UNSUPERVISED METHODS FOR NATURAL LANGUAGE PARSING

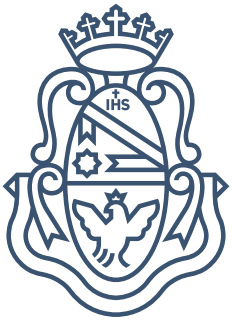
FRANCO M. LUQUE

DIRECTOR: GABRIEL INFANTE-LOPEZ

Presentada ante la Facultad de Matemática, Astronomía y Física como parte de los requerimientos para la obtención del grado de Doctor en Ciencias de la Computación de la

UNIVERSIDAD NACIONAL DE CÓRDOBA

Marzo 2012



©FaMAF — UNC 2012

Franco M. Luque: *Unsupervised Methods for Natural Language Parsing*,
Computer Science, PhD, © Marzo 2012

RESUMEN

En esta tesis estudiamos métodos no supervisados para el análisis sintáctico (*parsing*) de lenguaje natural, tanto desde un punto de vista teórico como desde uno práctico. Desde el acercamiento teórico, investigamos el uso de métodos provenientes del área de inferencia gramatical de lenguajes formales. Estudiamos las propiedades de aprendibilidad teórica de varias clases de gramáticas y su adecuación para representar la sintaxis de lenguaje natural. En el acercamiento práctico, estudiamos la aplicación de nuevos métodos no supervisados para el aprendizaje de autómatas al análisis sintáctico de lenguaje natural.

En la primera parte, estudiamos gramáticas No-Terminalmente Separadas (NTS), una familia de gramáticas libres de contexto que comparten propiedades con la sintaxis de lenguaje natural. Estas gramáticas son de interés para el aprendizaje no supervisado gracias al resultado de aprendibilidad PAC (Probablemente Aproximadamente Correcta) demostrado por Clark (2006), que dice que las gramáticas NTS Inambiguas (UNTS) son polinomialmente PAC-aprendibles a partir de ejemplos positivos. Primero, definimos las gramáticas Inambiguas Débilmente NTS (UWNTS), una generalización de las UNTS que también es polinomialmente PAC aprendible. Luego, estudiamos la adecuación de estas gramáticas para sintaxis de lenguaje natural usando un enfoque experimental. Para esto, desarrollamos métodos para encontrar cotas superiores de la performance *unlabeled* F_1 que cualquier gramática UWNTS puede alcanzar sobre un treebank de lenguaje natural dado. Aplicamos estos métodos sobre subconjunto WSJ10 del Penn Treebank, encontrando una cota para la F_1 de 76.1% para las gramáticas UWNTS sobre el alfabeto de POS tags. Luego, definimos las gramáticas k, l-UNTS$^{\leqslant}$, una jerarquía de clases de gramáticas que generaliza las gramáticas UWNTS agregando la noción de contextos de tamaño fijo (k, l). Probamos luego que las gramáticas k, l-UNTS$^{\leqslant}$ son también polinomialmente PAC-aprendibles a partir de ejemplos positivos, mostrando que éstas pueden ser inyectivamente convertidas a gramáticas UNTS sobre un alfabeto más rico.

En la segunda parte, estudiamos métodos espectrales para el aprendizaje de Split-Head Automaton Grammars (SHAGs) no-determinísticos, un formalismo de estados ocultos para el análisis sintáctico de dependencias. En un SHAG se utiliza un conjunto de autómatas para modelar las características sintácticas de las secuencias de modificadores a izquierda y derecha de las palabras núcleo. En primer lugar, presentamos un algoritmo espectral para aprender los autómatas no-determinísticos (PNFAs) que componen los SHAGs. Los

autómatas se aprenden de manera no supervisada, en contraste con los acercamientos previos a donde hay una estructura determinística dada y sólo se aprenden las probabilidades de las transiciones. Esto permite encontrar estructura específica para cada lenguaje automáticamente. El algoritmo espectral, a diferencia de algoritmos tradicionales como EM, es muy eficiente y no presenta problemas de máximos locales. Luego, presentamos un algoritmo de parsing para SHAGs no-deterministicos que corre en tiempo cúbico, manteniendo así los costos estándar de parsing con SHAGs. Finalmente, realizamos experimentos con el Penn Treebank para comparar el parser no-deterministico con algunos baselines deterministicos, y para comparar el algoritmo espectral con EM. En experimentos no-lexicalizados, vemos que el algoritmo espectral tiene una performance de parsing comparable a la de EM, pero en tiempos de entrenamiento varios órdenes de magnitud menores. En experimentos lexicalizados, definimos tres parsers deterministicos *baseline* diferentes, y observamos una mejora consistente en la precisión de parsing al agregarles a éstos la componente no-determinística. Además, un análisis cualitativo de los autómatas resultantes muestra que efectivamente el algoritmo espectral aprende estructura latente significativa.

ABSTRACT

In this thesis, we study unsupervised methods for natural language parsing, both from a theoretical point of view and from a practical one. Within the theoretical approach, we research the usage of methods coming from the area of grammatical inference of formal languages. We study the theoretical learnability properties of several grammar classes and their adequacy to represent natural language syntax. In the practical approach, we study the application of new unsupervised methods for automata learning to natural language parsing.

In the first part, we study Non-Terminally Separated (NTS) grammars, a family of context-free grammars that share properties with natural language syntax. These grammars are of interest for unsupervised learning thanks to the PAC (Probably Approximately Correct) learnability result proved by Clark (2006), that states that Unambiguous NTS (UNTS) grammars are polynomially PAC-learnable from positive examples. First, we define the Unambiguous Weakly NTS (UWNTS) grammars, a generalization of UNTS grammars that is still polynomially PAC learnable. Then, we study the adequacy of these grammars for natural language syntax using an experimental approach. To do this, we develop methods to find upper bounds for the unlabeled F_1 performance that any UWNTS grammar can achieve over a given natural language treebank. We do experiments with the WSJ10 subset of the Penn Treebank, finding an F_1 bound of 76.1% for the UWNTS grammars over the POS tags alphabet. Then, we define the k,l-UNTS$^{\leqslant}$ grammars, a hierarchy of classes of grammars that generalize the UWNTS grammars adding the notion contexts of fixed size (k,l). We then prove that the k,l-UNTS$^{\leqslant}$ grammars are also polynomially PAC-learnable from positive examples, showing that these grammars can be inyectively converted to Unambiguous NTS grammars over a richer alphabet.

In the second part, we study spectral methods to learn non-deterministic Split-Head Automaton Grammars (SHAGs), a hidden-state formalism for dependency parsing. In a SHAG, a set of automata is used to model the syntactic features of the left and right modifier sequences of the head words. First, we present a spectral algorithm to learn the non-deterministic automata (PNFAs) that compose the SHAGs. The automata are learnt in an unsupervised fashion, in contrast to previous approaches where a deterministic structure is given and only the transition probabilities are learnt. This allows us to find specific structure for each language automatically. The spectral learning algorithm, unlike traditional algorithms such as EM, is very

efficient and do not present local-maximum issues. Then, we present a parsing algorithm for non-deterministic SHAGs that runs in cubic time, hence mantaining the standard parsing costs for SHAGs. Finally, we do experiments with the Penn Treebank to compare the non-deterministic parser with some deterministic baselines, and to compare the spectral algorithm with EM. In unlexicalized experiments, we see that the spectral algorithm has a parsing performance comparable to the one of EM, but with training times several orders of magnitude smaller. In lexicalized experiments, we define three different deterministic baseline parsers, and we see a consistent improvement in parsing accuracy when adding to them the non-deterministic component. Moreover, a qualitative analysis of the resulting automata shows that effectively the spectral algorithm finds significant latent structure.

AGRADECIMIENTOS

A Gabriel Infante-Lopez, por dirigirme, aguantarme tanto tiempo, y finalmente llevarme a puerto.

A los miembros del tribunal, Paula Estrella, José Castaño y Elmer Fernández. Especialmente a Paula, que presidió el tribunal e hizo una revisión muy completa de mi tesis.

A Xavier Carreras y Ariadna Quattoni de la Universitat Politècnica de Catalunya (UPC), con quienes trabajé en la segunda parte de mi tesis, por llevarme a Barcelona y tratarme tan bien. A los integrantes del Departament de Llenguatges i Sistemes Informàtics (LSI) de la UPC, que me hicieron sentir como en casa.

A la comunidad de la FaMAF, y en particular al Grupo de Procesamiento de Lenguaje Natural (PLN), que fueron un entorno ideal para trabajar en la tesis, tanto en cuanto a lo académico como en lo social. A todos los que pasaron por el Grupo de PLN: Gabi Infante, Laura Alonso, Paula Estrella, Martín Domínguez, Ezequiel Orbe, Raúl Fervari, Guillaume Hoffmann, Elsa Tolone, Rafael Carrascosa, Carlos Areces, Luciana Benotti, Romina Altamirano, Matthias Gallé, Sergio Penkale, Pablo Duboue, David Racca y Sasha Vorobyova, entre otros. Al resto de los profesores de computación: Javier Blanco, Damián Barsotti, Nicolás Wolovick, Pedro D'Argenio, Nazareno Aguirre y Daniel Fridlender.

A todas las personas interesantes que conocí en estos años y que de alguna manera tuvieron influencia en este trabajo. Particularmente, a Borja Balle, François Coste, Alexander Clark y Ryo Yoshinaka.

A mis compañeros y amigos del día a día en la FaMAF, y de algunas noches. Matías "Chun" Lee, Juampi Agnelli, Miguel Pagano, Martincito Domínguez, Eze Orbe, Any Zwick, Demetrio Vilela, Francusa Rodríguez, Sergio Giro, Guillaume Hoffmann, Elsa Tolone, Raúl Fervari, Leti Losano, Renato Cherini, Ara Acosta y muchos más.

A mis amigos de FaMAF repartidos por el mundo. Especialmente, a Aldana Gonzalez, Belu Franzoni y Fede Pont.

A mis amigos de Barcelona. Especialmente, a mis compañeros de piso Mario y Iago, y a Solmaz Bagherpour.

A mis viejos, Fanny y Charo, y a mis hermanos, Pau e Isma. A mis amigos del colegio, Agus, Charly B., Charly F., Gabi y Rauli, aunque no se lo merezcan tanto. A la Lu Marchetti, mi compañera durante gran parte de esta etapa.

A la Universidad Nacional de Córdoba, al CONICET y a la Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT) por los cargos, subsidios y becas recibidos.

Y por último, pero no menos importante, al pueblo argentino por su apoyo a través de todos los subsidios y becas recibidos, y al Gobierno Argentino, que desde el año 2003 con las presidencias de Néstor Kirchner y Cristina Fernández ha prestado especial atención al desarrollo de la ciencia y la tecnología en el país. Los científicos argentinos estamos en deuda con nuestro pueblo, y debemos trabajar para que nuestra ciencia sea en pos de un país socialmente justo, económicamente independiente y políticamente soberano.

PUBLICATIONS

Several of the results in this thesis appeared in the following publications:

Franco M. Luque and Gabriel Infante-Lopez. Upper bounds for unsupervised parsing with Unambiguous Non-Terminally Separated grammars. In *Proceedings of CLAGI, 12th EACL*, pages 58–65, 2009.

Franco M. Luque and Gabriel Infante-Lopez. Bounding the maximal parsing performance of Non-Terminally Separated Grammars. In José M. Sempere and Pedro García, editors, *Grammatical Inference: Theoretical Results and Applications*, volume 6339 of *Lecture Notes in Computer Science*, chapter 12, pages 135–147. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010. ICGI 2010 Best Student Paper Award.

Franco M. Luque and Gabriel Infante-Lopez. PAC-learning unambiguous k, l-NTS$^{\leqslant}$ languages. In José M. Sempere and Pedro García, editors, *Grammatical Inference: Theoretical Results and Applications*, volume 6339 of *Lecture Notes in Computer Science*, chapter 11, pages 122–134. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

Franco M. Luque, Ariadna Quattoni, Borja Balle and Xavier Carreras. Spectral learning for non-deterministic dependency parsing. To appear in *Proceedings of the 13th Conference of the European Chapter of the ACL (EACL 2012)*, Avignon, France, April 2012. Association for Computational Linguistics. EACL 2012 Best Paper Award.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

CFG	Context Free Grammar 15
DFA	Deterministic Finite Automata 14
ILP	Integer Linear Programming 37
LAS	Labeled Attachment Score 26
LMF	Left Marked Form 53
MR-UNTS	Maximum R UWNTS 36
MR-UNTS-SC	Maximum R UWNTS-SC 36
MSG	Maximum Score Grammar 36
MW-UNTS	Maximum W UWNTS 36
MW-UNTS-SC	Maximum W UWNTS-SC 36
MWIS	Maximum Weight Independent Set 37
NTS	Non-Terminally Separated 30
PAC	Probably Approximately Correct 23
RCG	Right Contextualized Grammar 54
SHAG	Split-Head Automata Grammar 63
UAS	Unlabeled Attachment Score 26
UNTS	Unambiguous NTS 30
UWNTS	Unambiguous WNTS 31
UWNTS-SC	UWNTS Strongly Compatible 33
WNTS	Weakly Non-Terminally Separated 31
k, l-NTS	k, l-Non-Terminally Separated 46
k, l-NTS$^{\leqslant}$	k, l-Non-Terminally Separated$^{\leqslant}$ 47
k, l-UNTS$^{\leqslant}$	k, l-Unambiguous NTS$^{\leqslant}$ 47

Part I

PRELIMINARIES

1

INTRODUCTION

One of the main problems of Computational Linguistics is the development of natural language parsers, this is, programs that take as input natural language sentences and that output their syntactic analysis. Most Natural Language Processing (NLP) tasks require some kind of syntactic analysis to be performed.

The most studied approach to natural language parsing is based on supervised methods, that is, methods that rely on expert knowledge of some kind. This approach usually comprises the usage of machine learning algorithms over corpora of annotated data, in combination with manually defined rules or features. However, the need for expertise makes the development of parsers expensive and only possible on languages for which experts are available.

In the last years, there has been an increasing interest in the development of semi-supervised and unsupervised methods for parsing. In these approaches, machine learning algorithms are used to learn unnanotated aspects of natural language data. Fully unsupervised methods learn syntax from plain natural language text [4, 7, 32, 49, 55]. Semi-supervised methods combine supervised methods over annotated data with unsupervised methods to help leverage the limited information present in the data [34, 52]. Unsupervised methods are useful to alleviate the need of experts and annotated data to develop highly performant parsers.

The main subject of this thesis is the study of unsupervised methods for natural language parsing. It is divided in two parts, the first one dealing with theoretical aspects, and the second dealing with practical ones.

The first part studies the properties of formal models that can be used to unsupervisedly learn natural language syntax. Our approach is to map the natural languages to formal languages. Then, the unsupervised learning problem is mapped to the theoretical problem of learnability from positive examples, that is studied in the Grammatical Inference field. Along with theoretical learnability, we are interested in the linguistic property of adequacy, that is the capability of a formal model to express natural language.

We first study the adequacy of Unambiguous Weakly Non-Terminally Separated (UWNTS) grammars, a subclass of context-free gram-

mars that has interesting formal learnability properties. The properties of UWNTS grammars suggest that they can express natural language syntax in some degree [13]. We propose an experimental approximation to the problem, where the adequacy is measured with respect to a concrete syntactic corpus of a particular language, and in terms of standard Machine Learning quality measures.

After studying UWNTS adequacy and concluding that it is not expressive enough, we define a more general family of grammars, the hierarchy of k, l-NTS$^{\leqslant}$ grammars. We then prove that the Unambiguous versions of these grammars also have theoretical learnablility from positive examples, and observe that they are more adequate than UWNTS grammars.

The second part of this thesis is devoted to another unsupervised learning problem, but in the context of dependency parsing. Annotated corpora usually lack the explicit annotation of several linguistic phenomena. We study the usage of an unsupervised method to learn latent phenomena in annotated data, alleviating the need for experts to do language specific explicit feature engineering.

We first consider the problem of modeling the distribution of modifier sequences. The classic approach is to use probabilistic deterministic automata, with the automata structure manually engineered, and its probabilities learnt using supervised methods. Instead, we propose the usage of probabilistic non-deterministic automata, and an unsupervised method to learn them with no a priori structural bias. The learning method is the novel spectral algorithm from Hsu et. al. [28]. Unlike traditional algorithms such as Expectation Maximization [20], this algorithm is very efficient and does not have local optima issues. To evaluate this approach, we do experiments with training modifier sequences of the dependency version of the Penn Treebank [37]. We train an unlexicalized automata for each head and direction, and then analyze the results using quantitative measures and qualitative analysis, comparing them with deterministic baselines and EM learnt automata.

After modeling unlexicalized modifier sequences, we develop parsing models using them. These parsing models are non-deterministic versions of the Split-Head Automata Grammars (SHAGs) [22]. Our parsing algorithm runs in cubic time, maintaing the standard complexity of dependency parsing. To test our models, we first do experiments in multilingual unlexicalized parsing, showing that in all cases we outperform deterministic and EM baselines. We then turn to lexicalized parsing of the WSJ corpus, developing three different deterministic baselines and showing that adding the unlexicalized non-deterministic component to them consistently leads to an improvement in the parsing performance.

