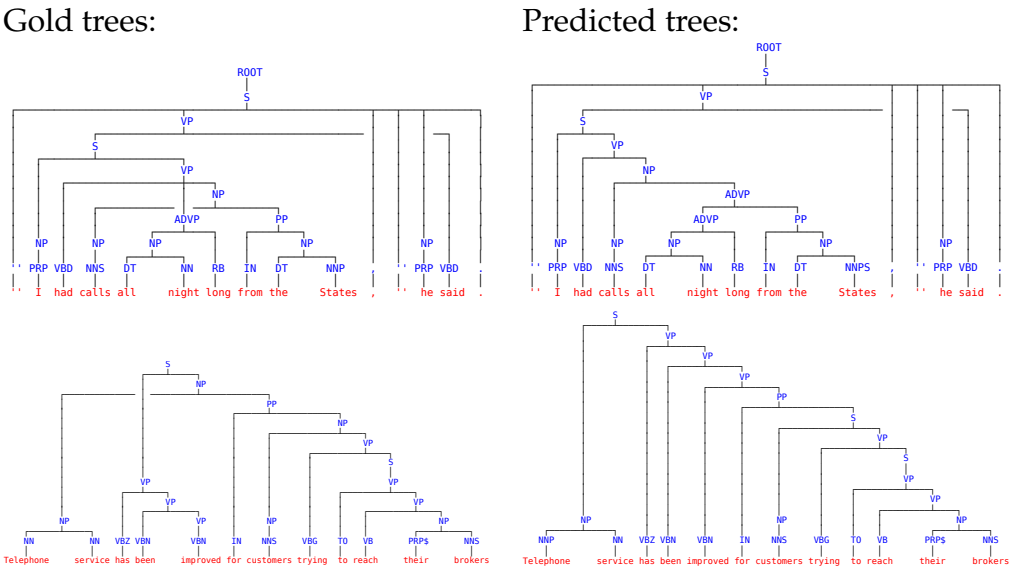


Figure 8.7: Non-detected extrapositions.

8.4.2.4 Circumpositioned Quotations

Example: ***"We wanted to highlight the individual, not the environment,"*** *he says,* ***"and black and white allows you to do that better than color."***

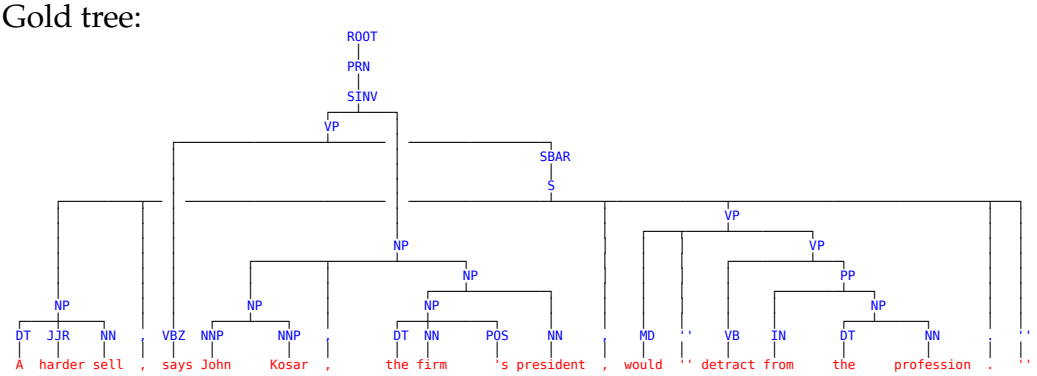
Circumpositioned quotations were mostly well recognized by the parser. In all partial recognition cases, the parser correctly attached a discontinuous quotation to a speech verb. However, it failed to label correctly the different fragments of the quotation. Figure 8.8 shows two typical cases of partial recognition. In the first sentence, the parser labelled the quotation as a VP, whereas it is an SBAR that rewrites as S in the gold reference.[11]

In the second sentence, the main verb is in between two coordinated VPs. The parser chose to analyse both fragments separately, predicting that the first one is a sentence, and that the second one is a VP without a corresponding subject.

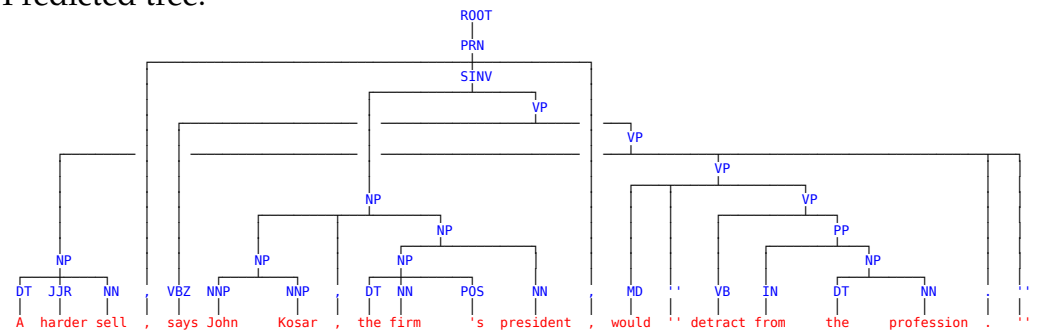
These errors illustrate the limitations of the model feature templates. Consider the configurations the parser is in when it labels the three embedded VPs of the tree in Figure 8.8 (upper part). These three configurations are shown in Table 8.16 along with the features used to predict whether d_0 is a constituent and what its label is. In fact, the parser predicts LABEL-VP

[11]The attachment of punctuation is also different but it is not taken into account by the evaluator.

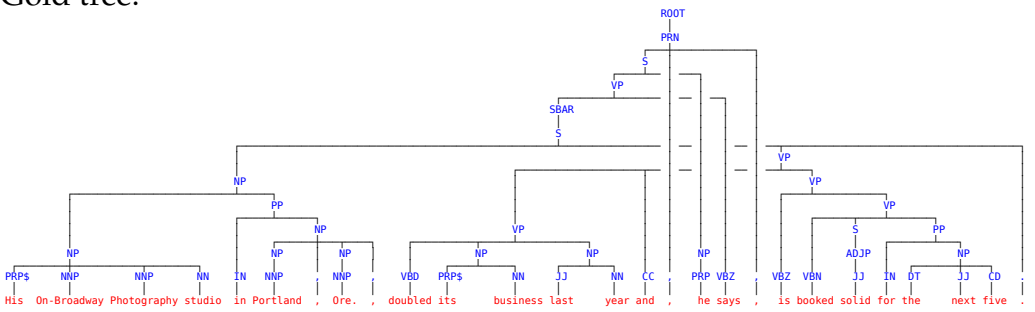
Gold tree:



Predicted tree:



Gold tree:



Predicted tree:

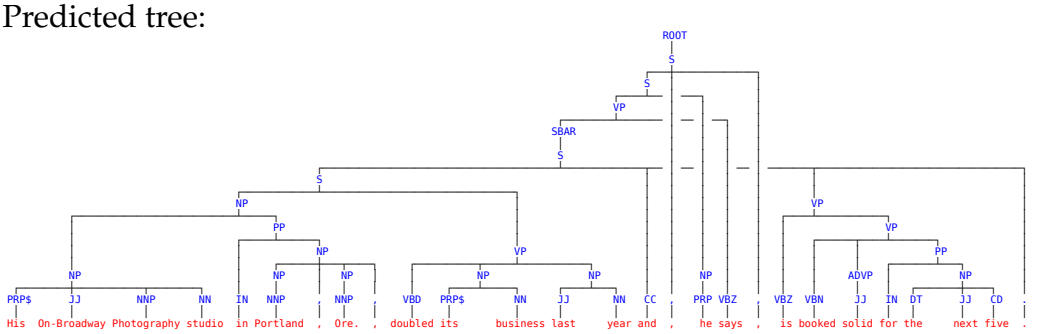


Figure 8.8: Partially recognized circumpositioned quotations.

for both c' and c'' because they have the exact same feature instantiation.[12] As a consequence, it could not have predicted correctly both the S-SBAR labels (with the unary chain S@SBAR) and the VP.