In the following two sections of this chapter we present in more detail the motivations and contributions of the two parts of this thesis.

1.1 NON-TERMINALLY SEPARATED GRAMMARS

Grammatical Inference (GI) is a field that studies the problem of learning a hidden formal language using some sort of available evidence [19]. Depending on the learning setting, the evidence can be positive or negative examples of the language, oracles answering queries about the language, etc. The case of learning from positive examples is analog to the natural language unsupervised parsing problem.

In GI, it is said that a class of languages is learnable if there exists an algorithm that always finds the hidden language through evidence. There are several learning models, each one with its own formal definition of learnability. One of them is the Probably Approximately Correct (PAC) learning model [31, 5]. In PAC learning, it is assumed that the evidence is sampled according to an unknown distribution. It is said that an algorithm PAC-learns a class of languages if the language it returns is closer to the target when it has access to more evidence.[1] If the algorithm runs in polynomial time and number of samples, it is said that it polynomially PAC-learns.

In the first part of this thesis, we study natural language as a formal language. Our goal is to find PAC-learnable language classes that are suitable to model natural language. This way, we could use these classes and their corresponding learning algorithms in unsupervised parsing systems.

In linguistics, the capability of a grammar class to express natural language is called adequacy. There are two types of adequacy. Weak adequacy is the capacity to describe the set of grammatical natural langauge sentences. Strong adequacy is, in addition, the capacity to assign to the sentences the correct or expected structural descriptions, from a linguistic point of view. In the eye of unsupervised parsing, the property of our interest is strong adequacy.

Linguists have discussed extensively the adequacy of different language classes. We give a brief overview of this discussion in section 2.3.2. The reference has mainly been the Chomsky hierarchy, that is comprised of four nested language classes: the regular languages, the context-free languages, the context-sensitive languages and the recursively enumerable languages. Today, there is a general agreement that natural language is context-sensitive, both in a weakly and in a strongly adequacy sense. However, as far as we know, there are no PAC learnability results from positive examples for context sensitive classes of languages. Moreover, learning context free grammars from positive examples is a hard problem, for which there are not known polynomial PAC-learning results [18]. So, we must resort to subclasses of context free grammars, at the cost of knowing that full adequacy for natural language will not be possible. Fortunately,

1 For a more formal definition, see section 2.4.2.

there are some known context free languages that are polynomially PAC learnable and have properties suggesting that they may be adequate for natural language in some degree.

In the following section we introduce the grammar classes studied in the first part of this thesis and address their properties, mainly focusing on learnability and adequacy.

1.1.1 *Weakly Non Terminally Separated Grammars*

A subclass of context-free grammars that is relevant to natural language is the class of Non Terminally Separated (NTS) grammars [13, 14]. The characteristic property of NTS grammars is that the languages generated by the non-terminals are all disjoint between themselves. In other words, given a string, it can only be generated by at most one non-terminal of the grammar. In natural language there is a similar notion: the sustitutability property only hold between constituents of the same type [14].

In [13, 14], it is argued that natural language is close to be in the class of Non-Terminally Separated (NTS) languages, and in [13] it is proved that Unambiguous NTS (UNTS) languages are PAC-learnable in polynomial time. However, we will see that UNTS grammars are much less expressive than NTS grammars and that they can not admit elemental sets of natural language sentences.

In order to shorten the gap between UNTS and NTS, we present a slight generalization of UNTS grammars called the Unambiguous Weakly NTS (UWNTS) grammars. In contrast with UNTS, UWNTS grammars are general enough to admit any finite language, but at the same time preserving every other aspect of UNTS grammars, including PAC-learnability in polynomial time.

We study the strong adequacy of UWNTS grammars, this is, its potential to model natural language syntax. Our approach to study UWNTS grammars is in the context of the unsupervised parsing problem, using the standard quantitative evaluation over gold treebanks. We do not aim at developing a learning algorithm that returns a UWNTS grammar, because the resulting evaluation will depend on the algorithm. Instead, our aim is to find upper bounds for the achievable F_1 performance of *all* the UWNTS grammars over a given gold treebank, regardless of the learning algorithm and the training material used to induce the grammars. Our bounds should only depend on the gold treebank that is going to be used for evaluation.

We see that the properties of UWNTS grammars and its variants allows us to develop a method to compute, given a gold treebank, upper bounds for the achievable F_1 quality measure for every grammar that parses the sentences of the treebank. Our method involves an optimization problem that we prove to be NP-Hard. However, some instances are quickly solvable using specialized software. In

this work, we compute upper bounds for UWNTS grammars over a treebank of short English sentences of POS tags.

In general, knowing an upper bound for a model is useful in at least two senses. First, if it is lower than the performance we want to achieve, we should consider not using that model at all. Second, if it is not low, it might be useful for the development of a learning algorithm, given that it can be assessed how close the performance of the algorithm is to the upper bound.

1.1.2 k, l-*NTS*$^{\leqslant}$ *Grammars*

Our experimental results on UWNTS grammars' adequacy leads us to search for more expressive, yet learnable, classes of languages.

In this thesis, we present two hierarchies of context-free grammars: The k, l-NTS grammars and the k, l-NTS$^{\leqslant}$ grammars. The k, l-NTS grammar hierarchy generalizes the concept of Non-Terminally Separated (NTS) grammar by adding a fixed size context to the constituents. The k, l-NTS$^{\leqslant}$ grammars are k, l-NTS grammars that also consider the boundaries of sentences as possible contexts.

k, l-NTS grammars express the idea of letting the contexts to influence the decision of considering the enclosed strings as constituents. This idea has been successfully applied to natural language and in particular to unsupervised learning, for instance in the Constituent Context Model by Klein and Manning [32], where a context of size $(k, l) = (1, 1)$ is used. Their idea of adding starting and ending markers to the sentences corresponds exactly to our definition of k, l-NTS$^{\leqslant}$.

We prove that Unambiguous k, l-NTS$^{\leqslant}$ (k, l-UNTS$^{\leqslant}$) grammars can be converted to plain old UNTS grammars over a richer alphabet. Using this and the result of polynomial PAC-learnability with positive data of UNTS grammars proved by Clark (2006), we prove that k, l-UNTS$^{\leqslant}$ languages are also PAC-learnable under the same conditions.

1.2 NON-DETERMINISTIC DEPENDENCY PARSING

Natural Language has lots of phenomena that usually can not be found explicitly annotated in corpora annotations. This is due in part to expressiveness limitations of the annotation scheme, and to assumptions of the linguistic theory behind it. For instance, the Penn Treebank lacks agreement annotation, as in the example of Fig. 1.1.a, and verb subcategory annotation.

The lack of detail in annotation imposes an important limit to parsing models with strong local independence assumptions. This is the case, for instance, of naive Probabilistic Context Free Grammar (PCFG) estimation with rules extracted from the Penn Treebank with no prior processing. As shown in Fig. 1.1.b, direct extraction of rules

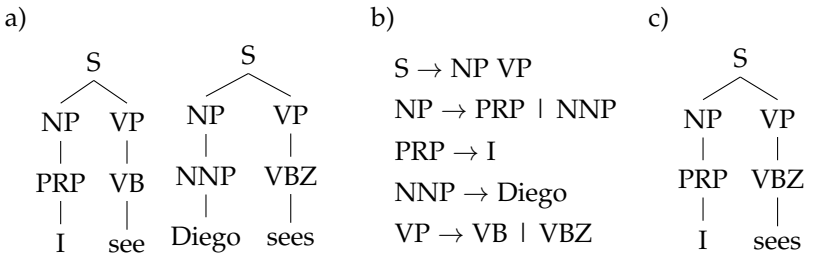


Figure 1.1: Example of unannotated non-local agreement phenomena. a) Two training trees. b) Extracted rules. c) Incorrectly generated sentence.

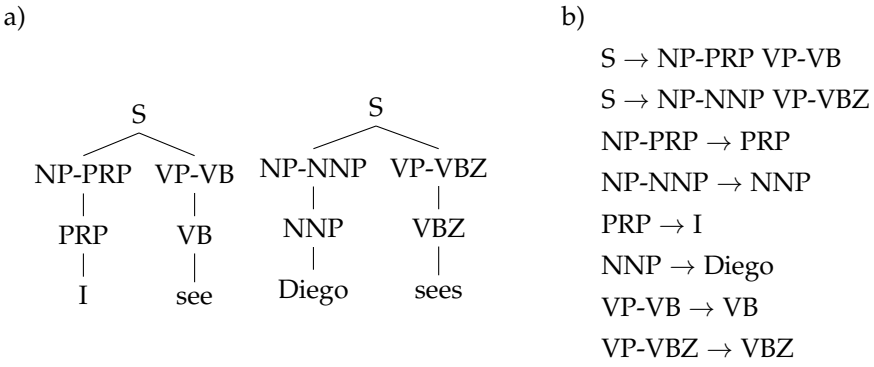


Figure 1.2: Fixing the agreement problem of Fig. 1.1 with higher order features. a) Decorated training trees. b) New rules.

allows the construction of ungrammatical sentences such as the one in Fig. 1.1.c.

To overcome this limitation, research resorted first to higher order models. These models drop some independence assumptions, letting the non-local context of a rule to influence it. For instance, the agreement problem of Fig. 1.1 can be fixed by decorating the NP and VP non-terminals with their child non-terminals, as shown in Fig. 1.2.

However, in higher order models the number of parameters grow significantly. To avoid data-sparsity and overfitting problems, yet maintaining the benefits of high order, there must be a careful selection of which part of the context to take into account. This is called feature engineering, a work that has to be done manually, and that requires some linguistic expertise. Feature engineering is a demanding task, and usually the decisions are not equally appliable to every language.

Nowadays, state of the art parsers are heavily based on explicit feature engineering [17, 21, 40, 10, 38, 33]. For instance, in Collins parser [17], the rules are enriched with several non-local features, such as parent and sibling nodes, lexical heads and subcategorization frames.

As an alternative to explicit feature engineering, a new trend in parsing has been the development of latent variable models, together

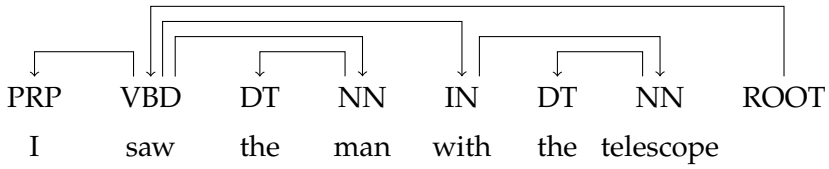


Figure 1.3: Example of a dependency structure.

Head	Direction	Modifiers
ROOT	LEFT	saw/VBD END
saw/VBD	LEFT	I/PRP END
saw/VBD	RIGHT	man/NN with/IN END
I/PRP	LEFT	END
I/PRP	RIGHT	END
man/NN	LEFT	the/DT END
man/NN	RIGHT	END
...		

Table 1.1: Modifier sequences for the dependency tree of Fig. 1.3 in head-outwards order and with the END marker.

with unsupervised learning algorithms to learn them [39, 46, 42]. Most of them use PCFGs with augmented non-terminals. Latent variables allow the encoding of high order unannotated phenomena. If they are correctly learnt, they would alleviate the need for feature engineering. They would even be able to find unexpected but significant behaviour.

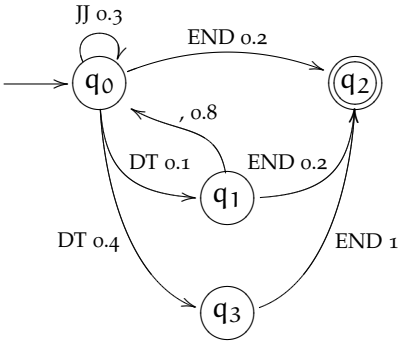
The main drawback of latent variable models is that they are difficult to learn. The most commonly used learning algorithm is the Expectation Maximization algorithm [20]. It has also been used in combination with heuristics and greedy algorithms, such as variable splitting and merging. To the best of our knowledge, all the formulations of the learning problem have been non-convex , leading to local optima issues and expensive training [39, 46, 42].

In the second part of this thesis, we propose a new latent variable model in the frame of dependency parsing. The main contribution of our work is the application of spectral learning, a novel unsupervised learning algorithm that overcomes the drawbacks of traditional algorithms. In the following sections we introduce both the algorithm and the latent variable model.

1.2.1 *Modifier Sequences and the Spectral Algorithm*

A dependency structure for a sentence is a tree where the nodes are the words of the sentence, and the arcs define head-modifier relatioships. In this work, we decompose the dependency tree into sets of modifier sequences. Each word of the sentence defines two modifier

a)



b)

$$\pi = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, A^{JJ} = \begin{bmatrix} 0.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$A^{DT} = \begin{bmatrix} 0 & 0.1 & 0.4 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, A^{END} = \begin{bmatrix} 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 1.4: a) Example of an unlexicalized PNFA generating left modifiers for the head `NN`. b) Matrix arrangement of the PNFA probabilities.

sequences, one for each direction. Fig. 1.3 is an example of a dependency tree, and Tab. 1.1 shows the modifier sequence decomposition.

Split-Head Automaton Grammars (SHAGs) use modifier sequence decomposition to define dependency models [22]. In this thesis, we propose to use probabilistic non-deterministic finite automata (PNFA) [19] to model the modifier sequence distribution. PNFAs are probabilistic automata where one can jump from a state to several different states using the same symbol. Fig. 1.4.a shows an example of a PNFA, where q_0 is the starting state and q_2 is the final state. Here, there is non-determinism in the state q_0 with the symbol DT, that transitions to the states q_1 and q_3. In PNFAs, the probability of a modifier sequence is the sum of the probabilites of all the possible paths of states that the sequence can traverse.

An important property of PNFAs is that their probabilities can be arranged intro matrices. This representation is called an *operator model*. For instance, the operator model for the previous example is shown in Fig. 1.4b. Here, π is the probability vector of initial states, and each A^a matrix contains in its i, j entry the probability of going from state q_i to state q_j generating the symbol a.

With the operator model, the probability of a sequence can be defined as a matrix product. For instance, in the example, the probability of the sequence "JJ JJ DT END" will be

$$\begin{aligned} P(\text{JJ JJ DT END}) &= \overrightarrow{1} A^{\text{END}} A^{\text{DT}} A^{\text{JJ}} A^{\text{JJ}} \pi \\ &= 0.3 \times 0.3 \times 0.1 \times 0.2 + 0.3 \times 0.3 \times 0.4 \times 1.0 \end{aligned}$$

The operator model allows us to use the spectral learning algorithm from Hsu et. al. [28]. The spectral algorithm has two important advantages over previous algorithms. First, that it is not subject to local optima issues, and second, that its complexity does not depend directly on the number of training instances. Indeed, it is a very quick algorithm that only does Singular Value Decomposition and other simple matrix operations. The input of the algorithm is a fixed set of matrices with unigram, bigram and trigram statistics, and the output is an operator model.

1.2.2 *Non-Deterministic Split Head Automata*

Split Head Automaton Grammars (SHAGs) combine deterministic models for head-modifier sequences to define dependency grammars [22]. SHAGs have proved useful for dependency parsing, and have the particular advantage of having a cubic time Maximum a Posteriori (MAP) parsing algorithm.

In our work, we adapt the SHAG framework to non-deterministic models. However, MAP parsing is intractable for the non-deterministic case [43]. To solve the parsing problem, we use the inside-outside algorithm to compute marginal probabilites for individual dependencies, and then choose the dependency structure that maximizes this probability [35].

2

BACKGROUND

2.1 FORMAL LANGUAGES

Formal Language theory studies the mathematical notion of language. It is very useful to the study of mathematical logic, computer science and linguistics. For instance, it has been used by Gödel in his completeness and incompleteness theorems, by Turing in his computability results, and by Chomsky in his linguistic theories.

A *formal language* is a possibly infinite set of words, where each word is a finite sequence of symbols drawn from a finite alphabet Σ. The set of all possible words that can be formed with a given alphabet is denoted Σ^*.

As sets, languages support union, intersection, difference, and complementation (w.r.t. Σ^*). They also support some additional operations, such as concatenation, defined as $L_1 L_2 = \{w_1 w_2 | w_1 \in L_1 \text{ and } w_2 \in L_2\}$.

In formal language theory, the languages are organized in classes, that are sets of languages that share some distinguishing property. For instance, the set of finite languages is a class of languages.

Formal languages can be defined using several kinds of formal devices. Each device also defines a class of languages, namely, the whole set of languages that can be defined using that formalism. Chomsky studied several kind of grammars as language devices from a linguistic perspective. This lead to the definition of a hierarchy of formal language classes called the *Chomsky hierarchy*, which is depicted in Table 2.1. In the following sections we introduce the two less expressive formalisms of this hierarchy, that are the most relevant for this thesis.

Type	Languages	Machine
Type-0	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Linear Bounded Automata
Type-2	Context-free	Context-free Grammar
Type-3	Regular	DFA

Table 2.1: The Chomsky hierarchy.