This observation has very interesting implications. First of all, it is somewhat surprising that the BASE feature model is the best one despite this defect. It would be expected that a higher number of templates and the use of head information would prevent this situation from happening. Secondly, it seems important to model the internal structure of a constituent to make a labelling decision. In particular, relying only on the two tokens at its left and right boundaries is insufficient because it does not take into account the presence of a discontinuity. In other words, the set of indices {*A harder sell says would "detract from the profession."*} and {*A harder sell would "detract from the profession."*} have the same representations (sentence from Figure 8.8). The only difference between them is the inclusion of *says* in the first one. This difference is invisible to features based only on the tokens at the boundaries of the constituent, i.e. *A* and *"* (quotation mark) in both cases. Designing feature templates on the form of a potential gap in a set of indices is a direction for improving the parser.

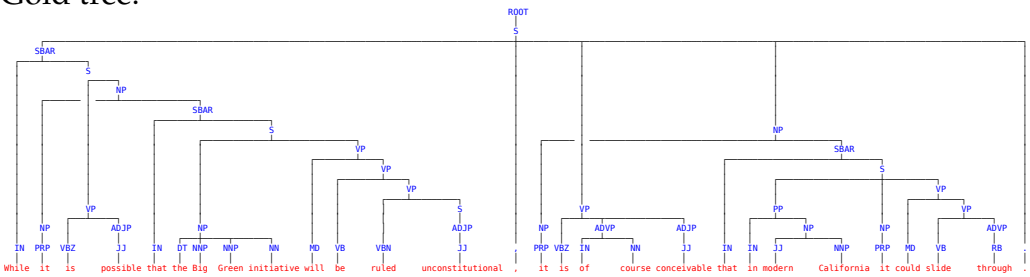
	s_4	s_3	s_2	s_1	s_0	d_0
$c =$	{A harder sell}	{,}	{says}	{John Kosar, the firm's president}	{,}	{would "detract from the profession."}
			LABEL-VP, GAP, GAP, GAP, MERGE $\Longrightarrow$			
$c' =$		{,}	{says}	{John Kosar, the firm's president}	{,}	{A harder sell would "detract from the profession."}
			LABEL-VP, GAP, GAP, MERGE $\Longrightarrow$			
$c'' =$			{,}	{John Kosar, the firm's president}	{,}	{A harder sell says would "detract from the profession."}

Template	Symbol value		
	c	c'	c''
b_0	NA	NA	NA
$d_0.w_l$	would	A	A
$d_0.w_r$	"	"	"
$d_1.w_r$	NA	NA	NA
$s_0.w_l$	,	,	,
$s_0.w_r$	,	,	,
$s_1.w_r$	president	president	president

Table 8.16: Limitations of the model feature templates: three configurations from the derivation of the tree in Figure 8.8 (second tree) and the corresponding features.

[12]The situation would have been different if the parser had chosen to attach punctuation sooner.

Gold tree:



Predicted tree:

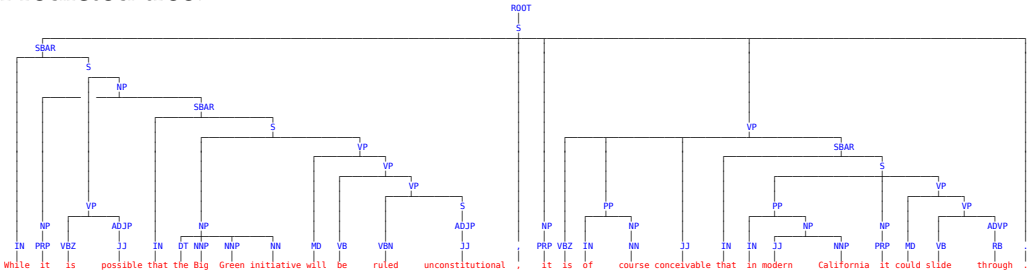


Figure 8.9: It-extraposition predictions.

8.4.2.5 It-extraposition

Example: *Were **it** true **that a weak currency paves the way for trade surpluses**, then presumably Argentina would be the center of today's global economy.*

The parser detected it-extrapositions correctly in around half of the cases. For the other occurrences, it attached the right part of the discontinuous NP to a predicate on its left, usually a verb or an adjective. Figure 8.9 shows a sentence with two occurrences of it-extrapositions. The first one was correctly predicted, whereas the second one was not, perhaps due to the intervening adverbial phrase (*of course*) that was parsed as a PP.

8.4.2.6 Subject-verb Inversions

Example: ***Grinned** Griffith Peck, a trader in Shearson Lehman Hutton Inc.'s OTC department: **"I tell you, this market acts healthy."***

The five cases of subject-verb inversions annotated with a discontinuity follow the same pattern: finite speech verb (*says*, *added*), subject NP, object S (quotation). The initial speech verb and the sentence form a discontinuous VP. In the single case not detected by the parser, the speech verb *Grinned* is followed by a named entity and was tagged as an NNP by the parser.

Finally, in the false positive case, the sentence follows the same pattern but was not annotated with a discontinuous VP: the verb, the subject NP and the object S share an SINV parent node. The parser predicts a discontinuous VP, consistently with the cases annotated likewise.

8.4.3 Discussion

Overall, several factors explain the mistakes the parser makes. The infrequent syntactic patterns are harder to predict, because the parser needs to generalize from few examples. A fair number of errors is caused by coordinations and modifier attachment ambiguities that are already known to be very difficult to solve in projective parsing, and sometimes require semantic or world knowledge to be handled adequately.

Finally, a limitation that is inherent to the parser is the representation of constituents based on their boundaries. Given that a constituent is represented by its leftmost token and rightmost token, two constituents that share the same boundaries will have the same representation, even though one of them contains a gap and the other does not. In this respect, finding a better representation for discontinuous constituents is an important issue.

8.5 Dynamic Programming

In this section, we outline how to apply dynamic programming decoding to transition-based discontinuous constituency parsing. We build on previous work on projective transition-based parsers with dynamic programming, starting with Huang and Sagae (2010) and derived works (Zhao et al., 2013; Thang et al., 2015; Mi and Huang, 2015; Shi et al., 2017).

The most frequent search algorithms used in transition-based parsing, greedy search and beam search, only explore a small part of the search space. With a beam of size k, a parser explores $\mathcal{O}(n \cdot k)$ parsing states, where n is the maximum length of a derivation for a sentence. With dynamic programming, the parser can explore an exponentially large number of parsing states, in polynomial time. Transition-based dynamic programming parsers that use beam search still achieve linear time parsing in practice (Huang and Sagae, 2010; Mi and Huang, 2015).

8.5.1 Equivalent Parsing States

The key to using dynamic programming is the fact that several parsing configurations might lead to the exact same feature instantiations. In such situations, the costs of outgoing transitions will be identical for these configurations and there is no need to compute them several times. Considering a feature template list S and a function $\mathbf{f}_S$ such that $\mathbf{f}_S(c)$ returns the list of symbols that instantiates the feature templates, two configurations c and c' are **equivalent** wrt S iff $\mathbf{f}_S(c) = \mathbf{f}_S(c')$. The equivalence relation depends crucially on S: when there are fewer atomic elements required to instantiate the feature templates, there will be fewer and larger equivalence classes. Therefore, minimal feature template sets are well suited to dynamic programming decoding.

In order to factorize equivalent states, the parser merges them into Dynamic Programming states (thereafter DP states). Whereas a parsing state encodes a single hypothesis, a DP state might encode an exponential number of hypotheses.

Figure 8.10 illustrates state merging for an artificial example. The three trees in (a) correspond to possible analyses. The derivations for constructing these trees with a shift-reduce-gap transition system[13] are represented as a tree in (b). Equivalent states are defined by a very simple feature template set ($\{s_0.l, s_0.r, d_0.l, d_0.r, b_0\}$), and are indicated by rectangles. For example, c''_7, c'_8 and c_7 form an equivalence class as illustrated in the following table:

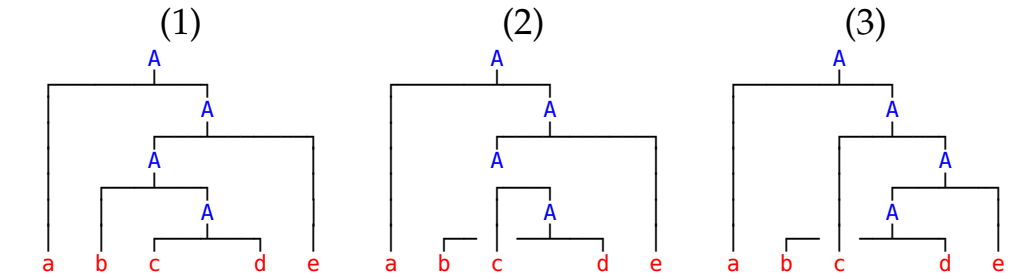
	D	S	B	$s_0.l$	$s_0.r$	$d_0.l$	$d_0.r$	b_0
c''_7	a c $A_{\{b,d\}}$	e	ϵ	e	e	b	d	ϵ
c'_8	a $A_{\{b,c,d\}}$	e	ϵ	e	e	b	d	ϵ
c_7	a $A_{\{b,c,d\}}$	e	ϵ	e	e	b	d	ϵ

8.5.2 Tree Structured Stack

With beam search, when the parser performs a new step, it considers every possible transition applicable to the k best states at step i and keeps the k best transitions to generate new states and insert them in the beam. Some of the new states might have the same predecessor state. A naive implementation of the beam consists in copying whole

[13]The shift-reduce-gap transition system was chosen for the sake of simplicity but these observations easily generalize to merge-label-gap transition systems.

(a) 3 possible analyses



(b) Search space for the three analyses. Rectangles indicate merged states.

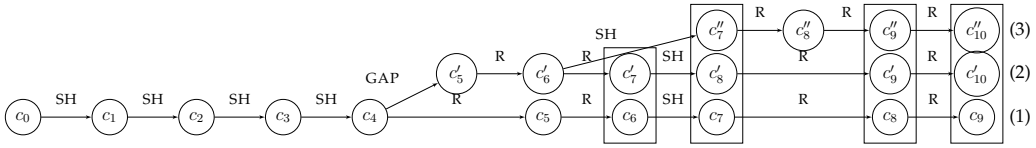


Figure 8.10: Dynamic programming: state merging. Feature template set: $\{s_0.l, s_0.r, d_0.l, d_0.r, b_0\}$.

configurations at each parsing state to generate the new states. With such an implementation, a projective parser has a $\mathcal{O}(n^2)$ complexity in time and space, as each parsing step incurs a $\mathcal{O}(n)$ cost for copying configurations.

To avoid this cost, an efficient beam implementation must factorize what is shared by different configurations. The Tree Structured Stack (TSS)[14] is a tree data structure in which each path from the root to a leaf encode the stack of a parsing configuration. In such a structure, the common stack prefixes of different configurations in the beam are shared. Parsing actions can be performed in $\mathcal{O}(1)$ time and space complexity, as no copying is required (Goldberg et al., 2013).

Figure 8.11 shows the tree-structured stack for hypotheses (1) and (2). Red dotted lines indicate the relations between several branches in the stack, i.e. the corresponding subtrees being constructed. Each path from the root to another node is the stack at one stage of the analysis. When the parser performs a shift, it adds a new arc to a leaf of the TSS and the stack grows. When it performs a reduction, the stack shrinks. The parser must grow a new branch from an internal node of the TSS.

Since discontinuous parsing involves reordering terminals, some amount of copying cannot be avoided. In particular, the cost of performing a reduction[15] is proportional to the number of preceding GAP actions. For

[14]The implementations of the parsers described throughout this dissertation are all based on a TSS for beam search.

[15]Or a merge, depending on the transition system.

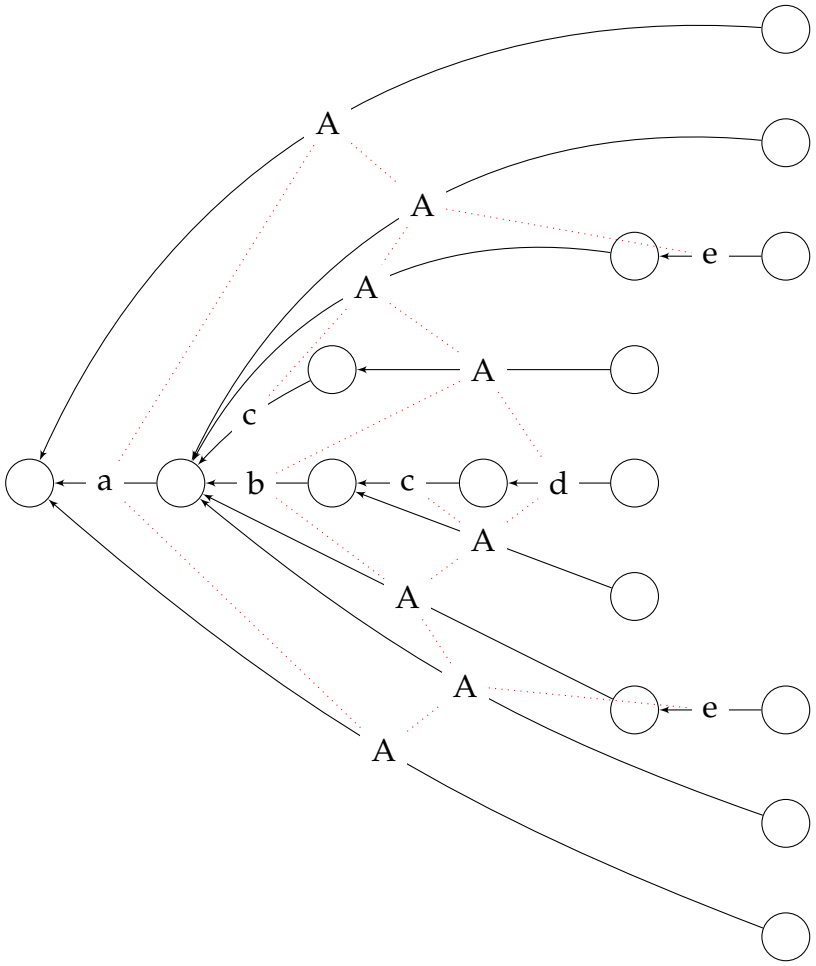
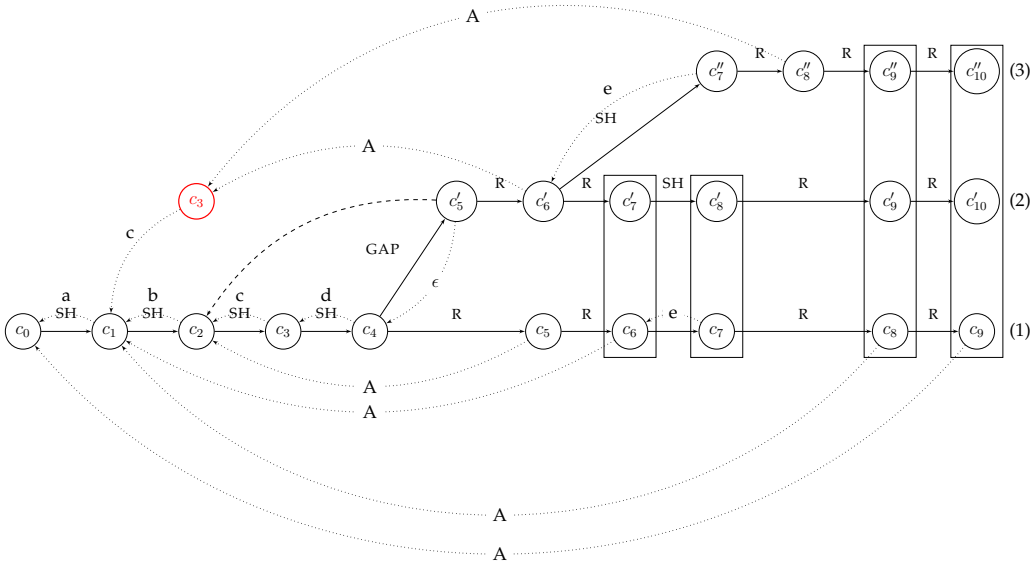