2.1.1 *Regular Languages and Deterministic Finite Automaton*

The *regular languages* is a class of languages that supports an elementary form of recursion or iteration in the generation of words. So, they include some infinite languages in addition to all the finite languages.

There are several mechanisms to define languages that have a generation power equivalent to the class of regular languages. Among these, we find the regular expressions and several kinds of automata, such as the deterministic and non-deterministic finite automata. In this section we will pay special attention to deterministic automata.

A *Deterministic Finite Automata* (DFA) is a finite state machine that emits a symbol each time a transition from one state to another is performed. It always starts at the same state called the initial state, and can optionally stop in those states that are marked as final or accepting states. Each state has exactly one transition to another state for each element of the alphabet. A graphical example of a DFA is shown in Fig. 2.3. Here, the missing transitions are assumed to go to an absorving non-final state.

A DFA defines a language that is the set of words that can be generated with the symbols emitted in a traversal of the automata, starting from the initial state and ending in a final state. It can also be seen as a machine that accepts or rejects words, depending on whether the traversal of the automata using the symbols of the word ends in a final state or not.

Formally, a DFA is a tuple $A = \langle \Sigma, Q, q_0, \delta, F \rangle$. Σ is the alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final or accepting states, and $\delta : Q \times \Sigma \to Q$ is the transition function. The language defined by A is

$$L(A) = \{a_0 \dots a_n | \exists q_1, \dots, q_n : q_n \in F \text{ and } \forall i, 0 \leqslant i < n, \delta(q_i, a_i) = q_{i+1}\}.$$

2.1.2 *Context-Free Languages and Grammars*

The *context-free languages* are languages that can be generated by a context free-grammar. Context free-grammars allow more complex recursive structures than the ones present in regular languages, such as the so-called central recursion. However, they are not as expressive as context-sensitive grammars.

A context-free grammar is defined by a set of non-terminals, one of which is the initial non-terminal, and a set of rules that specify the ways in which these non-terminals can be replaced by longer strings of terminals and non-terminals. A derivation is a sequence of application of rules in a string of terminals and non-terminals. So, the language defined by a context-free grammar is the set of terminal strings that can be derived from the initial non-terminal.

Context-free grammars provide a way to define natural language grammars. For instance, Fig. 2.1 shows an example of a context-free grammar for a simplified version of the English language, and Fig. 2.2 shows examples of derivation trees for this grammar.

Formally, a *Context Free Grammar* (CFG) is a tuple $G = \langle \Sigma, N, S, P \rangle$, where Σ is the terminal alphabet, N is the set of non-terminals, $S \in N$ is the initial non-terminal and $P \subseteq N \times (\Sigma \cup N)^+$ is the set of productions or rules.

We use letters from the beginning of the alphabet $a, b, c, \ldots$ to represent elements of Σ, from the end of the alphabet $r, s, t, u, v, \ldots$ to represent elements of Σ^*, and Greek letters to represent elements of $(\Sigma \cup N)^*$. We write the rules using the form $X \to \alpha$.

A CFG G defines a relation $\Rightarrow_G$ that results from the application of a rule in a string. This is, $\alpha X \beta \Rightarrow_G \alpha\gamma\beta$ iff $X \to \gamma \in P$. So, the *derivation* relation is the transitive closure of this relation $\Rightarrow_G$, and it is denoted $\overset{*}{\Rightarrow}_G$. The language generated by G is

$$L(G) = \{s \in \Sigma^* | S \overset{*}{\Rightarrow}_G s\}.$$

2.2 LINGUISTICS BASICS

Linguistics is the study of human language. Although it is a very ancient discipline and a very broad field, our concern will be mainly on modern linguistics and language syntax. In this section we introduce some of the basic notions that constitute modern language syntactic theories.

2.2.1 *Phrase Structure Grammars*

Phrase structure grammars were first introduced by Noam Chomsky in the mid 1950's, as an approach towards a general linguistic theory [36, 11, 12]. They come from the formalization of the classic notion of constituent. A *constituent* is a contiguous segment of words that plays a specific syntactic role in a sentence. The phrase structure of a sentence is composed of constituents organized in a tree.

A *phrase structure grammar* is defined by a set of rules that can be used in the generation of sentences. These rules describe the way in which the different types of constituents can be nested until the generation of an entire sentence. The resulting structure is a *constituent tree*, a labeled tree where the internal nodes are the constituents and the leaves are the words of the sentence.

An example of a simple phrase structure grammar for a natural language like English is the context-free grammar shown in Fig. 2.1. Fig. 2.2 shows examples of constituent trees that can be derived from this grammar. The top level constituent of a sentence is labeled S. Declarative sentences are usually composed of a subject and a

S → NP VP	Prn → I | ...
NP → Prn | Det NN | NP PP	Det → the | ...
NN → Adj NN | Noun	Noun → man | telescope | ...
VP → Verb | Verb NP | VP PP	Verb → saw | ...
PP → Prep NP	Prep → with | ...

Figure 2.1: A toy natural language Context-Free Grammar.

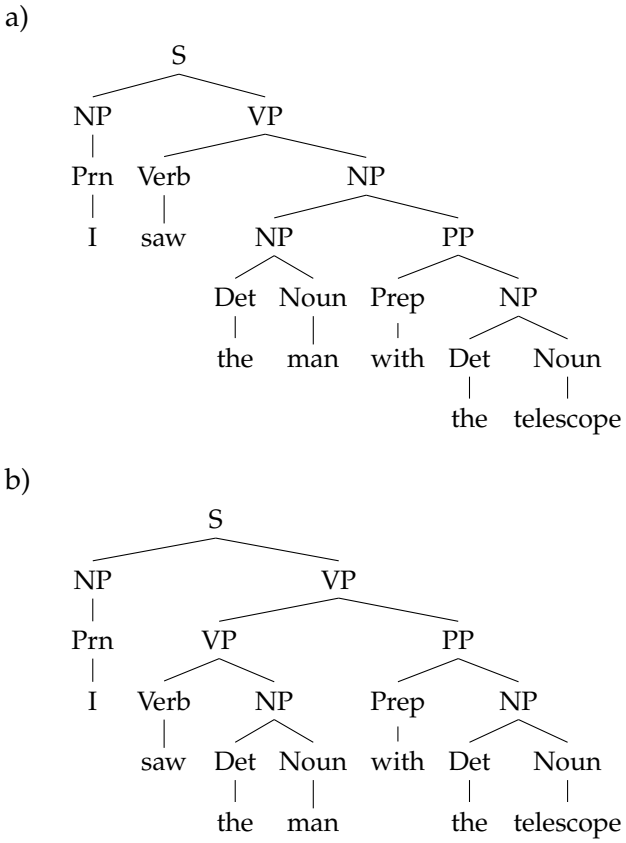


Figure 2.2: Two possible parses for the sentence "I saw the man with the telescope", using the ambiguous grammar of Fig. 2.1.

Tag	Description
S	Simple declarative clause
SBAR	Clause introduced by a subordinating conjunction
SBARQ	Direct question introduced by a wh-word or a wh-phrase
SINV	Inverted declarative sentence
SQ	Inverted yes/no question, or main clause of a wh-question
ADJP	Adjective Phrase
ADVP	Adverb Phrase
CONJP	Conjunction Phrase
FRAG	Fragment
INTJ	Interjection
LST	List marker
NAC	Not a Constituent
NP	Noun Phrase
NX	Head of the NP (used within certain complex NPs)
PP	Prepositional Phrase
PRN	Parenthetical
PRT	Particle
QP	Quantifier Phrase (used within NP)
RRC	Reduced Relative Clause
UCP	Unlike Coordinated Phrase
VP	Verbal Phrase
WHADJP	Wh-adjective Phrase
WHAVP	Wh-adverb Phrase
WHNP	Wh-noun Phrase
WHPP	Wh-prepositional Phrase
X	Unknown, uncertain, or unbracketable

Table 2.2: Penn Treebank Constituents tag set.

predicate. Subjects have the Noun Phrase syntactic category, labeled `NP`. Predicates are Verbal Phrases (`VP`) and may contain an object, of type `NP`, and complements, that are Prepositional Phrases (`PP`). Noun phrases have their internal structure and may have other NPs nested inside, creating recursive structures. They can also have modifiers such as PPs. There are several other syntactic categories. A summary of the categories for the English language used in the Penn Treebank corpus is shown in Table 2.2.

The pre-terminals in the trees classify the words into categories. These categories, called *part-of-speech* (POS), group the words that share some syntactical behaviour. Verbs, nouns, pronouns, prepositions, determiners and adjectives are examples of POS categories. A summary of the POS categories for the English language used in the Penn Treebank corpus is shown in Table 2.3.

Because of the contiguity of the constituents, word order plays an important role in phrase structure grammars. In fact, they are more suitable for languages with a restrictive word ordering, such as English or Spanish, rather than for free word order languages, such as Czech or Turkish, where dependency structures are preferred.

2.2.2 *Dependency Grammars*

Dependency grammars are a different kind of syntactic theory that was introduced in its modern form by Lucien Tesnière, published posthumously in the late 1950's [53]. They rely mainly on the *dependency* relationship, a relationship that holds between pairs of words in a sentence. In a dependency, one of the words is the head and the other is the dependent or modifier.

Tag	Description	Tag	Description
CC	Coordinating conjunction	PRP$	Possessive pronoun
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Adverb, comparative
EX	Existential there	RBS	Adverb, superlative
FW	Foreign word	RP	Particle
IN	Preposition/subord. conjunction	SYM	Symbol
JJ	Adjective	TO	to
JJR	Adjective, comparative	UH	Interjection
JJS	Adjective, superlative	VB	Verb, base form
LS	List item marker	VBD	Verb, past tense
MD	Modal	VBG	Verb, gerund or present participle
NN	Noun, singular or mass	VBN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non-3rd person sing. present
NNP	Proper noun, singular	VBZ	Verb, 3rd person sing. present
NNPS	Proper noun, plural	WDT	Wh-determiner
PDT	Predeterminer	WP	Wh-pronoun
POS	Possessive ending	WP$	Possessive wh-pronoun
PRP	Personal pronoun	WRB	Wh-adverb

Table 2.3: The Penn Treebank POS tag set.

A *dependency grammar* is a grammar that defines rules for the attachment of head and dependents in a natural language sentence. The resulting structure is a *dependency tree*, a tree where the nodes are the words of the sentence and the arcs represent dependency relations.

An example of a dependency structure is shown in Fig. 1.3. The root is the head of the whole sentence, that in a declarative sentence is usually the verb. It has as direct dependents the head of the subject and the head of the objects if it is a transitive verb. It may also have one or more complements, such as adverbs or prepositions. Nouns can have adjectives as depedendents, to the left in the case of English, and complements such as prepositions. Prepositions, in turn, have their own dependents. The dependency structures end at leaf words, that is, words that have no dependents in the sentence.

Dependency grammars have very different properties than phrase structure grammars. As there is no explicit notion of constituent, the grammars are less restrictive in word ordering and, therefore, more suitable for free word order languages. Dependency trees are also simpler and flatter than constituent trees.

Computational linguistics have brought much more attention to dependency grammars in the recent years. There has been also an important development of hybrid structures that combine phrase and dependency structures, such as the lexicalized phrase structure grammars.

2.3 NATURAL LANGUAGE AS A FORMAL LANGUAGE

The approach of studying natural language as a formal language requires an important number of methodological assumptions. Most of them date to the origins of modern linguistics and are in general still valid in the main branches of the Computational Linguistics field. They are discussed, among others, by Chomsky in [11, 12]. First, there is an idealization of the human linguistic capabilities, where it is assumed that there are no memory limitations, distractions or errors. Moreover, languages are assumed to have a static behaviour in time and space: They do not change or evolve, and they do not variate along the community of speakers.

A second set of assumptions is more specific to the mapping of natural languages to formal languages. The basic elements of a formal language are the symbols, that are drawn from a finite set called the alphabet of the language. In our case, the words of a natural language, or sometimes the word categories, are modeled as the alphabet. A formal language is a possibly infinite set of strings, where each string is a finite sequences of symbols. So, the natural language sentences are mapped to formal language strings, and the entire set of grammatical sentences of a natural language is mapped to a formal language.

2.3.1 *Weak Adequacy vs. Strong Adequacy*

The adequacy of a linguistic theory is how appropiate it is to express natural language phenomena. The traditional approach for discussing adequacy is an extensive analysis of the properties of well known natural languages and how a linguistic theory can express them.

An important distinction must be made when considering adequacy. *Weak adequacy* is the capability of a grammar to generate the set of grammatical sentences of a natural language. As there may be several different grammars generating the same set of sentences, it is not guaranteed that a weakly adequate grammar gives the expected structural descriptions to the sentences, which is called *strong adequacy*.

The linguistic concepts of weak and strong adequacy have their counterparts in formal language theory. Here, we say that two grammars are weakly equivalent if they define the same language and that they are strongly equivalent if they give the same structural descriptions to the language elements.

The importance of this distinction can be appreciated with the following example. Consider the grammar in Fig. 2.1, that corresponds to a very simplified version of the English language. Even though this is a CFG, the generated language is regular because it can also

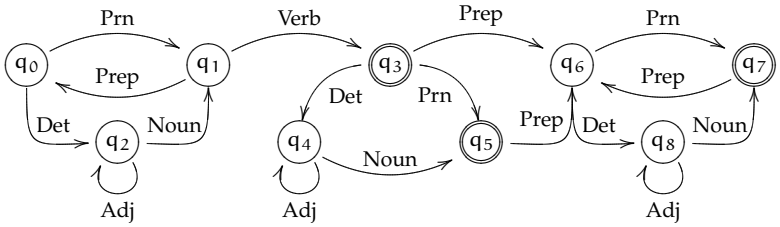


Figure 2.3: A DFA version for the toy natural language of Fig. 2.1.

be generated by the DFA of Fig. 2.3. Then, the DFA representation is weakly adequate in order to express the language. On the other hand, in terms of strong adequacy, the DFA representation is not admissible because it is not able to express some syntactic properties. Namely, it is not able to express the ambiguity in the prepositional phrase attachment. For instance, in the sentence "I saw the man with the telescope", the strong adequacy requires a way to tell if the prepositional phrase is modifying the verb or the object. The CFG does this distinction, allowing two different structures for the sentence, as shown in Fig. 2.2. There is no way to do this with the DFA representation, so it is not strongly adequate.

2.3.2 *Natural Language into the Chomsky Hierarchy*

The Chomksy hierachy is a good starting point to determine where natural language fits into the world of formal languages. In this section, we quickly review the arguments coming from known natural languages that have been given to adress this issue. More detailed reviews can be found in [51], [29] and [24].

The first and easiest question to answer is if natural languages are regular. There are several examples that show that it is not, not even in a weakly adequate way. In [11], p. 22, Chomsky derives some examples from English that can be mapped to the well known non-regular languages $\{a^n b^n | n > 0\}$ and $\{xy | x \in \{a, b\}^*, y = reverse(x)\}$.

The next question is about the adequacy of context-free languages. In terms of strong adequacy, Chomsky argues that context-free grammars can only clumsily accomodate several simple syntactic structures of English ([11], pp. 34–43). A more conclusive argument can be found in the Dutch language, that has unbounded cross-serial dependencies [8]. This construction can be mapped to a language of the form $\{a^n b^n\}$, with a syntactic structure that associates the elements in an intercalated fashion, as shown in Fig. 2.4 a). Even though context-free grammars can generate languages of the form $\{a^n b^n\}$, it can only be done using central recursive context-free rules, forcing a center-outwards association, as shown in Fig. 2.4 b).

The question of the context-free weak adequacy has been more elusive. Since Chomsky proposed this question in 1965, there were sev-

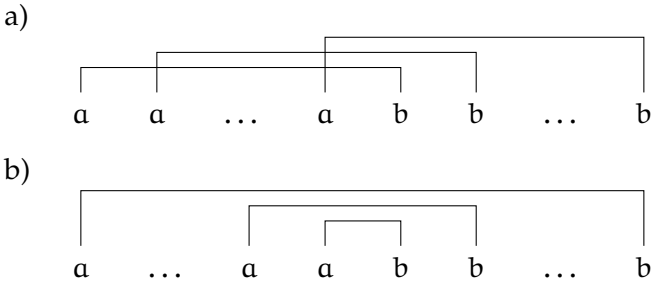


Figure 2.4: a) Structure of cross-serial dependencies in the Dutch language. b) Structure forced by a context-free grammar.

eral attempts to prove weak inadequacy, but they were all proven incorrect by Pullum and Gazdar in [48]. It was not until 1985 that a strong argument against weak adequacy of context-free languages was given, by Shieber in [50]. Shieber constructs a counterexample from the Swiss German dialect, that can be mapped to a non-context free language of the form $\{wa^m b^n x c^m d^n y\}$. The counterexample uses Swiss German cross-serial dependencies, as in Dutch, in combination with case-marking on direct and indirect objects (accusative case and dative case, respectively).

Going further in the Chomsky hierarchy, we find the context-sensitive languages. At least in terms of weak adequacy, context-sensitive languages can describe Swiss German cross-serial dependencies and other examples of context-free inadequacy. However, context-sensitive languages are inadequate due to performance issues. The recognition problem for context-sensitive languages, this is, answering if $w \in L(G)$ for a given context-sensitive grammar G and a string w, is PSPACE-complete [27]. This means that there is no efficient algorithm to determine if a sentence is grammatical for context-sensitive languages in general, a task that humans can do quickly for natural languages. To cope with this problem, a new class of languages has been proposed, the mildly context-sensitive languages, that lie between the context-free and the contex-sensitive languages. Mildly context-sensitive languages can describe several non-context free phenomena, at the same time having polynomial algorithms for recognition and parsing. Nevertheless, there is some evidence that mildly context-sensitive languages may not be weakly adequate for some natural languages [24].

There are even more expressive formalisms that has been proposed as adequate for natural languages, such as the unification grammars [24], that can generate the entire class of recursively enumerable languages, at the top of the Chomksy hierachy. Even if recursively enumerable languages seem to be weakly adequate, the recognition problem for them is undecidable.

2.4 GRAMMATICAL INFERENCE

Grammatical Inference (GI) is a general research area that studies the inference or learning of formal language grammars using some kind of available evidence about the language [19]. The main concern of GI is a formal approach to this problem, where mathematical definitions are given for the learning setting and for the correctness of the learning algorithms. However, the GI area also has room for more informal approaches, such as heuristical algorithms and empirical learning settings and correctness definitions.

GI is of interest to any discipline that deals with sequential data and the problem of finding patterns on it. So, it is closely related to Computational Linguistics and Computational Biology, among other fields. Some results from GI have also been used as arguments in cognitive sciences' problem of language acquisition. Particularly, Gold's theorem, that proves unlearnability of superfinite languages from positive examples, was used in favour of Chomsky's Poverty of the Stimulus hypothesis [25].

The general GI learning model assumes that the language to be learnt belongs to a known class of languages $\mathcal{L}$. It also assumes that there is a representation for the languages in the class, this is, a class of grammars $\mathcal{G}$ that can generate all the languages. The learning algorithm is defined as a function that takes a finite number of evidence elements and returns a hypothesis, this is, the grammar that is believed to generate the target language.

A key element of the learning model is the kind of information available to the learner, and how this evidence relates to the target language. The two main learning models are Identification in the Limit and Probably Approximately Correct learning. We describe them in more detail in the following sections.

2.4.1 *Identification in the Limit*

Identification in the Limit (IIL) is one of the first approaches towards a formalization of learnability. It was introduced by Gold in 1967 [25]. Here, we follow the notation from [19].

In the IIL model, the evidence is presented to the learner as an enumeration of elements. At each step, a new evidence element is presented to the learning algorithm and the algorithm returns a new hypothesis. The algorithm succeeds if, at some moment, it converges to the hypothesis that generates the target language.

The kind of evidence that is presented leads to completely different learning problems. For instance, in the *learning from text* setting, the evidence are elements of the target language. In the *learning from an informant* setting, the evidence are positive and negative examples of the target language.

The presentation of the evidence is formalized as an enumeration function $\phi : \mathbb{N} \to X$, where X is the set of possible evidences. The set of the first n elements of the evidence is denoted ϕ_n, this is, $\phi_n = \{\phi(i) | i \leqslant n\}$.

Given a class $\mathcal{L}$, the set of possible presentations $\mathtt{Pres}(L)$ for each language $L \in \mathcal{L}$ must be specified to have a precisely defined learning setting. The presentation functions for the learning from text and the learning from an informant settings can be defined as follows:

$$\begin{aligned} \text{TEXT}(L) &= \{\phi : \phi(\mathbb{N}) = L\} \\ \text{INFORMANT}(L) &= \{\phi : \phi(\mathbb{N}) = L \times \{1\} \cup (\Sigma^* - L) \times \{0\}\} \end{aligned}$$

Observe that, in both cases, no two different languages can have the same presentation, and every possible evidence is eventually presented. These are usual requirements for the learning task to be feasible.

Formally, a *learning algorithm* is a function $A : \{\phi_i | i \in \mathbb{N}, \phi \in \mathtt{Pres}(\mathcal{L})\} \to \mathcal{G}$. It is said that A *identifies in the limit* the class of languages $\mathcal{L}$ if, for all $L \in \mathcal{L}$ and presentation $\phi \in \mathtt{Pres}(L)$, there is n such that $L(A(\phi_n)) = L$ and $A(\phi_n) = A(\phi_m)$ for all $m \geqslant n$. When this happens, it is also said that the class $\mathcal{L}$ is *identifiable in the limit*.

2.4.2 *Probably Approximately Correct Learning*

Probably Approximately Correct (PAC) learning emerges as a more realistic attempt to define learnability [31, 5]. In the PAC learning model, it is assumed that the evidence is sampled following an unknown probability distribution, and a gradual notion of convergence to the target is admitted. PAC learning is a general learning framework, not specific to languages. In this section we follow the notation from [31].

The PAC model adresses the problem of learning a set into a domain X, or equivalently a binary classification function for the elements of X. These sets are called concepts and they are assumed to belong to a known set of possible concepts, called the concept class $\mathcal{C}$. The learner has access to an oracle that samples elements of X according to an unknown distribution and returns them along with their classification with respect to the target concept.

A learning algorithm is a program that queries the oracle and returns a hypothesis concept. The error of the hypothesis is the probability mass of the set of elements that are classified differently by the target concept. An algorithm succeeds if it guarantees that this error is lower than a value ϵ that is given as input.

There is a problem in the fact that the oracle does sampling. The sampling may be bad, leading to a very unrepresentative sample. So, the algorithm will never be able to guarantee the error bound with probability 1. However, it will be able to do enough sampling in order

to bound the probability that the sampling is bad, to be less than a confidence value δ, also given as input to the algorithm. So, it is said that an algorithm *PAC learns* a concept class $\mathcal{C}$ if, given an error ϵ and a confidence δ, it returns a hypothesis that, with probability greater than $1 - \delta$, has an error bounded by ϵ. If this happens, it is said that the concept class $\mathcal{C}$ is *PAC learnable*.

2.5 PARSER EVALUATION

To evaluate and compare parsing systems, it is necessary to use objective methods to assess their quality. The most used approach is the corpus based one, where the evaluation is done with respect to expert annotated natural language corpora. In supervised parsing, corpora is also used to train the parsers . So, in this case, corpora are usually splitted in two parts: a *training set* and a *test set*.

In the corpus based evaluation approach, the sentences of the test set are parsed and the resulting *proposed structures* are compared with the so called *gold structures* of the corpus. To do this comparation objectively, a scoring scheme must be defined. In the following sections we review the main scores used to evaluate constituent and dependency parsing systems.

2.5.1 *Phrase Structure Metrics*

For phrase structures, the scores are defined in terms of the set of constituents of the sentences. The most common used scores are the *PARSEVAL measures* [1] and their different variants. The two fundamental PARSEVAL measures are the *precision* and the *recall*. The precision is the proportion of proposed constituent that are correct, while the recall is the proportion of correct constituents that are proposed. If we call X to the set of proposed constituents, and Y to the set of gold constituents, the measures can be generally defined as follows:

$$P = \frac{|X \cup Y|}{|X|}$$
$$R = \frac{|X \cup Y|}{|Y|}$$

The variants of this measures depend in first place on how are the elements of X and Y defined. They may or may not include the constituent labels, leading to labeled and unlabeled scores respectively. Also, unary branching constituents and the full sentence constituent may be omitted. This is particularly senseful in the case of unlabeled scores, because these constituents add redundancy.

The preterminals, this is, the POS tags of the sentence, are never considered constituents. Usually, POS tagging evaluation is done separately from parsing evaluation, and has its own particular scoring schemes.

Consider, for instance, the trees of Fig. 2.2. Let Fig. 2.2.a be the proposed tree and Fig. 2.2.b be the gold tree. In this case, for labeled scores, the sets of constituents are

$$X = \{(S,0,7),(VP,1,7),(NP,2,7),(NP,2,4),(PP,4,7),(NP,5,7)\}$$
$$Y = \{(S,0,7),(VP,1,7),(VP,1,4),(NP,2,4),(PP,4,7),(NP,5,7)\}$$

where we use the triple (V,i,j) to represent the constituent labeled V that spans the segment $s_i \dots s_{j-1}$ of the sentence $s_0 \dots s_n$. Then, the labeled precision and recall scores are

$$P = \frac{|X \cup Y|}{|X|} = \frac{5}{6}$$
$$R = \frac{|X \cup Y|}{|Y|} = \frac{5}{6}.$$

There are more variants on the measures depending on how global scores are computed for an entire test set. If the global precision and recall are computed just by averaging the precision and recall for each sentence, the measures are called *micro-averaged*. They are called *macro-averaged measures* if, in the contrary, they are computed first summing the number of correct proposed constituents for all the sentences, and then dividing by the totals, as in the following formulas:

$$P = \frac{\sum_i |X_i \cup Y_i|}{\sum_i |X_i|}$$
$$R = \frac{\sum_i |X_i \cup Y_i|}{\sum_i |Y_i|}$$

To obtain a single global score for comparison with other systems, the harmonic mean between precision and recall, called the F_1 *score*, is also usually reported:

$$F_1 = \frac{2PR}{P+R}.$$

2.5.2 *Dependency Metrics*

For dependency trees, the scores are defined in terms of the set of dependency pairs of the sentences. If the structures are well formed, the number of dependencies for a sentence is fixed. So, the scores are simpler than those for phrase structures because they rise from the comparation of two sets of equal size.

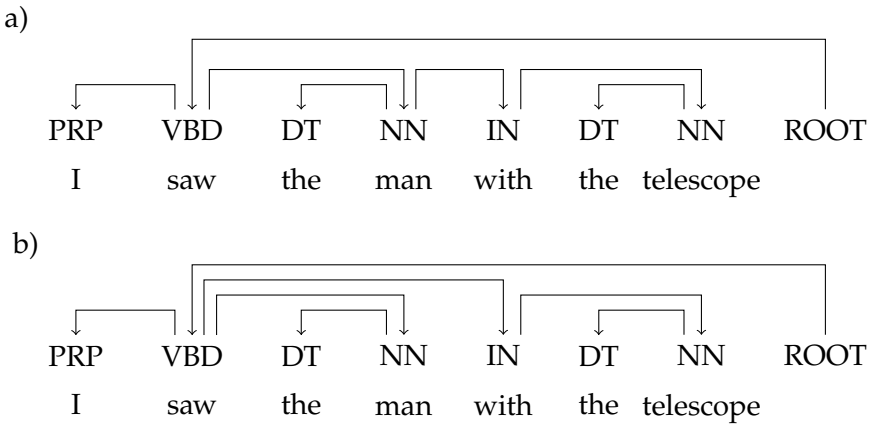


Figure 2.5: Two possible dependency trees for the sentence "I saw the man with the telescope". a) "with/IN" headed by "man/NN". b) "with/IN" headed by "saw/VBD".

The *Unlabeled Attachment Score* (UAS) is the proportion of depedencies that are correct. Formally, if X is the set of proposed dependencies, Y is the set of gold dependencies, and n is the length of the sentence, the UAS is defined by

$$\text{UAS} = \frac{|X \cup Y|}{n}$$

For instance, consider the dependency trees of Fig. 2.5. Let Fig. 2.5.a be the proposed tree and Fig. 2.5.b be the gold tree. Then, the sets of dependencies are

$$X = \{(0,1),(1,7),(2,3),(3,1),(4,3),(5,6),(6,4)\}$$
$$Y = \{(0,1),(1,7),(2,3),(3,1),(4,1),(5,6),(6,4)\}$$

where we use the pair (i,j) to represent the dependency $s_j \rightarrow s_i$ in the sentence $s_0 \dots s_n$, using $j = n+1$ when s_i is the root of the tree. This way, the UAS is

$$\text{UAS} = \frac{|X \cup Y|}{n} = \frac{6}{7}.$$

If the dependencies have some kind of labeling specifying the type of the dependency relation, there is also a labeled score, the *Labeled Attachment Score* (LAS). As with constituent scores, the dependency scores for an entire test set can be macro or micro averaged, but the established scores are the macro-averaged ones.

Part II

NON-TERMINALLY SEPARATED GRAMMARS

3

BOUNDING THE MAXIMAL PARSING PERFORMANCE OF NON-TERMINALLY SEPARATED GRAMMARS

Unambiguous Non-Terminally Separated (UNTS) grammars have good learnability properties but are too restrictive to be used for natural language parsing. We present a generalization of UNTS grammars called Unambiguous Weakly NTS (UWNTS) grammars that preserve the learnability properties. Then, we study the problem of using them to parse natural language and of evaluating them against a gold treebank. If the target language is not UWNTS, there will be an upper bound in the parsing performance.

In this chapter we present methods to obtain such upper bounds for the class of UWNTS grammars. Our methods allow us to compute the bounds without explicitly working with grammarsbecause we are only interested in the parsings of the gold sentences that the grammars return.

In principle, the function to be optimized is F_1, but instead we define a new metric W, related to F_1, but whose optimization is feasible. Our methods are based on the optimization of W and the recall, and on the translation of these optimal values to an upper bound of the F_1.

The optimization problems are solved by reducing them to the well known Integer Linear Programming (ILP) with binary variables problem, that is known to be NP-Hard, but for which there exists software to solve it [3]. Moreover, we show that the optimization of W is NP-Hard, by reducing the Maximum Independent Set problem for 2-subdivision graphs to it.

Finally, we solve the optimization problems for the WSJ10 subset of the Penn Treebank [37], compute the upper bounds, and compare them to the performance of state-of-the-art unsupervised parsers. Our results show that UWNTS grammars over the POS tags alphabet can not improve state-of-the-art unsupervised parsing performance.

3.1 NOTATION AND DEFINITIONS

We skip conventional definitions and notation for formal languages and context-free grammars, but define concepts that are more specific to this chapter.

Given a language L, $\mathrm{Sub}(L)$ is the set of non-empty substrings of the elements of L, and $\overline{\mathrm{Sub}}(L)$ is the set of non-empty proper substrings. We say that two strings r, s *overlap* if there exist non-empty strings r', s', t such that $r = r't$ and $s = ts'$; $r'ts'$ is called an *overlapping* of r and s. We say that two strings r, s *occur overlapped* in a language L if they overlap and if an overlapping of them is in $\mathrm{Sub}(L)$. Given a grammar G and $s \in L(G)$, a substring r of $s = urv$ is called a *constituent in* (u, v) if and only if there is $X \in N$ such that $S \overset{*}{\Rightarrow} uXv \overset{*}{\Rightarrow} s$. In contrast, r is called a non-constituent or *distituent in* (u, v) if it is not a constituent in (u, v).

More than in the grammars, we are in fact interested in all the possible ways a given finite set of sentences $S = \{s_1, \ldots, s_n\}$ can be parsed by a particular class of grammars. As we are modeling unlabeled parsing, the parse of a sentence is an unlabeled tree, or equivalently, a bracketing of the sentence. Formally, a *bracketing* of a sentence $s = a_0 \ldots a_{n-1}$ is a set b of pairs of indexes that marks the starting and ending positions of the constituents, consistently representing a tree. A bracketing always contains the full-span bracket $(0, n)$, never has duplicated brackets (it is a set), and does not have brackets of span 1, i.e., of the form $(i, i+1)$. Usually, we will represent the bracketings together with their corresponding sentences. For instance, we will jointly represent the sentence $abcde$ and the bracketing $\{(0,5),(2,5),(2,4)\}$ as $ab((cd)e)$. Fig. 3.1 shows an example of this. Observe that the full-span bracket is omitted.

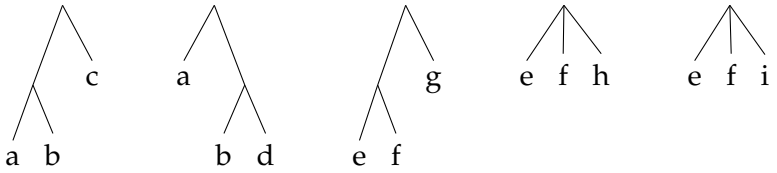
Given an unambiguous grammar G and $s \in L(G)$, the *bracketing of* s *with* G, $\mathrm{br}_G(s)$, is the bracketing corresponding to the parse tree of s with G. Given $S \subseteq L(G)$, the *bracketing of* S *with* G, $\mathrm{Br}_G(S)$, is the set $\{\mathrm{br}_G(s) | s \in S\}$.

3.1.1 *UWNTS Grammars*

A grammar G is said to be *Non-Terminally Separated* (NTS) if and only if, for all $X, Y \in N$ and $\alpha, \beta, \gamma \in (\Sigma \cup N)^*$, $X \overset{*}{\Rightarrow} \alpha\beta\gamma$ and $Y \overset{*}{\Rightarrow} \beta$ implies $X \overset{*}{\Rightarrow} \alpha Y \gamma$ [13]. A grammar is *Unambiguous NTS* (UNTS) if it is unambiguous and NTS.