Figure 8.11: Tree-structured stack for two analyses.

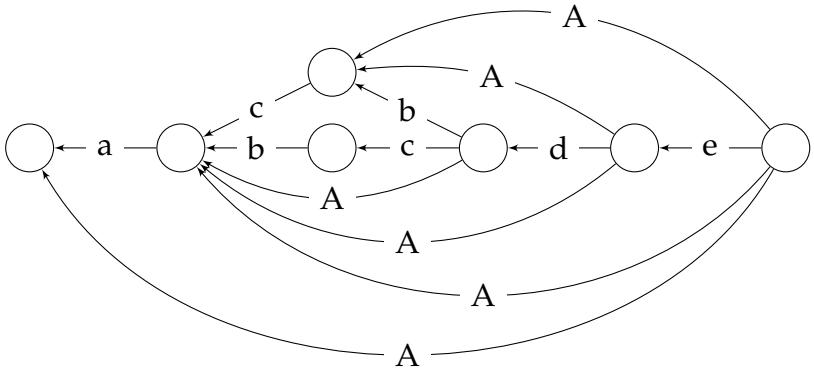
example, when the parser performs the reduction to construct the discontinuous constituent $A_{\{b,d\}}$, it must copy the content of the stack between these two arcs onto a new branch, to which it pushes the new nonterminal. The second arc labelled *c*, in the upper part of Figure 8.11, is the result of this operation (it has not been constructed by a SHIFT).

8.5.3 Encoding a TSS in the Parsing States

The TSS can be encoded directly into the parsing states with a set of pointers. Figure 8.12a represents the search space augmented with such pointers.



(a) GSS encoded in the search space with backpointers.



(b) Table of well-formed sub-strings in the discontinuous case (GSS).

Figure 8.12: Search space and GSS.

8.5.3.1 Projective Case

Let us consider first analysis (1), which is simpler because it produces a projective tree. The dotted arcs, labelled with grammar symbols, link each state to its predecessor in the stack. For example, the state of the stack $(S + D)$ in configuration c_5 can be recovered by following the dotted arcs downto c_0: [a, b, A] corresponding to the path (c_5, c_2, c_1, c_0). In the terms of Huang and Sagae (2010), the stack predecessor c' of a parsing state c is called a **predictor state** and is written $c' \in \pi(c)$. The set $\pi(c)$ contains the states with which c can combine in a reduction. In non-DP decoding, $\pi(c)$ is a singleton, in DP decoding though, it can have several elements.

The computation of the predictor state of a parsing state c depends on the action producing state c.

- If $c = \text{SHIFT}(c')$, then c' is the predictor state of c, because SHIFT pushes a new item onto the stack. In Figure 8.12a, each shifted state points to the state that generated it with an arc labelled by the newly shifted terminal.
- If $c = \text{REDUCE-UNARY-X}(c')$, then $\pi(c) = \pi(c')$.
- If $c = \text{REDUCE-X}(c')$, then $\pi(c) = \pi(c'')$, where $c'' \in \pi(c')$. For example, c_6 is the result of $\text{REDUCE-A}(c_5)$. The reduction combines state c_5 and its predictor state c_2. The predictor state of c_6 is c_1, i.e. the predictor state of c_2. The arc between c_1 and c_6 is labelled by a nonterminal A.

8.5.3.2 Discontinuous Case

The situation is slightly more complex when GAP actions are involved, because of the reordering that happens. In the projective case, a reduction involves a state and its predictor and focuses on two topmost element in the stack. In the discontinuous case, a reduction applies to a state (addressing the topmost element in the stack d_0) and another state that may be arbitrary far in the stack (s_0).

To handle the discontinuous case, we make the distinction between **stack relations** (σ) that are used to recover the full stack $(S + D)$ corresponding to a configuration, and **predictor relations** ($\pi(c)$) that points to the states with which c can combine in a reduction.

In projective parsing, these two relations coincide perfectly because s_0 will always address the second element in the stack. However, they will not coincide when this is not the case, i.e. after a GAP action.

Let us now consider analysis (2) in Figure 8.12a. If $c = \text{GAP}(c')$, then the predictor state of c is the predictor state of the predictor state of c':

$$\pi(c) = \bigcup_{c'' \in \pi(c')} \pi(c'')$$

Therefore, c_2 is in $\pi(c'_5)$. We materialized this relation with a dashed arrow. In contrast, $\sigma(c) = \sigma(c')$, as the stack is not modified by the GAP action. This equality is represented with a dotted arc with an empty label (ϵ) between c'_5 and c_4. The reduction that produces c'_6 applies to c'_5 and its predictor state $c_2 \in \pi(c'_5)$. The two corresponding stack items are *b* for c_2 and *d* for c'_5, thanks to the arc with the empty label.

Finally, when a reduction applies to a gapped configuration, such as c'_5, the configurations between $\sigma(c'_5) = c_3$ and $\pi(c'_5) = \{c_3\}$ following stack arcs (σ), are duplicated to grow a new branch in the TSS. The new c_3 configuration, coloured in red, is the predictor state of c'_6 and has c_1 as predictor state.

8.5.3.3 Graph-Structured Stack

The parsing states in a set of merged states share their predictor states. For this reason, we only drew a single dotted arc going out of a DP-state. For example, c_8, c'_9 and c''_9 all have c_1 as a predictor state. Seen from the perspective of DP states, the stack encoded in predictor states is in fact a graph, illustrated in Figure 8.12b. The Graph-Structured Stack (Tomita, 1988, GSS) data structure encodes the same information as the table of well-formed substrings used in chart parsing (Huang and Sagae, 2010).

8.5.4 Towards Exhaustive Search with Dynamic Programming

As observed by Cross III (2016) and Shi et al. (2017), minimal feature template sets lend themselves well to dynamic programming, as they encourage the merging of many states. In fact, the number of DP states in the search space depends on the number of indices used to extract features from a configuration. Cross III (2016) proposed a simplified feature set for the merge-label transition system for projective constituency parsing (Cross and Huang, 2016a) that is determined by only two indices. This leads to only $\mathcal{O}(n^2)$ possible DP states. This observation has important consequences, as it makes exhaustive search more efficient than was previously thought possible in neural transition-based constituency parsing,

and opens the way for globally normalized loss functions. Decoding decomposes in two parts (Shi et al., 2017): scoring DP states and finding the highest scoring tree. The complexity of the first part is controlled by the number of templates, whereas the second depends on the transition system and its expressivity (typically $\mathcal{O}(n^3)$ for a projective dependency of constituency transition system).