Unambiguous NTS grammars are much less expressive than NTS grammars: It can be proved that a UNTS language having two overlapping sentences can not have the overlapping of them as a sentence. For instance, the set $\{ab, bc, abc\}$ can not be a subset of a UNTS language because abc is an overlapping of ab and bc. This situation is very common in the WSJ10 sentences of POS tags, and consequently

(a)



(b)

(ab)c a(bd) (ef)g efh efi

Figure 3.1: An example of gold treebank or bracketing. (a) Graphical notation. (b) Textual notation.

there is no UNTS grammar that accepts this set. For instance, it has the sentences "NN NN NN" and "NN NN", but "NN NN NN" is an overlapping of "NN NN" with itself.

The limitations of UNTS grammars lead us to define a more expressive class, UWNTS, that is able to parse any finite set of sentences preserving, at the same time, every other aspect of UNTS grammars. The properties of UWNTS will also let us characterize the sets of bracketings of all the possible grammars that parse a given finite set of sentences. This characterization will be at the core of our methods for finding bounds.

Definition 1. *A grammar* $G = \langle \Sigma, N, S, P \rangle$ *is* Weakly Non-Terminally Separated (WNTS) *if* S *does not appear on the right side of any production and, for all* $X, Y \in N$ *and* $\alpha, \beta, \gamma \in (\Sigma \cup N)^*$ *such that* $Y \neq S$,

$$X \stackrel{*}{\Rightarrow} \alpha\beta\gamma \text{ and } Y \stackrel{*}{\Rightarrow} \beta \text{ implies } X \stackrel{*}{\Rightarrow} \alpha Y \gamma.$$

A grammar is Unambiguous WNTS (UWNTS) *if it is unambiguous and WNTS.*

Note that any finite set $\{s_1, \ldots, s_n\}$ is parsed by a UWNTS grammar with rules $\{S \to s_1, \ldots, S \to s_n\}$. It is easy to see that every NTS language is also WNTS. A much more interesting result is that a WNTS language can be converted into an NTS language without loosing any information.

Lemma 1. *If* L *is WNTS, then* $xLx = \{xsx | s \in L\}$ *is NTS, where* x *is a new element of the alphabet.*

Proof. See Appendix A. □

This conversion allows us to prove PAC-learnability of UWNTS languages using the PAC-learnability result for UNTS languages of [13]. If we want to learn a UWNTS grammar from a sample S, we can simply give the sample xSx to the learning algorithm for UNTS

grammars, and remove the x's from the resulting grammar. A detailed proof of this result is beyond the scope of this chapter. In the next chapter we will present a more general learnability result, of a class that contains the UWNTS languages.

In a UWNTS grammar G, every substring is either always a constituent in every proper occurrence or always a distituent in every proper occurrence. If two strings r, s occur overlapped in $L(G)$, then at least one of them must be always a distituent. In this case, we say that r and s are *incompatible* in $L(G)$ and we say that they are *compatible* in the opposite case. We say that a set of strings is compatible in $L(G)$ if every pair of strings of the set is compatible in $L(G)$.

Now let us consider a finite set of sentences S, a UWNTS grammar G, and the bracketing of S with G. The given properties imply that in the bracketing there are no substrings marked some times as constituents and some times as distituents. Consequently, the information in $Br_G(S)$ can be simply represented as the set of substrings in $\overline{Sub}(S)$ that are always constituents. We call this the set $Const_G(S)$. When considering all the possible UWNTS grammars G with $S \subseteq L(G)$, there is a 1-to-1 mapping between all the possible bracketings $Br_G(S)$ and all the possible sets $Const_G(S)$. Using this information, we can define our search space $\mathscr{C}(S)$ as

Definition 2. $\mathscr{C}(S) \doteq \{Const_G(S) : G \textit{ UWNTS and } S \subseteq L(G)\}$.

With UWNTS grammars we can characterize the search space in a way that there is no need to explicitly refer to the grammars. We can see that the search space is equal to all the possible subsets of $\overline{Sub}(S)$ that are compatible in S:

Theorem 1. $\mathscr{C}(S) = \{C : C \subseteq \overline{Sub}(S), C \textit{ compatible in } S\}$.

Proof. See Appendix A for a sketch. □

The proof of $\subseteq$ follows immediately from the given properties. $\supseteq$ is proved by constructing a UWNTS grammar mainly using the fact that S is finite and C is compatible in S.

3.1.2 *UWNTS-SC Grammars*

In the parser induction problem, it is usually expected that the induced parsers will be able to parse any sentence, and not only the finite set of sentences S that is used to evaluate them. However, the defined search space for UWNTS grammars does not guarantee this. In Theorem 1, it is evident that the sets of constituents are required to be compatible in S, but may be incompatible with other sets of sentences. For instance, if $S = \{abd, bcd\}$ and $C = \{ab, bc\}$, C is compatible in S but not in $\{abc\}$. We define UWNTS-SC grammars as the subclass of UWNTS that guarantee that the constituents are compatible with any set of sentences.

Definition 3. *A grammar* G *is* UWNTS Strongly Compatible (UWNTS-SC) *if it is UWNTS and, for every* $r, s \in \mathrm{Const}_G(L(G))$, r *and* s *do not overlap. When* r *and* s *do not overlap, we say that* r *and* s *are* strongly compatible. *And when every pair of a set of strings* C *do not overlap, we say that* C *is* strongly compatible.

As UWNTS-SC is a subclass of UWNTS, the properties of UWNTS grammars still hold. The search space for UWNTS-SC grammars and its characterization are:

Definition 4. $\mathscr{C}_{SC}(S) \doteq \{\mathrm{Const}_G(S) : G \textit{ UWNTS-SC and } S \subseteq L(G)\}$.

Theorem 2. $\mathscr{C}_{SC}(S) = \{C : C \subseteq \overline{\mathrm{Sub}}(S), C \textit{ strongly compatible}\}$.

The proof of this theorem is analog to the proof of Theorem 1.

Theorem 2 shows a more natural way of understanding the motivation under UWNTS-SC grammars. While the compatibility property depends on the sample S, the strong compatibility does not. It can be checked for a given set of constituents without looking at the sample, and if it holds, we know that the set is compatible for any sample.

3.2 THE W MEASURE AND ITS RELATIONSHIP TO THE F_1

In this section we will study the evaluation measures in the frame of UWNTS grammars. In general, a measure is a function of similarity between a given gold bracketing $B = \{b_1, \ldots, b_n\}$ of a set of gold sentences and a proposed bracketing $\hat{B} = \{\hat{b}_1, \ldots, \hat{b}_n\}$ over the same set of sentences. The standard measures used in unsupervised constituent parsing are the micro-averaged precision, recall and F_1 as defined in [32]:

$$P(\hat{B}) \doteq \frac{\sum_{k=1}^{n} |\hat{b}_k \cap b_k|}{\sum_{k=1}^{n} |\hat{b}_k|}$$

$$R(\hat{B}) \doteq \frac{\sum_{k=1}^{n} |\hat{b}_k \cap b_k|}{\sum_{k=1}^{n} |b_k|}$$

$$F_1(\hat{B}) \doteq \frac{2P(\hat{B})R(\hat{B})}{P(\hat{B}) + R(\hat{B})}.$$

Note that these measures differ from the standard PARSEVAL measures [1], because from the definition of bracketing it follows that the syntactic categories are ignored and that unary branches are ignored.

Another way to define precision and recall is in term of two measures that we will call hits and misses.

Definition 5. *The* hits *is the number of proposed brackets that are correct, and the* misses *is the number of proposed brackets that are not correct. Formally,*

$$H(\hat{B}) \doteq \sum_{k=1}^{n} |\hat{b}_k \cap b_k|$$
$$M(\hat{B}) \doteq \sum_{k=1}^{n} |\hat{b}_k - b_k|.$$

Using these two measures, and defining

$$K \doteq \sum_{k=1}^{n} |b_k|$$

to be the number of brackets in the gold bracketing, we have

$$P(\hat{B}) = \frac{H(\hat{B})}{H(\hat{B}) + M(\hat{B})}$$
$$R(\hat{B}) = \frac{H(\hat{B})}{K}.$$

As we saw in the previous section, in the case of UWNTS grammars the bracketings can be represented as sets of constituents $\hat{C}$. So, we will rewrite the definitions of the measures in terms of $\hat{C}$ instead of $\hat{B}$. Observe that, if $s \in \hat{C}$, $\hat{B}$ contains all the occurrences of s marked as a constituent. But in the gold B, s may be marked some times as a constituent and some times as a distituent. Let $c(s)$ and $d(s)$ be number of times s appears in B as a constituent and as a distituent respectively:

Definition 6.

$$c(s) \doteq |\{(u, s, v) : uv \neq \lambda, usv \in S \textit{ and } s \textit{ is a constituent in } (u, v)\}|$$
$$d(s) \doteq |\{(u, s, v) : uv \neq \lambda, usv \in S \textit{ and } s \textit{ is a distituent in } (u, v)\}|$$

Then, for every $s \in \hat{C}$, $\hat{B}$ will have $c(s)$ hits and $d(s)$ misses. This is,

$$H(\hat{C}) = \sum_{s \in \hat{C}} c(s)$$
$$M(\hat{C}) = \sum_{s \in \hat{C}} d(s).$$

Using this, we can see that

$$F_1(\hat{C}) = \frac{2 \sum_{s \in \hat{C}} c(s)}{K + \sum_{s \in \hat{C}} c(s) + d(s)}.$$

Now that we have written the F_1 in terms of $\hat{C}$, we would like to define an algorithm to find the optimal $F_1(\hat{C})$ for every $\hat{C} \in \mathscr{C}(S)$. As

Bracketing	P	R	F_1	H	M	W
$\{(ab)c, (ab)d, efg, efh, efi\}$	50%	33%	40%	1	1	0
$\{(ab)c, (ab)d, (ef)g, (ef)h, (ef)i\}$	40%	67%	50%	2	3	-1

Table 3.1: Comparison of the F_1 and W measures: The scores of two bracketings with respect to the gold bracketing of Fig. 3.1.

the search space is finite, a simple algorithm to find $\max_{\hat{C}} F_1(\hat{C})$ is to compute the F_1 for every $\hat{C}$. The problem is that the order of this algorithm is $O(2^{|Sub(S)|})$.

Given that the optimization of the F_1 does not seem to be feasible, we will define another measure W which optimization will result more tractable. This measure has its own intuition and is a natural way to combine hits and misses. It is very different to the F_1, but we will see that an upper bound of W can be translated to an upper bound of the F_1 measure. We will also see that an upper bound of the recall R can be used to find a better bound for the F_1.

We want W to be such that a high value for it reflects a big number of hits and a low number of misses, so we simply define:

Definition 7. $W(\hat{B}) \doteq H(\hat{B}) - M(\hat{B})$.

Note that, when dealing with UWNTS grammars, as H and M are linear expressions over $\hat{C}$, W will also be linear over $\hat{C}$, unlike the F_1 measure. This is what will make it more tractable.

W and F_1 are actually very different measures. In first place note that W is not in the range $[0, 1]$ as the F_1. The range of W will depend on the concrete gold bracketing B. It can be seen that W will have a value of at most K, and of at least $-K'_S$, where $K'_S = \sum_{s \in S} |s| - 2$ is the maximal number of brackets that a bracketing of S can have.

Moreover, W and F_1 do not define the same ordering over candidate bracketings $\hat{B}$. For instance, Table 3.1 shows the scores for two different bracketings $\hat{B}_1, \hat{B}_2$ with respect to the same gold bracketing. Here, F_1 is higher for $\hat{B}_2$ but W is higher for $\hat{B}_1$.

However, the W and F_1 measures are related in some way, as shown in the following formula:

$$F_1(\hat{B}) = \frac{2R(\hat{B})}{1 + 2R(\hat{B}) - \frac{W(\hat{B})}{K}}.$$

Here, we can forget about the bracketings and write the value of the F_1 directly in terms of the recall value $r = R(\hat{B})$ and the W value $w = W(\hat{B})$:

$$F_1(r, w) = \frac{2r}{1 + 2r - \frac{w}{K}}. \tag{3.1}$$

From this formula it can be seen that the F_1 is monotonically increasing in both w and r. In the case of r, this is proved by observing that

$\frac{\partial F_1}{\partial r} \geqslant 0$ iif $w \leqslant K$, which is in fact true. In the case of w, the proof is trivial.

Then, if r and w are upper bounds for their respective measures, $F_1(r, w)$ is an upper bound for the F_1 measure. If we do not know an upper bound for the recall, we can simply use the bound $r = 1$.

3.3 THE OPTIMIZATION OF W AND R

In this section we first formalize the optimization problems of W and R over the UWNTS and UWNTS-SC grammars. We define them in terms of a more general problem of optimization of a score over a class of grammars. Then, we show how to reduce the problems to Integer Linear Programming problems, using the fact that W and R are computed as linear expressions in terms of $\hat{C}$. Finally, we show that the problems are NP-Hard.

Definition 8. *Given a class of grammars $\mathcal{G}$, a score function s, a set of gold sentences S and a set of gold bracketings B, the* Maximum Score Grammar (MSG) *problem is such that it computes*

$$\text{MSG}(\mathcal{G}, s, S, B) = \max_{G \in \mathcal{G}, S \subseteq L(G)} s(B, \text{Br}_G(S)).$$

Definition 9. *The* Maximum W UWNTS (MW-UNTS), Maximum W UWNTS-SC (MW-UNTS-SC), Maximum R UWNTS (MR-UNTS) *and* Maximum R UWNTS-SC (MR-UNTS-SC) *problems are such that they compute*

$$\begin{aligned}
\textit{MW-UWNTS}(S, B) &= \text{MSG}(\textit{UWNTS}, W, S, B) \\
\textit{MW-UWNTS-SC}(S, B) &= \text{MSG}(\textit{UWNTS-SC}, W, S, B) \\
\textit{MR-UWNTS}(S, B) &= \text{MSG}(\textit{UWNTS}, R, S, B) \\
\textit{MR-UWNTS-SC}(S, B) &= \text{MSG}(\textit{UWNTS-SC}, R, S, B).
\end{aligned}$$

3.3.1 *Solving the Problems for UWNTS Grammars*

Let us first consider the problem of W maximization, MW-UWNTS. By the characterization of the search space of Theorem 1, we have that

$$\text{MW-UWNTS}(S, B) = \max_{C \subseteq \overline{\text{Sub}}(S), C \text{ compatible in } S} \sum_{s \in C} c(s) - d(s).$$

Now, let $H(S, B) = \langle V, E, w \rangle$ be an undirected weighted graph such that

- $V = \overline{\text{Sub}}(S)$,
- $w(s) = c(s) - d(s)$ and
- $E = \{ (s, t) : s, t \text{ incompatible in } S \}$.

An independent set of a graph is a set of nodes such that every pair of nodes in it is not connected by an edge. So, C is an independent set of $H(S, B)$ if and only if C is compatible in S. Now, consider the Maximum Weight Independent Set (MWIS) problem, that given G returns the weight of the independent set with maximum weight of G [30]. Clearly,

$$\text{MW-UWNTS}(S, B) = \text{MWIS}(H(S, B)).$$

This is a reduction of our problem to the MWIS problem, that is known to be NP-Hard [30]. In turn, MWIS is reducible to the Integer Linear Programming (ILP) problem with binary variables. This problem is also NP-Hard, but there exists software that implements efficient strategies for solving some of its instances [3].

An ILP problem with binary variables is defined by three parameters $\langle \mathbf{x}, f, \mathbf{c} \rangle$: A set of variables $\mathbf{x}$ that can take values in $\{0, 1\}$, an objective linear function $f(\mathbf{x})$ that has to be maximized, and a set of linear constraints $\mathbf{c}(\mathbf{x})$ that must be satisfied. The result of the ILP problem is a valuation of the variables that satisfies the constraints and maximizes the objective function.

In the case of MWIS, given a weighted graph $G = \langle V, E, w \rangle$, we define a binary variable x_v for every node v. A valuation in $\{0, 1\}$ of the variables will define a subset of nodes $\{v | x_v = 1\}$. To ensure that the possible subsets are independent sets, we define the constraints $\mathbf{c}(\mathbf{x}) = \{x_v + x_w \leqslant 1 | (v, w) \in E\}$. Finally, the objective function is $f(\mathbf{x}) = \sum_{v \in V} x_v w(v)$. Using this instance $I(G) = \langle \mathbf{x}, f, \mathbf{c} \rangle$, we have that in general $\text{MWIS}(G) = \text{ILP}(I(G))$, and in particular that

$$\text{MW-UWNTS}(S, B) = \text{ILP}(I(H(S, B))).$$

We illustrate the reduction of MW-UWNTS to MWIS and then to ILP with the example instance of Fig. 3.2 (a). The sets of substrings such that $w(c) \geqslant 0$ is $\{da, cd, bc, cda, ab, bch\}$. The input graph for the MWIS problem and the instance of the ILP problem are given in Fig. 3.3 (a) and (b).

Now we consider the solution of the problem of recall maximization, MR-UWNTS. We can do a similar reduction to MWIS changing the graph weights from $c(s) - d(s)$ to $c(s)$, and then reduce MWIS to ILP. If $H'(S, B)$ is the MWIS instance, then

$$\text{MR-UWNTS}(S, B) = \frac{1}{K} \text{ILP}(I(H'(S, B))).$$

3.3.2 *Solving the Problems for UWNTS-SC Grammars*

The problems for UWNTS-SC grammars can also be reduced to MWIS as in the previous section just changing in the graph construction the definition of the edge set. Instead of saying that $(s, t) \in E$

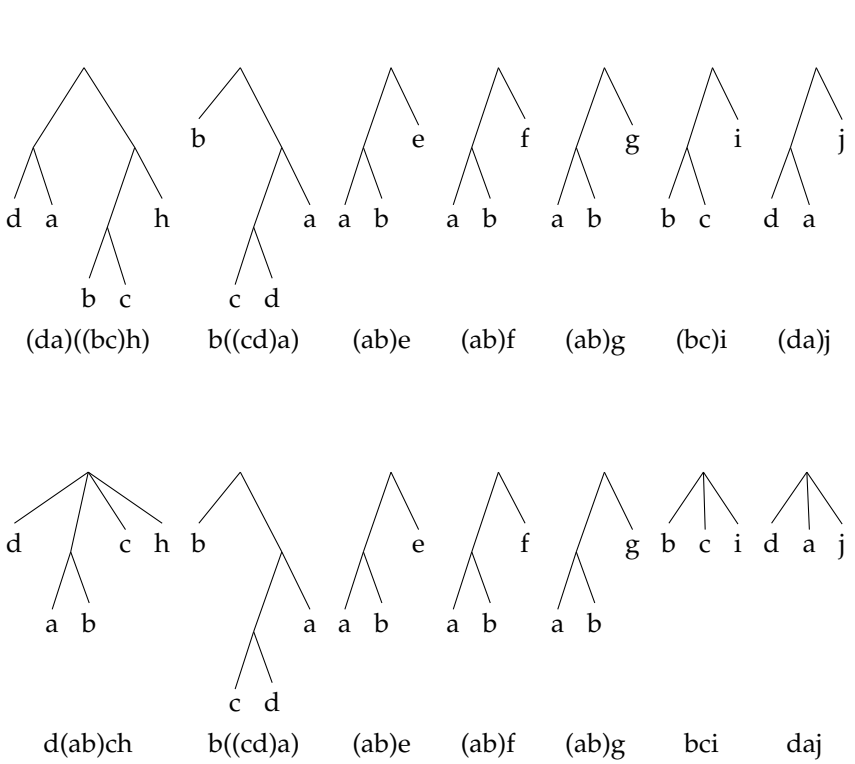


Figure 3.2: (a) A gold bracketing. (b) The bracketing generated by the UWNTS grammars with $C = \{cd, ab, cda\}$.

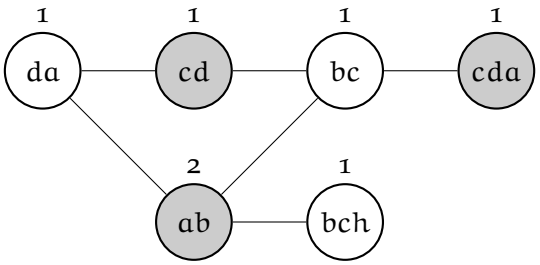
iff s and t are incompatible in S, we say that $(s,t) \in E$ iff s and t are strongly incompatible, this is, s and t overlap. But this reduction leads to a much bigger number of edges. In our experiments with the WSJ10, this resulted in a number of ILP constraints that was unmanageable by our working version of the ILP software we used.

For instance, if there are n substrings that end with the symbol a and m substrings that start with a, there will be nm edges/constraints. These constraints express the fact that "every string starting with an a is strongly incompatible with every string ending with an a", or equivalently "every string starting with an a has value 0 or every string ending with an a has value 0". But this can be expressed with less constraints using a new ILP variable y_a that has value 0 if every substring starting with a has value 0, and value 1 if every substring ending with a has value 0. To do this we use, for each substring of the form as, the constraint $x_{as} \leqslant y_a$ and for each substring of the form ta, the constraint $x_{ta} \leqslant 1 - y_a$ leading to $n + m$ constraints instead of nm.

In the general form, the ILP instance is $I = \langle \mathbf{x}, f, \mathbf{c} \rangle$, where

$$\begin{aligned}\mathbf{c}(\mathbf{x}) = &\{x_s \leqslant y_t | s \in V, t \text{ proper prefix of } s\} \\ &\cup \{x_s \leqslant 1 - y_u | s \in V, u \text{ proper suffix of } s\}.\end{aligned}$$

(a)



(b)

$$
\begin{aligned}
\mathbf{x} &= \{x_{da}, x_{cd}, x_{bc}, x_{cda}, x_{ab}, x_{bch}\} \\
f(\mathbf{x}) &= x_{da} + x_{cd} + x_{bc} + x_{cda} + 2x_{ab} + x_{bch} \\
\mathbf{c}(\mathbf{x}) &= \{x_{da} + x_{cd} \leqslant 1, x_{cd} + x_{bc} \leqslant 1, \\
&\quad x_{bc} + x_{cda} \leqslant 1, x_{da} + x_{ab} \leqslant 1, \\
&\quad x_{ab} + x_{bc} \leqslant 1, x_{ab} + x_{bch} \leqslant 1\}
\end{aligned}
$$

Figure 3.3: (a) Graph for the MWIS problem for the gold bracketing of Fig. 3.2. The shadowed nodes conform the solution. (b) Instance for the ILP problem.

3.3.3 *NP-Hardness of the Problems*

It can be shown that the presented problems, MW-UWNTS, MR-UWNTS, MW-UWNTS-SC and MR-UWNTS-SC, are all NP-Hard. In this section, we prove only that the MW-UWNTS problem is NP-Hard. The proofs for the rest of the problems are analogous.

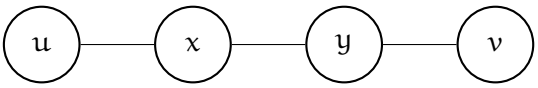
Theorem 3. *MW-UWNTS is NP-Hard.*

Proof. We will prove this by reducing the NP-Hard Maximum Independent Set (MIS) graph problem to the MW-UWNTS problem. MIS is the optimization problem over unweighted graphs that is equivalent to the MWIS problem over weighted graphs where all the weights are 1. We have to provide a way to convert instances of MIS to instances of MW-UWNTS. To simplify this process, we will restrict the domain of the MIS problem to the class of 2-subdivision graphs, where it is still known to be NP-Hard [47]. This is the key idea of the proof. The 2-subdivision graphs are those graphs that are obtained from another graph by replacing every edge $\{u, v\}$ by a path $uxyv$, where x and y are new nodes. The path $uxyv$ is called the *2-subdivision path* of the edge $\{u, v\}$ and x and y are called the *2-subdivision nodes* of $\{u, v\}$.

Let H be a 2-subdivision graph for which we want to solve the MIS problem, and let G be the graph from which H is obtained by 2-subdividing it. We will construct an instance of the MW-UWNTS problem (S, B) in terms of H, this is, a set of gold sentences and its

(a)

H:



2-subdivision of G:

(b) $\Sigma = \{ a_u, b_u, a_v, b_v, c_{uv}, d, e\}$

$Substrings = \{ a_u b_u, a_v b_v, b_u c_{uv}, c_{uv} a_v\}$

$(S, B) = \{(a_u b_u)c_{uv}, (b_u c_{uv})a_v, (c_{uv} a_v)b_v,$
$(b_u c_{uv})d, (c_{uv} a_v)d, (a_v b_v)d, (a_v b_v)e\}$

(c)

H(S, B):

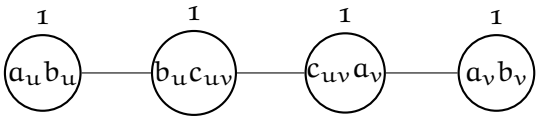


Figure 3.4: Example of conversion of a 2-subdivision graph H (a), instance of the MIS problem, to a pair (S, B) (b), instance of the MW-UWNTS problem. In (c) it is shown that H is equivalent to $H(S, B)$.

corresponding gold bracketing. The instance (S, B) will be such that, when reduced to the graph $G^w_{S,B}$ as explained in section 3.3.1, this graph will have all weights equal to 1 and will be structurally equal to the original graph H. As a solution of the MW-UWNTS problem gives a solution of the MWIS problem for $G^w_{S,B}$, this solution is also a solution of the MIS problem for H. This way, we can successfully solve the MIS problem in terms of a MW-UWNTS problem.

To describe the instance (S, B), we will first define the alphabet it uses, then the set of substrings that occur in S, and finally the gold sentences S and the brackets that conform the gold bracketing B. The alphabet will have, for each node v in H that is also in G, two symbols a_v and b_v, and for each 2-subdivision path $uxyv$ in H, a symbol c_{uv}. The set of substrings will have one substring for each node in H. The substrings must overlap in a way that the overlappings encode the edges of H. For every node v in G we will use the substring $a_v b_v$ in the treebank, and for every 2-subdivision nodes x and y of $\{u, v\}$ we will use the substrings $b_u c_{uv}$ and $c_{uv} a_v$ respectively. Note that, for every 2-subdivision path $uxyv$ of H, the string corresponding to the node u overlaps with the string of the node x, the string of x overlaps with the string of y, and the string of y overlaps with the string of v. Also, it can be seen that there is no overlapping that does not come from an edge of a 2-subdivision path of H.

The sentences must contain the mentioned substrings in a way that all the pairs of substrings that overlap, effectively appear overlapped. This way, the edges of H will be rebuilt from the treebank. To do this, for each 2-subdivision path $uxyv$ we define the sentences $\{a_u b_u c_{uv}, b_u c_{uv} a_v, c_{uv} a_v b_v\}$.

We must define also the brackets for these sentences. These have to be such that every substring s that correspond to a node in H has weight $w(s) = c(s) - d(s) = 1$. This will not be possible unless we use some extra sentences. For instance, if we use the bracketings $\{(a_u b_u) c_{uv}, (b_u c_{uv}) a_v, (c_{uv} a_v) b_v\}$ we will have $w(a_u b_u) = 1$, $w(b_u c_{uv}) = w(b_u c_{uv}) = 0$, and $w(a_v b_v) = -1$. All the weights can be set to 1 using new symbols and sentences like $\{(b_u c_{uv}) d, (c_{uv} a_v) d, (a_v b_v) d, (a_v b_v) e\}$. The new substrings, such as $c_{uv} d$, will all have weight < 0 so they will not appear in the graph. In general, it will always be possible to fix the weights of the substrings by adding new sentences.

The described conversion of H to (S, B) is $O(|V(H)|)$ in time and space or, equivalently, $O(|V(G)| + |E(G)|)$.

A very simple example of this process is shown in Fig. 3.4. □

3.4 UPPER BOUNDS FOR THE WSJ10 TREEBANK

This section shows the results of computing concrete upper bounds for the classes of UWNTS and UWNTS-SC grammars over the WSJ10 treebank. The WSJ10 consists of the sentences of the WSJ Penn Treebank whose length is of at most 10 words after removing punctuation marks [32].

There is a total of 63742 different non-empty substrings of POS tags in $\overline{\mathrm{Sub}}(S)$. To solve the optimization problems, the strings with $w(s) \leqslant 0$ can be ignored because they do not improve the scores. In the case of the MW problems, where $w(s) = c(s) - d(s)$, the number of strings with $w(s) > 0$ is 7029. In the case of the MR problems, where $w(s) = c(s)$, the number of strings with $w(s) > 0$ is 9112. The sizes of the resulting instances are summarized in Table 3.2. Observe that the number of edges of the MWIS instances for UWNTS-SC grammars have a higher order of magnitude than the size of the improved ILP instances.

Using the obtained results of Maximal W and Maximal R we proceeded to compute the upper bounds for the F_1 using (3.1) from Sect. 3.2. Table 3.3 shows the results, together with the performance of actual state-of-the-art unsupervised parsers. We show the computed maximal W and maximal recall for UWNTS and UWNTS-SC grammars, and also the values of all the other measures associated to these solutions. We also show the upper bounds for the F_1 in the rows labeled UBoundF_1(UWNTS) and UBoundF_1(UWNTS-SC), together with their corresponding precision and recall. RBranch is

	MWIS		ILP	
	Nodes	Edges	Variables	Constraints
MW-UWNTS	7029	1204	7029	1204
MR-UWNTS	9112	45984	9112	45984
MW-UWNTS-SC	7029	1467257	26434	67916
MR-UWNTS-SC	9112	3166833	28815	79986

Table 3.2: Sizes of the MWIS and ILP instances generated by the MW and MR problems for the WSJ10 treebank.

Model	P	R	F_1	H	M	W
RBranch	55.1	70.0	61.7			
DMV+CCM	69.3	88.0	77.6			
U-DOP	70.8	88.2	78.5			
Incremental	75.6	76.2	75.9			
MW-UWNTS	91.2	62.8	74.4	22169	2127	**20042**
MR-UWNTS	73.8	**69.0**	71.3	24345	8643	15702
UBoundF_1(UWNTS)	85.0	69.0	**76.1**			
MW-UWNTS-SC	89.1	52.2	65.8	18410	2263	**16147**
MR-UWNTS-SC	65.3	**61.1**	63.1	21562	11483	10079
UBoundF_1(UWNTS-SC)	79.9	61.1	**69.2**			

Table 3.3: Summary of the results of our experiments with the WSJ10, in contrast with state-of-the-art unsupervised parsers. The numbers in bold mark the upper bounds or maximal values obtained in each row.

a baseline parser that parses every sentence with a right-branching bracketing. DMV+CCM is the parser from Klein and Manning [32], U-DOP is the parser from Bod [7] and Incremental is the parser from Seginer [49].

3.5 DISCUSSION

Our bounding methods are specific to the evaluation approach of comparing against gold treebanks, but this is the most accepted and supported approach, and it is scalable and unbiased with respect to the evaluator [56]. Our methods are also specific to the unlabeled F_1 measure as we defined it, with micro-averaging and removal of unary brackets. Also in [56], micro-averaging and removal of trivial structure are proposed as the preferred evaluation method.

The quality of our upper bounds depend on how far they are from the optimal F_1. We can measure this by looking at the F_1 associated

to the solution for the MW problem, because it represents an actual grammar. For instance, the F_1 for MW-UWNTS over the WSJ10 is 74.4%, so the upper bound of 76.1% is near to the optimal bound.

4

PAC-LEARNING UNAMBIGUOUS k, l-NTS$^{\leqslant}$ GRAMMARS

Among the most recent results in the Grammatical Inference field, there are the results of polynomial PAC-learnability of Unambiguous NTS (UNTS) languages from Clark [13], and the polynomial identifiability in the limit of Substitutable Context-Free Languages (SCFLs) from Clark and Eyraud [15]. These two results are related in the sense that substitutable languages are a sort of language level definition of the NTS property, and Clark and Eyraud actually conjectured that the SCFLs are a subclass of the NTS languages. Another recent result is Yoshinaka's [57] generalization of Clark and Eyraud work, defining the hierarchy of k, l-substitutable languages, and proving their polynomial identifiability in the limit.

In this chapter, we define the hierarchy of k, l-NTS grammars that generalizes NTS grammars. k, l-NTS grammars add the notion of a fixed size context to the constituency property of the NTS grammars. We then present the k, l-NTS$^{\leqslant}$ grammars, a hierarchy of subclasses of the k, l-NTS grammars. These grammars also consider the boundaries of sentences as valid contexts.

We prove that k, l-UNTS$^{\leqslant}$ grammars can be converted injectively to UNTS grammars over a richer alphabet. We do this by showing how to convert k, l-UNTS$^{\leqslant}$ grammars with $l > 0$ to $k, l-1$-UNTS$^{\leqslant}$ grammars. Applying this conversion recursively, we can get to $k, 0$-UNTS$^{\leqslant}$ grammars, and applying a symmetric conversion, we finally get to $0, 0$-UNTS$^{\leqslant}$ grammars, that are exactly UNTS grammars.

We then use the conversion to UNTS grammars to prove that k, l-UNTS$^{\leqslant}$ grammars are PAC-learnable. A sample of a k, l-UNTS$^{\leqslant}$ language can be converted to a sample of a UNTS language, and this sample can be used with Clark's algorithm to learn a grammar that has a language that converges to the UNTS language. The learned grammar can then be converted back, to obtain a k, l-UNTS$^{\leqslant}$ grammar with a language that converges to the original k, l-UNTS$^{\leqslant}$ language.

4.1 NOTATION AND DEFINITIONS

Recalling from section 2.1.2, a Context-Free Grammar (CFG) is a tuple $G = \langle \Sigma, N, S, P \rangle$, where Σ is the terminal alphabet, N is the set of non-terminals, $S \in N$ is the initial non-terminal, and $P \subseteq N \times (\Sigma \cup N)^+$ is the set of productions or rules.