Applying the same reasoning to the discontinuous case, the BASE feature template set is perfectly determined by six indices. There are indeed seven feature templates, but b_0 is necessarily instantiated by the token that follows $d_0.w_r$ in the sentence. Thus, the number of possible DP states is in $\mathcal{O}(n^6)$. However, the cost for finding the best-scoring tree remains a high polynomial. Searching exhaustively the space of all labelled discontinuous trees is not reasonable, but limiting the search to trees with a bounded fan-out might be a way to perform exhaustive search with the ML-GAP transition system.

8.6 Conclusion

In this chapter, we have presented an unlexicalized structure-label transition system based on the GAP action, and its lexicalized counterpart. We have investigated the properties of these transition systems and evaluated them in two multilingual settings: discontinuous constituency parsing and projective constituency parsing.

The experimental comparisons we have carried out showed that lexicalized systems are not required to achieve very high parsing accuracies and that even for lexicalized systems, the use of lexical features leads to no improvement compared to minimal span features.

There are several possible interpretations for these results. First of all, they suggest that it is questionable that parsers rely on bilexical statistics, which have been known to be very sparse (Bikel, 2004). It is more likely that they rely on surface syntactic patterns, captured by constituent boundaries. We hypothesize that the results also confirm the ability of LSTMs to encode a fair amount of syntactic information, and echo the study by Linzen et al. (2016) about verb agreement in English.

In the last sections, we have carried out an error analysis of our best model on the Discontinuous Penn Treebank and discussed dynamic programming decoding. We plan to implement and evaluate dynamic programming with beam search for discontinuous transition-based parsing. Such a decoding algorithm might prove useful for our parser as it uses a

minimal feature template set. Future work will also focus on the design of a dynamic oracle for the ML-GAP transition system.

Chapter 9

Conclusion and Perspectives

This dissertation has dealt with incremental transition-based constituency parsing and focused on parsing morphologically rich languages. Our contributions addressed several key issues in parsing MRLs, regarding learning biases, morphology-syntax interaction, and parsing algorithms.

First of all, we have designed a dynamic oracle algorithm for a lexicalized shift-reduce transition system. The oracle addressed learning biases that greedy parsers are subject to when they are trained with a fixed set of gold examples. The dynamic oracle was shown to improve learning on the languages of the SPMRL dataset and on the Penn Treebank.

Secondly, we have provided a generalizable method to integrate morphology and functional structure in a constituency parser with multitask learning. The architecture that we have presented obtains state-of-the-art results on the SPMRL dataset, both in constituency parsing and in morphological analysis. It is able to output high-quality labelled dependency trees as a by-product of constituency parsing. Moreover the architecture is generalizable to other situations, such as discontinuous constituency parsing (Chapter 8).

Thirdly, we have investigated how to predict discontinuous constituency trees, which are a way to model directly linguistic phenomena related to word-order variation. To this aim, we have designed a new transition system, SR-GAP that achieved state-of-the-art parsing results. We have investigated formal and empirical properties of SR-GAP to better understand its behaviour and performance.

Fourthly, we have studied the question of lexicalization in both projective and discontinuous transition-based constituency parsing and shown that explicit lexicalization is not required to achieve very high parsing results in a multilingual setting. To do so, we have presented a family of structure-label transition systems, ML-GAP and ML-GAP-LEX that we

have compared to SR-GAP using the multitask architecture introduced in Chapter 6. We have provided an error analysis of the best discontinuous model on English and discussed ways to improve it further. We have also outlined a dynamic programming decoding algorithm for discontinuous parsing.

Finally, we have released the different versions of the parser used in the experiments presented in this dissertation,[1] as well as scripts to produce the discontinuous version of the French Treebank used in Chapter 8, and the discontinuous versions of the French Question Bank (Seddah and Candito, 2016) and the Sequoia Treebank (Candito and Seddah, 2012).[2] The parser is currently state-of-the-art on discontinuous corpora, and on the SPMRL dataset, and performs jointly constituency parsing, morphological analysis and functional labelling.

There are several possible continuations of our work. First of all, we plan to implement and evaluate beam search decoding with dynamic programming for discontinuous constituency parsing. In order to improve training, we also plan to investigate globally normalized models for discontinuous parsing as well as the design of a dynamic oracle for the ML-GAP transition system.

Secondly, another interesting approach to improve the parser on difficult structures is to design a non-monotonic transition system, i.e. a transition system in which some actions can undo previous actions in order to recover from early wrong decisions. Such systems have been implemented for dependency parsing (Honnibal et al., 2013; Honnibal and Johnson, 2015; Fernández-González and Gómez-Rodríguez, 2017) but never proposed for constituency parsing.

Thirdly, we plan to investigate the use of the methods we developed to other tasks. The parser we presented in Chapter 5 has already been used to parse discourse structures (Braud, Coavoux, and Søgaard, 2017) in a multilingual setting. Some of the contributions of this dissertation could prove useful for discourse parsing. In particular, the multitask architecture of Chapter 6 may be extended to model both syntax and discourse relations and perform joint parsing.

[1] www.github.com/mcoavoux/mtg

[2] www.github.com/mcoavoux/french_disco_data

Bibliography

Anne Abeillé, Lionel Clément, and François Toussenel. *Building a Treebank for French*, pages 165–187. Springer Netherlands, Dordrecht, 2003. ISBN 978-94-010-0201-1. doi: 10.1007/978-94-010-0201-1_10. URL http://dx.doi.org/10.1007/978-94-010-0201-1_10.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1231. URL http://aclweb.org/anthology/P16-1231.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 349–359, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1041.

Rachel Bawden and Benoît Crabbé. Boosting for efficient model selection for syntactic parsing. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1–11, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL http://aclweb.org/anthology/C16-1001.

D.M. Bikel. *On the Parameter Space of Generative Lexicalized Statistical Parsing Models*. 2004. URL https://books.google.fr/books?id=M2RsPgAACAAJ.

Anders Björkelund, Ozlem Cetinoglu, Richárd Farkas, Thomas Mueller, and Wolfgang Seeker. (re)ranking meets morphosyntax: State-of-the-art results from the SPMRL 2013 shared task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 135–145, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W13-4916.

Anders Björkelund, Özlem Çetinoğlu, Agnieszka Faleńska, Richárd Farkas, Thomas Mueller, Wolfgang Seeker, and Zsolt Szántó. Introducing the ims-wrocław-szeged-cis entry at the spmrl 2014 shared task: Reranking and morpho-syntax meet unlabeled data. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 97–102, Dublin,

Ireland, August 2014. Dublin City University. URL http://www.aclweb.org/anthology/W14-6110.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428, 2013.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer. URL http://leon.bottou.org/papers/bottou-2010.

Pierre Boullier. Proposal for a Natural Language Processing Syntactic Backbone. Research Report RR-3342, INRIA, 1998. URL https://hal.inria.fr/inria-00073347.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. Tiger treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories, September 20-21 (TLT02)*, Sozopol, Bulgaria, 2002.

Chloé Braud, Maximin Coavoux, and Anders Søgaard. Cross-lingual rst discourse parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 292–304, Valencia, Spain, April 2017. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/E17-1028.

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

Marie Candito and Djamé Seddah. Le corpus sequoia : annotation syntaxique et exploitation pour l'adaptation d'analyseur par pont lexical (the sequoia corpus : Syntactic annotation and use for a parser lexical domain adaptation method) [in french]. In *Proceedings of the Joint Conference JEP-TALN-RECITAL 2012, volume 2: TALN*, pages 321–334. ATALA/AFCP, 2012. URL http://aclanthology.coli.uni-saarland.de/pdf/F/F12/F12-2024.pdf.

Marie Candito, Benoît Crabbé, and Pascal Denis. Statistical french dependency parsing: Treebank conversion and first results. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*. European Languages Resources Association (ELRA), 2010. URL http://www.lrec-conf.org/proceedings/lrec2010/pdf/392_Paper.pdf.