We use letters from the beginning of the alphabet $a, b, c, \ldots$ to represent elements of Σ, from the end of the alphabet $r, s, t, u, v, \ldots$ to represent elements of Σ^*, and Greek letters to represent elements of $(\Sigma \cup N)^*$. We use the notation $(\alpha)_i$ to refer to the i-th element of α. We write the rules using the form $X \to \alpha$, and call the derivation relation $\overset{*}{\Rightarrow}$ to the transitive closure of the relation $\Rightarrow$ where $\alpha X \beta \Rightarrow \alpha\gamma\beta$ iff $X \to \gamma \in P$.

The language generated by a CFG G is the Context-Free Language (CFL) $L(G) = \{s \in \Sigma^* | S \overset{*}{\Rightarrow} s\}$. As in [13], we will assume that all the non-terminals are useful, that is, that they are used in the derivation of some element of the language, and that they are not duplicated, that is, that they all generate different sets of strings.

A grammar is said to be Non-Terminally Separated (NTS) if and only if whenever $X \overset{*}{\Rightarrow} \alpha\beta\gamma$ and $Y \overset{*}{\Rightarrow} \beta$, we have that $X \overset{*}{\Rightarrow} \alpha Y \gamma$. This definition implies that if a string occurs as a constituent, then all of its occurrences are also constituents.

4.1.1 k, l-*NTS Grammars*

We present a generalization of NTS grammars that relaxes the NTS condition to introduce the influence of fixed size contexts. In the generalization, the constituency of an occurrence of a string will not be determined only by the string itself but also by the context where it occurs.

Definition 10. *Let k, l be two non-negative integers. A grammar $G = \langle \Sigma, N, S, P \rangle$ is* k, l-Non-Terminally Separated (k, l-NTS) *if, for all $X, Y \in N$, α, β, γ, α', $\gamma' \in (\Sigma \cup N)^*$, and $(u, v) \in \Sigma^k \times \Sigma^l$,*

$$X \overset{*}{\Rightarrow} \alpha u \beta v \gamma, Y \overset{*}{\Rightarrow} \beta \text{ and } S \overset{*}{\Rightarrow} \alpha' u Y v \gamma' \text{ implies } X \overset{*}{\Rightarrow} \alpha u Y v \gamma.$$

A grammar is k, l-Unambiguous NTS (k, l-UNTS) *if it is unambiguous and k, l-NTS.*

This definition states that if a string occurs as a constituent in a context (u, v), then every time it occurs in that context, it has to be a constituent. In contrast with NTS grammars that have one global set of constituents, in k, l-NTS there is a set of constituents for each possible context (u, v) such that $(|u|, |v|) = (k, l)$.

From the definition, it is easy to see that the $0, 0$-NTS grammars are exactly the NTS grammars, and that if a grammar is k, l-NTS, then it is m, n-NTS for $m \geqslant k$, $n \geqslant l$. So, k, l-NTS grammars conform a

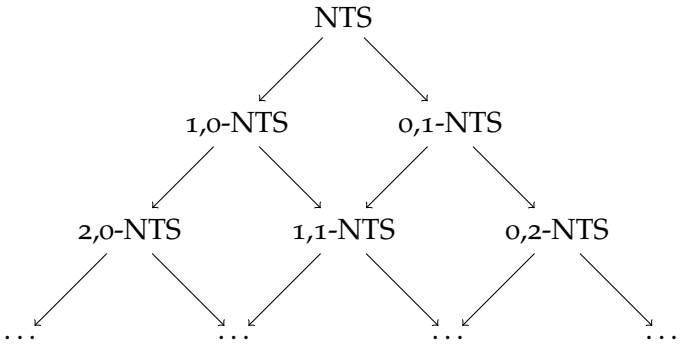


Figure 4.1: Hierarchy of k, l-NTS classes of grammars. The arrows represent proper inclusion.

hierarchy of classes of grammars that starts with the NTS grammars. This hierarchy is strict. For instance, the grammar with productions $P = \{S \to a^k Y c^{l-1}, S \to a^{k+1} b c^l, Y \to b\}$ with $l > 0$, is k, l-NTS but it is not $k, l-1$-NTS. Fig. 4.1 illustrates the hierarchy, showing the strict inclusion relationships that hold between the particular classes.

4.1.2 k, l-*NTS*$^{\leqslant}$ *Grammars*

We will focus our attention on a particular family of subclasses of k, l-NTS grammars. Observe that in the definition of k, l-NTS grammars, the Y non-terminals are those that can be derived from S with contexts of size of at least (k, l). This means that the k, l-NTS condition does not affect those that have smaller contexts.

In k, l-NTS$^{\leqslant}$ grammars the constituency of the strings will be determined not only by contexts of size (k, l) but also by smaller contexts if they occur next to the boundaries of the sentences. We will define k, l-NTS$^{\leqslant}$ in terms of k, l-NTS and use the following simple grammar transformation.

Definition 11. *Let* $G = \langle \Sigma, N, S, P \rangle$ *be a CFG, and let* $u, v \in \Sigma'^*$. *Then,* $uGv = \langle \Sigma \cup \Sigma', N \cup \{S'\}, S', P' \rangle$ *is a CFG such that* $P' = P \cup \{S' \to uSv\}$.

It is easy to see that $L(uGv) = \{usv | s \in L(G)\}$.

Definition 12. *Let* G *be a CFG and let* $\bullet$ *be a new element of the alphabet. Then,* G *is* k, l-Non-Terminally Separated$^{\leqslant}$ (k, l-NTS$^{\leqslant}$) *if and only if* $\bullet^k G \bullet^l$ *is* k, l-*NTS. A grammar is* k, l-Unambiguous NTS$^{\leqslant}$ (k, l-UNTS$^{\leqslant}$) *if it is unambiguous and* k, l-*NTS*$^{\leqslant}$.

The definition says that if we add a prefix $\bullet^k$ and a suffix $\bullet^l$ to every element of the language, then the resulting language is k, l-NTS. Doing this, we guarantee that every substring in the original language has a context of size (k, l) and therefore is affected by the k, l-NTS condition.

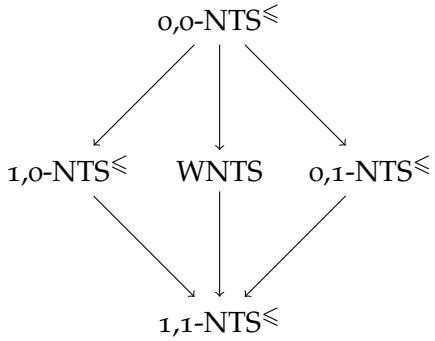


Figure 4.2: Relationship between the WNTS and the k,l-NTS$^{\leqslant}$ classes of grammars. The arrows represent proper inclusion.

It is easy to see that $0,0$-NTS$^{\leqslant} = 0,0$-NTS = NTS, and that k,l-NTS$^{\leqslant}$ is a hierarchy, where k,l-NTS$^{\leqslant} \subset m,n$-NTS$^{\leqslant}$ for every $m > k$, $n > l$. It can be seen that this hierarchy is strict using the same example given for k,l-NTS grammars. Also, it can be proved that k,l-NTS$^{\leqslant} \subseteq k,l$-NTS, and that this inclusion is proper if $k + l > 0$.

The difference between k,l-NTS and k,l-NTS$^{\leqslant}$ grammars can be illustrated with the following example. In the grammar G with $P = \{S \to cba, S \to Aa, A \to b\}$, there is no constituent with a context of size $(0,2)$ so it is trivially $0,2$-NTS. Instead, in $G\bullet^2$, that has $P' = \{S' \to S\bullet^2, S \to cba, S \to Aa, A \to b\}$, every constituent (excluding S') has a context of size $(0,2)$, in particular $S \overset{*}{\Rightarrow} ba$ with context $(\lambda, \bullet^2)$, and $A \overset{*}{\Rightarrow} b$ with $(\lambda, a\bullet)$. But these two constituents occur with the same contexts in $S' \overset{*}{\Rightarrow} cba\bullet^2$, so the $0,2$-NTS$^{\leqslant}$ condition says that $S' \overset{*}{\Rightarrow} cS\bullet^2$ and $S' \overset{*}{\Rightarrow} cAa\bullet^2$ should hold. As this is not true, G is not $0,2$-NTS$^{\leqslant}$.

The WNTS grammars from the previous chapter are related to the k,l-NTS$^{\leqslant}$ grammars. The relationship is shown in the following lemma, and is illustrated in Fig. 4.2.

Lemma 2. *0,0-NTS$^{\leqslant}$ $\subset$ WNTS $\subset$ 1,1-NTS$^{\leqslant}$.*

Proof. See Appendix B. □

4.2 LEARNING ALGORITHM FOR k,l-UNTS$^{\leqslant}$ GRAMMARS

In this section we see intuitively how k,l-UNTS$^{\leqslant}$ grammars can be injectively converted into UNTS grammars. Using this and the **PACCFG** algorithm for UNTS grammars from [13], we will define the learning algorithm **k,l-PACCFG**.

First, remember that in k,l-UNTS$^{\leqslant}$ the substrings are constituents or not, depending on the substrings themselves and on the contexts where they occur, while in UNTS the constituency of the substrings depends only on the substrings themselves. So, in the conversion of

k,l-UNTS$^{\leqslant}$ languages to UNTS languages, we must find a way to encode into the substrings the context where they are occurring each time. A way to do this is to add to each letter the context where it is occurring. To mark the contexts of the letters that are at the boundaries of the sentences, we will use a new terminal $\bullet$. So, we must change the alphabet from Σ to the triplets $\Sigma_\bullet^k \times \Sigma \times \Sigma_\bullet^l$, where $\Sigma_\bullet = \Sigma \cup \{\bullet\}$. To be more compact, we will write the triplets (u, b, v) using subscripts, in the form ${}_u b_v$. For instance, with $(k, l) = (1, 1)$, the string abc would be mapped to ${}_\bullet a_b \, {}_a b_c \, {}_b c_\bullet$. With $(k, l) = (0, 2)$, abc would be mapped to $a_{bc} \, b_{c\bullet} \, c_{\bullet\bullet}$.

This is simply the way to convert a k,l-UNTS$^{\leqslant}$ language into a UNTS language. We use it in the first step of our algorithm in the following way:

Algorithm 1 `k,l-PACCFG`

- **Input:** A sample S. Numbers k, l. Precision ϵ, confidence δ, and some other parameters (see section 4.2.2).
- **Result:** A context-free grammar $\hat{G}$.
- **Steps:**
  1. Convert S into a new sample S' by marking the contexts.
  2. Run **PACCFG** with S' and let $\hat{G}'$ be the resulting grammar.
  3. Remove the marks in $\hat{G}'$ and return the resulting grammar $\hat{G}$.

In the third step, the removal of contextual marks is done on the terminals that occur on the rules, that are known to belong to the alphabet $\Sigma_\bullet^k \times \Sigma \times \Sigma_\bullet^l$.

4.2.1 *Towards a Proof of PAC-Learnability*

At the core of the proof that the presented algorithm **k,l-PACCFG** is PAC is the fact that the given conversion effectively results in a UNTS language. To prove this, we have to give a UNTS grammar that generates the converted language. In this section we illustrate with an example our algorithmic procedure to build the UNTS grammar starting with the original k,l-UNTS$^{\leqslant}$ grammar.

Consider the grammar in Fig. 4.3 (a), that generates the language $\{abc, abd, bdc\}$. It is $1,1$-UNTS$^{\leqslant}$, but it is not UNTS because $S \overset{*}{\Rightarrow} abd$ and $X \overset{*}{\Rightarrow} ab$ but $S \overset{*}{\not\Rightarrow} Xd$. The derivation trees are shown in Fig. 4.3 (a').

Our aim is to change the rules in order to add the context marks in the terminals. We must do this consistently, in a way that the

resulting grammar generates the desired language. In the example, the language must be $\{{}_{\bullet}a_b\ {}_a b_c\ {}_b c_{\bullet}, {}_{\bullet}b_d\ {}_b d_c\ {}_d c_{\bullet}, {}_{\bullet}a_b\ {}_a b_d\ {}_b d_{\bullet}\}$.

For instance, to mark the contexts in the terminals of the rule $X \to ab$, we must first know the contexts where X may appear. Once we know these contexts, we create a new non-terminal and rule for each possible context. As X only occurs in the context $(\bullet, c)$, it will result in a new non-terminal ${}_{\bullet}X_c$, and the rule $X \to ab$ will result in a rule ${}_{\bullet}X_c \to {}_{\bullet}a_b\ {}_a b_c$.

But we come to a problem when we consider rules that have non-terminals in the right side, as in $S \to Xc$. To mark the left context of c, we need a new type of information, that is the last letter that will be generated by X. So, we must also add boundaries marks to all the non-terminals, this is, the initial and final substrings that the non-terminals will generate.

Actually, to be able to mark all the contexts in the grammar, we must first mark completely all the boundaries on the non-terminals. This marking can be done with a bottom-up procedure that starts at the terminal rules and then recursively process the other rules using the already marked non-terminals. The boundaries are marked in the non-terminals using superscripts. In the example, $X \to ab$ and $X \to bd$ are marked as ${}^{a}X^{b} \to ab$ and ${}^{b}X^{d} \to bd$, and after that the rule $S \to Xc$ is marked in two different ways: ${}^{a}S^{c} \to {}^{a}X^{b}c$ and ${}^{b}S^{c} \to {}^{b}X^{d}c$. The resulting grammar and derivations for the example can be seen in Fig. 4.3 (b) and (b′). Just for uniformity, the initial symbol will be denoted ${}^{\bullet}S^{\bullet}$.

Once that all the boundaries have been marked, we can proceed to mark the contexts. This procedure is done top-down, starting at the initial symbol with context $(\bullet, \bullet)$. For instance, in the rule ${}^{a}S^{c} \to {}^{a}X^{b}c$ we know that the left context of c is b, so it will be marked as ${}^{a}_{\bullet}S^{c}_{\bullet} \to {}^{a}_{\bullet}X^{b}_{c}\ {}_b c_{\bullet}$. The resulting grammar and derivations are shown in Fig. 4.3 (c) and (c′).

In Sect. 4.3 we will formally define this procedure, and prove that the resulting grammar has a language that is equal to the converted language, and that if the original grammar is k, l-UNTS$^{\leqslant}$, then the resulting grammar is UNTS.

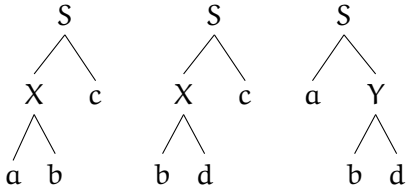
As in [13], to prove that the algorithm is PAC we will assume that the samples are generated by a PCFG. So, we will have to generalize the procedure to PCFGs in a way that the probabilities distributions are preserved.

4.2.2 *Parameters and Bounds*

The two standard main parameters of a PAC-learning algorithm are the precision ϵ and the confidence δ [5]. The precision determines how close the induced instance will be to the hidden instance, and the confidence determines with how much probability. These parameters

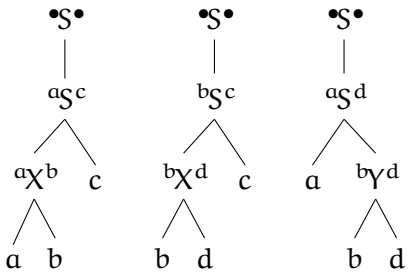
(a)

$$
\begin{aligned}
S &\to Xc \\
S &\to aY \\
X &\to ab \\
X &\to bd \\
Y &\to bd
\end{aligned}
$$



(b)

$$
\begin{aligned}
{\bullet}S{\bullet} &\to {}^{a}S^{c} \\
{\bullet}S{\bullet} &\to {}^{a}S^{d} \\
{\bullet}S{\bullet} &\to {}^{b}S^{c} \\
{}^{a}S^{c} &\to {}^{a}X^{b}\,c \\
{}^{a}S^{d} &\to a\,{}^{b}Y^{d} \\
{}^{b}S^{c} &\to {}^{b}X^{d}\,c \\
{}^{a}X^{b} &\to ab \\
{}^{b}X^{d} &\to bd \\
{}^{b}Y^{d} &\to bd
\end{aligned}
$$



(c)

$$
\begin{aligned}
{\bullet}S{\bullet} &\to {}^{a}_{\bullet}S^{c}_{\bullet} \\
{\bullet}S{\bullet} &\to {}^{b}_{\bullet}S^{c}_{\bullet} \\
{\bullet}S{\bullet} &\to {}^{a}_{\bullet}S^{d}_{\bullet} \\
{}^{a}_{\bullet}S^{c}_{\bullet} &\to {}^{a}_{\bullet}X^{b}_{c}\,{}_{b}c_{\bullet} \\
{}^{b}_{\bullet}S^{c}_{\bullet} &\to {}^{b}_{\bullet}X^{d}_{c}\,{}_{d}c_{\bullet} \\
{}^{a}_{\bullet}S^{d}_{\bullet} &\to {\bullet}a_{b}\,{}^{b}_{a}Y^{d}_{\bullet} \\
{}^{a}_{\bullet}X^{b}_{c} &\to {\bullet}a_{b}\,{}_{a}b_{c} \\
{}^{b}_{\bullet}X^{d}_{c} &\to {\bullet}b_{d}\,{}_{b}d_{c} \\
{}^{b}_{a}Y^{d}_{\bullet} &\to {}_{a}b_{d}\,{}_{b}d_{\bullet}
\end{aligned}
$$

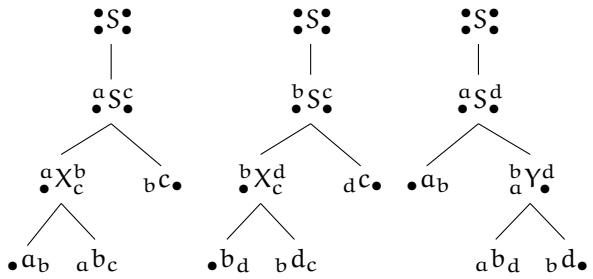


Figure 4.3: Example of conversion of a grammar (a) first adding the boundaries (b) and then adding the contexts (c).

are commonly used in the body of the algorithm and in the definition of the sample complexity. The sample complexity is the minimum number of samples required to guarantee that the algorithm achieves a given precision and confidence [5].