Rich Caruana. Multitask learning. *Mach. Learn.*, 28(1):41–75, July 1997. ISSN 0885-6125. doi: 10.1023/A:1007379606734. URL http://dx.doi.org/10.1023/A:1007379606734.

Daniel Cer, Marie-Catherine de Marneffe, Dan Jurafsky, and Chris Manning. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk,

Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA). ISBN 2-9517408-6-7.

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180. Association for Computational Linguistics, 2005. URL http://aclanthology.coli.uni-saarland.de/pdf/P/P05/P05-1022.pdf.

Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, October 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1082.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1179.

Maximin Coavoux and Benoit Crabbé. Neural greedy constituent parsing with dynamic oracles. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 172–182, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1017.

Maximin Coavoux and Benoît Crabbé. Prédiction structurée pour l'analyse syntaxique en constituants par transitions : modèles denses et modèles creux. *Traitement Automatique des Langues*, 57(1), 2016. URL https://hal.inria.fr/hal-01365252.

Maximin Coavoux and Benoit Crabbé. Incremental discontinuous phrase structure parsing with the gap transition. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1259–1270, Valencia, Spain, April 2017a. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/E17-1118.

Maximin Coavoux and Benoit Crabbé. Multilingual lexicalized constituency parsing with word-level auxiliary tasks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 331–336, Valencia, Spain, April 2017b. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/E17-2053.

Maximin Coavoux and Benoît Crabbé. Comparaison d'architectures neuronales pour l'analyse syntaxique en constituants. In *Actes de la 22e conférence sur le Traitement Automatique des Langues Naturelles*, pages 293–304, Caen, France, June 2015. Association pour le Traitement Automatique des Langues. URL http://www.atala.org/taln_archives/TALN/TALN-2015/taln-2015-long-025.

Maximin Coavoux and Benoît Crabbé. Représentation et analyse automatique des discontinuités syntaxiques dans les corpus arborés en constituants du français. In *Actes de la 24e conférence sur le Traitement Automatique des Langues Naturelles*, pages 77–92, Orléans, France, June 2017. Association pour le Traitement Automatique des Langues. URL http://taln2017.cnrs.fr/wp-content/uploads/2017/06/actes_TALN_2017-vol1.pdf.

Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23, Madrid, Spain, July 1997. Association for Computational Linguistics. doi: 10.3115/976909.979620. URL http://www.aclweb.org/anthology/P97-1003.

Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics, July 2002. doi: 10.3115/1118693.1118694. URL http://www.aclweb.org/anthology/W02-1001.

Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July 2004. doi: 10.3115/1218955.1218970. URL http://www.aclweb.org/anthology/P04-1015.

Ronan Collobert. Deep learning for efficient discriminative parsing. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 224–232. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011. URL http://www.jmlr.org/proceedings/papers/v15/collobert11a/collobert11a.pdf.

Caio Corro, Joseph Le Roux, and Mathieu Lacroix. Efficient discontinuous phrase-structure parsing via the generalized maximum spanning arborescence. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1645–1655, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D17-1172.

Michael A. Covington. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102, 2001.

Benoit Crabbé. An lr-inspired generalized lexicalized phrase structure parser. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 541–552, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics. URL http://www.aclweb.org/anthology/C14-1052.

Benoit Crabbé. Multilingual discriminative lexicalized phrase structure parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1847–1856, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1212.

Matthew W Crocker. Mechanisms for sentence processing. *Language processing*, pages 191–232, 1999.

James Cross and Liang Huang. 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*, pages 1–11, Austin, Texas, November 2016a. Association for Computational Linguistics. URL https://aclweb.org/anthology/D16-1001.

James Cross and Liang Huang. Incremental parsing with minimal features using bi-directional lstm. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 32–37, Berlin, Germany, August 2016b. Association for Computational Linguistics. URL http://anthology.aclweb.org/P16-2006.

James Henry Cross III. *Parsing with Recurrent Neural Networks*. PhD thesis, 2016.

Hal Daumé, Iii, John Langford, and Daniel Marcu. Search-based structured prediction. *Mach. Learn.*, 75(3):297–325, June 2009. ISSN 0885-6125. doi: 10.1007/s10994-009-5106-x. URL http://dx.doi.org/10.1007/s10994-009-5106-x.

Éric Villemonte de la Clergerie. Parsing mcs languages with thread automata. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 193–200, UniversitÃ¡ di Venezia, May 2002. URL http://www.aclweb.org/anthology/W02-2228.

Amit Dubey and Frank Keller. Probabilistic parsing for german using sister-head dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 96–103, Sapporo, Japan, July 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075109. URL http://www.aclweb.org/anthology/P03-1013.

Greg Durrett and Dan Klein. 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 (Volume 1: Long Papers)*, pages 302–312, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1030.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1033.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1024.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: 10.1016/0364-0213(90)90002-E. URL http://groups.lis.illinois.edu/amag/langev/paper/elman90findingStructure.html.

Kilian Evang. *Parsing discontinuous constituents in English*. PhD thesis, Master's thesis, University of Tübingen, 2011.

Kilian Evang and Laura Kallmeyer. Plcfrs parsing of english discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 104–116, Dublin, Ireland, October 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W11-2913.

Vanessa Wei Feng and Graeme Hirst. Text-level discourse parsing with rich linguistic features. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 60–68, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P12-1007.

Daniel Fernández-González and Carlos Gómez-Rodríguez. A full non-monotonic transition system for unrestricted non-projective parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 288–298, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL http://aclweb.org/anthology/P17-1027.

Daniel Fernández-González and André F. T. Martins. Parsing as reduction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1523–1533, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1147.

Richard Futrell, Kyle Mahowald, and Edward Gibson. Quantifying word order freedom in dependency corpora. In *Proceedings of the Third International Conference on Dependency Linguistics (Depling 2015)*, pages 91–100, Uppsala, Sweden, August 2015. Uppsala University, Uppsala, Sweden. URL http://www.aclweb.org/anthology/W15-2112.

Daniel Gildea. Corpus variation and parser performance. In Lillian Lee and Donna Harman, editors, *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pages 167–202, 2001.

Yoav Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015. URL http://arxiv.org/abs/1510.00726.

Yoav Goldberg and Michael Elhadad. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750. Association for Computational Linguistics, 2010a. URL http://aclweb.org/anthology/N10-1115.

Yoav Goldberg and Michael Elhadad. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California, June 2010b. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N10-1115.

Yoav Goldberg and Joakim Nivre. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India, December 2012. The COLING 2012 Organizing Committee. URL http://www.aclweb.org/anthology/C12-1059.

Yoav Goldberg and Joakim Nivre. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414, 2013. ISSN 2307-387X. URL https://transacl.org/ojs/index.php/tacl/article/view/145.

Yoav Goldberg, Kai Zhao, and Liang Huang. Efficient implementation of beam-search incremental parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 628–633, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P13-2111.

Yoav Goldberg, Francesco Sartorio, and Giorgio Satta. A tabular method for dynamic oracles in transition-based parsing. *Transactions of the Association for Computational Linguistics*, 2:119–130, 2014. ISSN 2307-387X. URL https://transacl.org/ojs/index.php/tacl/article/view/302.

Carlos Gómez-Rodríguez and Daniel Fernández-González. An efficient dynamic oracle for unrestricted non-projective parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 256–261, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-2042.

Carlos Gómez-Rodríguez, Marco Kuhlmann, Giorgio Satta, and David Weir. Optimal reduction of rule length in linear context-free rewriting systems. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 539–547. Association for Computational Linguistics, 2009. URL http://aclanthology.coli.uni-saarland.de/pdf/N/N09/N09-1061.pdf.

Carlos Gómez-Rodríguez, Francesco Sartorio, and Giorgio Satta. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 917–927, Doha, Qatar, October 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1099.

Joshua Goodman. Semiring parsing. *Computational Linguistics*, 25(4), 1999. URL http://aclanthology.coli.uni-saarland.de/pdf/J/J99/J99-4004.pdf.

John Hale. A probabilistic earley parser as a psycholinguistic model. In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*, 2001. URL http://aclweb.org/anthology/N01-1021.

John T Hale. *Automaton theories of human sentence comprehension*. CSLI Publications, 2014.

David Hall, Greg Durrett, and Dan Klein. Less grammar, more features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 228–237, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P14-1022.

Johan Hall and Joakim Nivre. *Parsing Discontinuous Phrase Structure with Grammatical Functions*, pages 169–180. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-85287-2. doi: 10.1007/978-3-540-85287-2_17. URL http://dx.doi.org/10.1007/978-3-540-85287-2_17.

Katsuhiko Hayashi and Masaaki Nagata. Empty element recovery by spinal parser operations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 95–100, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://anthology.aclweb.org/P16-2016.

James Henderson. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 24–31, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1073445.1073459. URL http://dx.doi.org/10.3115/1073445.1073459.

James Henderson. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 95–102, Barcelona, Spain, July 2004. doi: 10.3115/1218955.1218968. URL http://www.aclweb.org/anthology/P04-1013.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Matthew Honnibal and Mark Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1162.

Matthew Honnibal, Yoav Goldberg, and Mark Johnson. A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 163–172, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W13-3518.

Liang Huang. Advanced dynamic programming in semiring and hypergraph frameworks (tutorial). *Survey Paper for the COLING Tutorial, Manchester, UK*, 2008.

Liang Huang and Kenji Sagae. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P10-1110.

Liang Huang, Suphan Fayong, and Yang Guo. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151, Montréal, Canada, June 2012. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N12-1015.

Mark Johnson. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 136–143, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073107. URL http://www.aclweb.org/anthology/P02-1018.

Aravind Joshi. How much context sensitivity is necessary for characterizing structural descriptions. In D. Dowty, L. Karttunen, and A Zwicky, editors, *Natural Language Processing, Theoretical Computational and Psychological Perspectives*. Cambridge University Press, 1985.

Laura Kallmeyer. *Parsing Beyond Context-Free Grammars*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 364214845X, 9783642148453.

Laura Kallmeyer and Wolfgang Maier. Data-driven parsing with probabilistic linear context-free rewriting systems. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 537–545, Beijing, China, August 2010. Coling 2010 Organizing Committee. URL http://www.aclweb.org/anthology/C10-1061.

Laura Kallmeyer and Wolfgang Maier. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1), 2013. doi: 10.1162/COLI_a_00136. URL http://aclweb.org/anthology/J13-1006.

Laura Kallmeyer and Wolfgang Maier. Lr parsing for lcfrs. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1250–1255, Denver, Colorado, May–June 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N15-1134.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016. ISSN 2307-387X. URL https://transacl.org/ojs/index.php/tacl/article/view/885.

Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075150. URL http://www.aclweb.org/anthology/P03-1054.

D.E. Knuth. On the translation of languages from left to right. *Information and Control*, 8 (6):607–639, 1965.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258, Valencia, Spain, April 2017. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/E17-1117.

Matthieu Labeau, Kevin Löser, and Alexandre Allauzen. Non-lexical neural architecture for fine-grained pos tagging. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 232–237, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1025.

Peter Lane and James Henderson. Incremental syntactic parsing of natural language corpora with simple synchrony networks. *IEEE Trans. Knowl. Data Eng.*, 13(2):219–231, 2001. doi: 10.1109/69.917562. URL https://doi.org/10.1109/69.917562.

Joël Legrand and Ronan Collobert. Deep neural networks for syntactic parsing of morphologically rich languages. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 573–578, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://anthology.aclweb.org/P16-2093.

Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P14-2050.

Roger Levy and Christopher Manning. Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 327–334, Barcelona, Spain, July 2004. doi: 10.3115/1218955.1218997. URL http://www.aclweb.org/anthology/P04-1042.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association of Computational Linguistics*, 4:521–535, 2016. URL http://aclanthology.coli.uni-saarland.de/pdf/Q/Q16/Q16-1037.pdf.

Jiangming Liu and Yue Zhang. In-order transition-based constituent parsing. *CoRR*, abs/1707.05000, 2017a. URL http://arxiv.org/abs/1707.05000.

Jiangming Liu and Yue Zhang. Shift-reduce constituent parsing with neural lookahead features. *Transactions of the Association for Computational Linguistics*, 5:45–58, 2017b. ISSN 2307-387X. URL https://transacl.org/ojs/index.php/tacl/article/view/927.

Wolfgang Maier. Discontinuous incremental shift-reduce parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1202–1212, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1116.

Wolfgang Maier and Timm Lichte. Characterizing discontinuity in constituent treebanks. In *Formal Grammar. 14th International Conference, FG 2009. Bordeaux, France, July 25-26, 2009. Revised Selected Papers*, volume 5591 of *LNCS/LNAI*, pages 167–182, Berlin, Heidelberg, New York, 2011. Springer-Verlag. doi: 10.1007/978-3-642-20169-1. URL http://webloria.loria.fr/~degroote/FG09/.

Wolfgang Maier and Timm Lichte. Discontinuous parsing with continuous trees. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 47–57, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W16-0906.

Christopher D. Manning. Computational linguistics and deep learning. *Comput. Linguist.*, 41(4):701–707, December 2015. ISSN 0891-2017. doi: 10.1162/COLI_a_00239. URL http://dx.doi.org/10.1162/COLI_a_00239.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics, Volume 19, Number 2, June 1993, Special Issue on Using Large Corpora: II*, 1993. URL http://aclweb.org/anthology/J93-2004.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 75–82, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219850. URL http://www.aclweb.org/anthology/P05-1010.

Marshall R. Mayberry III and Risto Miikkulainen. Sardsrn: A neural network shift-reduce parser. In *Proceedings of the 16th Annual International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 820–825, Stockholm, Sweden, 1999. San Francisco, CA: Kaufmann. URL http://nn.cs.utexas.edu/?mayberry:ijcai99.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/H/H05/H05-1066.

Ryan McDonald, Slav Petrov, and Keith Hall. Multi-source transfer of delexicalized dependency parsers. In *Proceedings of the 2011 Conference on Empirical Methods in*

Natural Language Processing, pages 62–72. Association for Computational Linguistics, 2011. URL http://aclanthology.coli.uni-saarland.de/pdf/D/D11/D11-1006.pdf.

Haitao Mi and Liang Huang. Shift-reduce constituency parsing with dynamic programming and pos tag lattice. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1030–1035, Denver, Colorado, May–June 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N15-1108.

Thomas Mueller, Helmut Schmid, and Hinrich Schütze. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D13-1032.

Mark-Jan Nederhof. Squibs and discussions: Weighted deductive parsing and knuth's algorithm. *Computational Linguistics*, 29(1), 2003. URL http://aclanthology.coli.uni-saarland.de/pdf/J/J03/J03-1006.pdf.

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*, pages 149–160, 2003.

Joakim Nivre. Incrementality in deterministic dependency parsing. In Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman, editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain, July 2004. Association for Computational Linguistics.

Joakim Nivre. Incremental non-projective dependency parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 396–403, Rochester, New York, April 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N/N07/N07-1050.

Joakim Nivre. Algorithms for deterministic incremental dependency parsing. *Comput. Linguist.*, 34(4):513–553, December 2008. ISSN 0891-2017. doi: 10.1162/coli.07-056-R1-07-027. URL http://dx.doi.org/10.1162/coli.07-056-R1-07-027.

Joakim Nivre. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P09/P09-1040.

Joakim Nivre and Mario Scholz. Deterministic dependency parsing of english text. In *Proceedings of Coling 2004*, pages 64–70, Geneva, Switzerland, Aug 23–Aug 27 2004. COLING.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76, Paris, France, October 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W09-3811.

Fernando C. N. Pereira and David H. D. Warren. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, 1983. URL http://aclanthology.coli.uni-saarland.de/pdf/P/P83/P83-1021.pdf.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July 2006. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P06/P06-1055.

Emily Pitler and Ani Nenkova. Using syntax to disambiguate explicit discourse connectives in text. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 13–16, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P09/P09-2004.

Oliver Plaehn. Computing the most probable parse for a discontinuous phrase structure grammar. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*, pages 91–106. Kluwer Academic Publishers, Norwell, MA, USA, 2004. ISBN 1-4020-2293-X. URL http://dl.acm.org/citation.cfm?id=1139041.1139047.

Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 412–418, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://anthology.aclweb.org/P16-2067.

B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992. doi: 10.1137/0330046. URL https://doi.org/10.1137/0330046.

Corentin Ribeyre, Eric Villemonte de la Clergerie, and Djamé Seddah. Because syntax does matter: Improving predicate-argument structures parsing with syntactic features. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 64–74, Denver, Colorado, May–June 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N15-1007.

Brian Roark, Asaf Bachrach, Carlos Cardenas, and Christophe Pallier. Deriving lexical and syntactic expectation-based measures for psycholinguistic modeling via incremental top-down parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 324–333, Singapore, August 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D/D09/D09-1034.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017. URL http://arxiv.org/abs/1706.05098.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.

Kenji Sagae and Alon Lavie. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132, Vancouver, British Columbia, October 2005. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W05/W05-1513.

Kenji Sagae and Alon Lavie. A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 691–698, Sydney, Australia, July 2006. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P06/P06-2089.

Cicero Dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1818–1826, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/santos14.html.

Helmut Schmid. Trace prediction and recovery with unlexicalized pcfgs and slash features. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 177–184, Sydney, Australia, July 2006. Association for Computational Linguistics. doi: 10.3115/1220175.1220198. URL http://www.aclweb.org/anthology/P06-1023.

Djamé Seddah, Marie Candito, Benoit Crabbé, and Henestroza Enrique Anguiano. Ubiquitous usage of a broad coverage french corpus: Processing the est republicain corpus. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*. European Language Resources Association (ELRA), 2012. URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/1130_Paper.pdf.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W13-4917.

Djamé Seddah and Marie Candito. Hard time parsing questions: Building a questionbank for french. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo,

Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may 2016. European Language Resources Association (ELRA). ISBN 978-2-9517408-9-1.

Tianze Shi, Liang Huang, and Lillian Lee. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 12–23, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D17-1002.

Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343, Aug 1985. ISSN 1573-0549. doi: 10.1007/BF00630917. URL https://doi.org/10.1007/BF00630917.

Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. Bayesian symbol-refined tree substitution grammars for syntactic parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–448, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P12-1046.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 88–95, Washington, DC, USA, March 1997. Association for Computational Linguistics. doi: 10.3115/974557.974571. URL http://www.aclweb.org/anthology/A97-1014.

Richard Socher, Christopher D Manning, and Andrew Y Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–9, 2010.

Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P13-1045.

Anders Søgaard and Yoav Goldberg. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 231–235, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://anthology.aclweb.org/P16-2038.

Miloš Stanojević and Raquel Garrido Alhama. Neural discontinuous constituency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1667–1677, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D17-1174.

Pontus Stenetorp. Transition-based dependency parsing using recursive neural networks. In *Deep Learning Workshop at the 2013 Conference on Neural Information Processing Systems (NIPS)*, Lake Tahoe, Nevada, USA, December 2013.

Mitchell Stern, Jacob Andreas, and Dan Klein. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL http://aclweb.org/anthology/P17-1076.

Milan Straka, Jan Hajič, Jana Straková, and jr. Jan Hajič. Parsing universal dependency treebanks using neural networks and search-based oracle. In *14th International Workshop on Treebanks and Linguistic Theories (TLT 2015)*, pages 208–220, Warszawa, Poland, 2015. IPIPAN, IPIPAN. ISBN 978-83-63159-18-4.

Shunsuke Takeno, Masaaki Nagata, and Kazuhide Yamamoto. Empty category detection using path features and distributed case frames. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1335–1340, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1156.

Le Quang Thang, Hiroshi Noji, and Yusuke Miyao. Optimal shift-reduce constituent parsing with structured perceptron. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1534–1544, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1148.

Juliette Thuilier. *Soft constraints and word order in French*. Theses, Université Paris-Diderot - Paris VII, September 2012. URL https://tel.archives-ouvertes.fr/tel-00781228.

Ivan Titov and James Henderson. Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 632–639, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P07-1080.

Masaru Tomita. An efficient augmented-context-free parsing algorithm. *Comput. Linguist.*, 13(1-2):31–46, January 1987. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=26386.26390.

Masaru Tomita. Graph-structured stack and natural language parsing. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 249–257, Buffalo, New York, USA, June 1988. Association for Computational Linguistics. doi: 10.3115/982023.982054. URL http://www.aclweb.org/anthology/P88-1031.

Andreas van Cranenburgh. Efficient parsing with linear context-free rewriting systems. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 460–470, Avignon, France, April 2012. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/E12-1047.

Andreas van Cranenburgh and Rens Bod. Discontinuous parsing with an efficient and accurate DOP model. In *Proceedings of IWPT*, pages 7–16, 2013. URL http://www.illc.uva.nl/LaCo/CLS/papers/iwpt2013parser_final.pdf.

Andreas van Cranenburgh, Remko Scha, and Federico Sangati. Discontinuous data-oriented parsing: A mildly context-sensitive all-fragments grammar. In *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages*, pages 34–44, Dublin, Ireland, October 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W11-3805.

Andreas van Cranenburgh, Remko Scha, and Rens Bod. Data-oriented parsing with discontinuous constituents and function tags. *Journal of Language Modelling*, 4(1):57–111, 2016. URL http://dx.doi.org/10.15398/jlm.v4i1.100.

Yannick Versley. Incorporating semi-supervised features into discontinuous easy-first constituent parsing. *CoRR*, abs/1409.3813, 2014a. URL http://arxiv.org/abs/1409.3813.

Yannick Versley. Experiments with easy-first nonprojective constituent parsing. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 39–53, Dublin, Ireland, August 2014b. Dublin City University. URL http://www.aclweb.org/anthology/W14-6104.

Yannick Versley. Discontinuity reˆ2-visited: A minimalist approach to pseudoprojective constituent parsing. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 58–69, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W16-0907.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the Association for Computational Linguistics*, 1987. URL http://aclanthology.coli.uni-saarland.de/pdf/P/P87/P87-1015.pdf.

Zhiguo Wang, Haitao Mi, and Nianwen Xue. Feature optimization for constituent parsing via neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1138–1147, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1110.

Taro Watanabe and Eiichiro Sumita. 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 (Volume 1: Long Papers)*, pages 1169–1179, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1113.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on*

Natural Language Processing (Volume 1: Long Papers), pages 323–333, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1032.

Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, 2003.

Yuan Zhang and David Weiss. Stack-propagation: Improved representation learning for syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1566, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1147.

Yue Zhang and Stephen Clark. Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 162–171, Paris, France, October 2009. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W09-3825.

Yue Zhang and Joakim Nivre. Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *Proceedings of COLING 2012: Posters*, pages 1391–1400, Mumbai, India, December 2012. The COLING 2012 Organizing Committee. URL http://www.aclweb.org/anthology/C12-2136.

Kai Zhao, James Cross, and Liang Huang. Optimal incremental parsing via best-first dynamic programming. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 758–768, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D13-1071.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1213–1222, Beijing, China, July 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1117.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P13-1043.