In **PACCFG**, there is a set of additional parameters that stratify the learning problem, stating properties that are assumed to be satisfied by the underlying UNTS PCFGs. There must be known upper bounds for the number of non-terminals, n, for the number of productions, p, and for the length of the right side of the productions, m. There are also parameters that specify distributional properties. There must be a known upper bound L for the expected number of substrings, and there must be known μ_1, ν and μ_2 such that the underlying PCFGs are μ_1-distinguishable, ν-separable and μ_2-reachable. The sample complexity for **PACCFG** is a function of all these parameters $N(\mu_1, \mu_2, \nu, n, p, m, L, \delta, \epsilon)$. The exact formula for N is rather complex and is not of special interest. We just observe that N is $\mathcal{O}(\frac{n+p}{\epsilon \mu_1^m \mu_2^2 \nu^2})$. We refer the reader to [13] for the details of the parameters of **PACCFG** and for the definition of μ_1-distinguishability, ν-separability and μ_2-reachability.

As our algorithm **k,l-PACCFG** is defined in terms of **PACCFG**, it also stratifies the learning problem. We will assume that the underlying k, l-UNTS$^{\leqslant}$ PCFGs have the mentioned bounds n, p, m and L, and a new bound o for the number of non-terminals in the right sides ($o \leqslant m$). Knowing these bounds, it is possible to compute the corresponding bounds for the UNTS PCFGs that are the result of converting the k, l-UNTS$^{\leqslant}$ PCFGs with the process described in Sect. 4.2. We will show this in Sect. 4.3.

There will be a different treatment for the parameters μ_1, ν and μ_2. We will directly assume that the k, l-UNTS$^{\leqslant}$ PCFGs are such that, when converted, the resulting UNTS PCFGs are μ_1-distinguishable, ν-separable and μ_2-reachable.

4.3 PROOF OF PAC-LEARNABILITY

In this section we give the formal elements required to prove that our algorithm is PAC. In the first place, we show how to convert a CFG using the Left Marked Form and then the Right Contextualized Grammar. Then we show that when these conversions are applied to a k, l-UNTS$^{\leqslant}$ grammar with $l > 0$, they return a $k, l-1$-UNTS$^{\leqslant}$ grammar. Then, we extend the conversion procedure to PCFGs, showing that the distributions are preserved. Finally, we see that these results give a conversion from k, l-UNTS$^{\leqslant}$ to UNTS PCFGs and prove that our algorithm is PAC.

4.3.1 *The Left Marked Form of a Grammar*

The Left Marked Form of a grammar adds a mark to each non-terminal that states which is the first terminal it generates, without changing the shape of the rules and the generated language. It is constructed using a recursive bottom-up procedure as described intuitively in Sect. 4.2.

Definition 13. *Let* $G = \langle \Sigma, N, S, P \rangle$ *be a CFG. Let* $G'_i = \langle \Sigma, N'_i, {}^{\bullet}S, P'_i \rangle$, $i \geqslant 0$ *be such that*

$$
\begin{aligned}
N'_i &= \{ {}^{\bullet}S \} \cup \{ {}^{a}X |\ {}^{a}X \to \alpha \in P'_i \} \\
P'_0 &= \{ {}^{a}X \to a s | X \to a s \in P \} \\
P'_{n+1} &= \{ {}^{a}X \to u_1\ {}^{a_1}X_1 u_2 \dots u_m\ {}^{a_m}X_m u_{m+1} | \forall i\ {}^{a_i}X_i \in N'_n \text{ and} \\
&\qquad X \to u_1 X_1 u_2 \dots u_m X_m u_{m+1} \in P \text{ and} \\
&\qquad a = (u_1)_0 \text{ if } u_1 \neq \lambda, a = a_1 \text{ otherwise} \} \cup \\
&\qquad \{ {}^{\bullet}S \to {}^{a}S |\ {}^{a}S \in N'_n \}
\end{aligned}
$$

If there exists k *such that* $G'_{k+1} = G'_k$, *the* Left Marked Form (LMF) *of* G *is the grammar* $G' = G'_k$.

Lemma 3. *The LMF always exists and is unique.*

Proof. See Appendix B. □

The sequence of grammars $G'_0, G'_1, \dots$ represents the steps in the procedure. Lemma 3 states that this procedure converges. The following lemmas state the fundamental properties of the LMF.

Lemma 4. *Let* G *be a CFG and* G' *its LMF. Then, for all* $X, m \geqslant 0$, $u_1, \dots, u_{m+1}, X_1, \dots, X_m$,

$$X \overset{*}{\Rightarrow}_G u_1 X_1 u_2 \dots u_m X_m u_{m+1}$$

if and only if

there exist $a_1, \dots, a_m$ *such that* ${}^{a}X \overset{*}{\Rightarrow}_{G'} u_1\ {}^{a_1}X_1 u_2 \dots u_m\ {}^{a_m}X_m u_{m+1}$

with $a = (u_1)_0$ *if* $u_1 \neq \lambda, a = a_1$ *otherwise.*

Proof. See Appendix B for a sketch. □

Lemma 5. $L(G') = L(G)$.

Proof. Corollary of Lemma 4 for the case $m = 0$ and $X = S$. □

For the parameter conversion of Sect. 4.2.2, we must observe that if n, p and o are bounds for G, then $n' = n|\Sigma| + 1$, $p' = p|\Sigma|^o$ and $o' = o$ are bounds for G'.

4.3.2 *The Right Contextualized Grammar of a Grammar*

The Right Contextualized Grammar adds a mark to each terminal that says which terminal goes next, while preserving the shape of the rules. As we saw intuitively in Sect. 4.2, it uses the LMF and is constructed with a recursive top-down procedure. The procedure involves changing the right side of the LMF rules to add the right contexts. To do this we use the following definition.

Definition 14. *Let* G *be a CFG and* G' *its LMF. Let* $\alpha \in (\Sigma \cup N')^*$ *and* $b \in \Sigma$. *Then, the* right-contextualization of α with final context b, $rc_b(\alpha)$, *is recursively defined by*

$$\begin{aligned} rc_b(\lambda) &= \lambda \\ rc_b(\alpha a) &= rc_a(\alpha) a_b \\ rc_b(\alpha\, {}^{a}X) &= rc_a(\alpha)\, {}^{a}X_b \end{aligned}$$

The right-contextualization of a language L *is* $rc(L) = \{rc_\bullet(s) | s \in L\}$.

Definition 15. *Let* $G = \langle \Sigma, N, S, P \rangle$ *be a CFG and* $G' = \langle \Sigma, N', {}^{\bullet}S, P' \rangle$ *its LMF. Let* $G''_i = \langle \Sigma \times \Sigma, N''_i, {}^{\bullet}S_\bullet, P''_i \rangle, i \geqslant 0$ *be such that*

$$\begin{aligned} N''_n &= \{ {}^{\bullet}S_\bullet \} \cup \{ {}^{a}X_b | {}^{c}Y_d \to \alpha\, {}^{a}X_b \beta \in P''_n \} \\ P''_0 &= \emptyset \\ P''_{n+1} &= \{ {}^{a}X_b \to rc_b(\alpha) | {}^{a}X_b \in N''_n \text{ and } {}^{a}X \to \alpha \in P' \} \end{aligned}$$

If there exists k *such that* $G''_{k+1} = G''_k$, *the* Right Contextualized Grammar (RCG) *of* G *is the grammar* $G'' = G''_k$.

Lemma 6. *The RCG always exists and is unique.*

Proof. Analogous to proof of Lemma 3. □

The following lemmas state the fundamental properties of the RCG.

Lemma 7. *Let* G *be a CFG,* G' *its LMF and* G'' *its RCG. Then, for all* a, b, X, α, ${}^{a}X_b \overset{*}{\Rightarrow}_{G''} \alpha$ *if and only if there is* α_0 *such that* $\alpha = rc_b(\alpha_0)$ *and* ${}^{a}X \overset{*}{\Rightarrow}_{G'} \alpha_0$.

Proof. Proof sketch analogous to proof sketch of Lemma 4 in Appendix B. □

Lemma 8. $L(G'') = rc(L(G))$.

Proof. Corollary of Lemma 7 for the case ${}^{a}X_b = {}^{\bullet}S_\bullet$ and $\alpha \in \Sigma^*$ (also using Lemma 5). □

For the parameter conversion of Sect. 4.2.2, we must observe that if n', p' and o' are bounds for G', then $n'' = (n' - 1)(|\Sigma| + 1) + 1$, $p'' = p'(|\Sigma| + 1)$ and $o'' = o'$ are bounds for G''.

4.3.3 *Converting* k, l*-UNTS*$^{\leqslant}$ *Grammars*

Lemma 9. *Let* G *be a* k, l*-NTS*$^{\leqslant}$ *grammar. Then, its LMF* G' *is also* k, l*-NTS*$^{\leqslant}$.

Proof. See Appendix B for a sketch. □

Lemma 10. *Let* G *be a* k, l*-UNTS*$^{\leqslant}$ *grammar with* $l > 0$. *Then, its RCG* G'' *is* $k, l-1$*-UNTS*$^{\leqslant}$.

Proof sketch. Here we only show the proof that G'' is $k, l-1$-NTS$^{\leqslant}$, omitting the unambiguity part. Unambiguity is not difficult but cumbersome in notation. Let $H' = \bullet^k G' \bullet^l$ and $H'' = \bullet^k G'' \bullet^{l-1}$. Then, we must prove that H'' is $k, l-1$-NTS knowing that H' is k, l-NTS.

Suppose that

- ${}^a X_b \overset{*}{\Rightarrow}_{H''} \alpha u \beta v \gamma$,
- ${}^c Y_d \overset{*}{\Rightarrow}_{H''} \beta$ and
- ${}^\bullet S_\bullet \overset{*}{\Rightarrow}_{H''} \alpha' u \, {}^c Y_d v \gamma'$, with $|u| = k$, $|v| = l-1$.

We have to show that ${}^a X_b \overset{*}{\Rightarrow}_{H''} \alpha u \, {}^c Y_d v \gamma$. To do this, we must try to generate the three conditions over H' that let us apply the hypothesis that it is k, l-NTS.

First, we see that if ${}^c Y_d = {}^\bullet S_\bullet$, then everything is λ except β that has the form $\bullet^k \beta' \bullet^{l-1}$, so ${}^a X_b$ is also ${}^\bullet S_\bullet$, ${}^a X_b \overset{*}{\Rightarrow}_{H''} \alpha u \, {}^c Y_d v \gamma$ is equivalent to ${}^\bullet S_\bullet \overset{*}{\Rightarrow}_{H''} {}^\bullet S_\bullet$ and we are done with the proof. So, we can continue assuming ${}^c Y_d \neq {}^\bullet S_\bullet$.

- Condition 1: It is derived from ${}^a X_b \overset{*}{\Rightarrow}_{H''} \alpha u \beta v \gamma$ but will not talk about ${}^a X$. Instead, we will go back to ${}^\bullet S_\bullet$ using that there are α'', γ'' such that ${}^\bullet S_\bullet \overset{*}{\Rightarrow}_{H''} \alpha'' \, {}^a X_b \gamma'' \overset{*}{\Rightarrow}_{H''} \alpha'' \alpha u \beta v \gamma \gamma''$. Moving this derivation to G'', applying Lemma 7 to go to G', and adding again the $\bullet$ markers to go to H', we can see that ${}^\bullet S \overset{*}{\Rightarrow}_{H'} \alpha''_0 \alpha_0 u_0 \beta_0 v_0 \gamma_0 \gamma''_0 \bullet$. We still do not have the desired condition because we need a right context of size l.
- Condition 2: Since ${}^c Y_d \neq {}^\bullet S_\bullet$, then β does not have any $\bullet$, and also in G'' ${}^c Y_d \overset{*}{\Rightarrow}_{G''} \beta$. Using Lemma 7, we have ${}^c Y \overset{*}{\Rightarrow}_{G'} \beta_0$ and also in H' ${}^c Y \overset{*}{\Rightarrow}_{H'} \beta_0$, so we have the second condition.
- Condition 3: Using that ${}^\bullet S_\bullet \overset{*}{\Rightarrow}_{H''} \alpha' u \, {}^c Y_d v \gamma'$ and applying the same arguments used in the first condition we can see that ${}^\bullet S \overset{*}{\Rightarrow}_{H'} \alpha'_0 u_0 \, {}^c Y v_0 \gamma'_0 \bullet$. Again, we still need to find a right context of size l.
- Conditions 1 and 3: In order to find the correct right context for both conditions we start observing that either $\gamma_0 \gamma''_0$ and γ'_0 must be both λ, or both must derive the same first terminal e. In the first case the right context will be $v_0 \bullet$ and in the second

case it will be $\nu_0 e$, and in both cases we will have the desired conditions.

Having the conditions, we apply the hypothesis to obtain in H' a derivation of the form ${}^{\bullet}S \overset{*}{\Rightarrow}_{H'} \alpha_0'' \alpha_0 u_0 {}^{c}Y \nu_0 e \delta_0$. Moving to G', applying Lemma 7 to go to G'' and then to H'' adding again the $\bullet$ markers, we have that ${}^{\bullet}S_{\bullet} \overset{*}{\Rightarrow}_{H''} \alpha'' \alpha u {}^{c}Y_d \nu e_f \delta$ for some f. Now, in H'' we have two different ways to derive the same thing:

$$ {}^{\bullet}S_{\bullet} \overset{*}{\Rightarrow} \left\{ \begin{array}{c} \overset{*}{\Rightarrow} \alpha'' \alpha u {}^{c}Y_d \nu e_f \delta \overset{*}{\Rightarrow} \\ \overset{*}{\Rightarrow} \alpha'' {}^{a}X_b \gamma'' \overset{*}{\Rightarrow} \alpha'' \alpha u \beta \nu \gamma \gamma'' \overset{*}{\Rightarrow} \end{array} \right\} \overset{*}{\Rightarrow} \alpha'' \alpha u \beta \nu e_f \delta $$

Since H'' is unambiguous, these two derivations must be the same tree, enforcing ${}^{a}X_b \overset{*}{\Rightarrow}_{H''} \alpha u {}^{c}Y_d \nu \gamma$. □

4.3.4 *Extending the results to PCFGs*

In this section we extend the definitions of Left Marked Form and Right Contextualized Grammar to PCFGs so that the probabilities of the derivations are preserved.

In the case of the LMF, the derivations have the same tree structure as in the original grammar but using an additional initial rule of the form ${}^{\bullet}S \to {}^{a}S$. So, if every rule that comes from a rule of the original grammar takes the same probability as the original rule, and all the new initial rules have probability 1, the probabilities of the derivations will be preserved. The problem is that we will not end up with a PCFG but a Weighted CFG (WCFG), because the probabilities of the rules for ${}^{\bullet}S$ may sum more than 1 and the rules for the other non-terminals may sum less than 1. For instance, the rules $A \overset{p_1}{\to} a, A \overset{p_2}{\to} b$ will be mapped to the non-PCFG rules ${}^{a}A \overset{p_1}{\to} a$, ${}^{b}A \overset{p_2}{\to} b$. To solve this, we can use the renormalization formula presented in [2] to convert the WCFG back to a PCFG with the same language probability distribution.

Fortunately, in the case of the RCG the derivations have the same tree structure as in the LMF, and every rule comes from a rule of the LMF, so the probabilities can be directly propagated. Despite the fact that the non-terminals are more granular, in this case we always end up with a PCFG without the need of renormalization. The LMF rules ${}^{a}A \overset{p_1}{\to} \alpha_1, \ldots, {}^{a}A \overset{p_n}{\to} \alpha_n$ will be mapped in the RCG to several sets of rules of the form ${}^{a}A_b \overset{p_1}{\to} rc_b(\alpha_1), \ldots, {}^{a}A_b \overset{p_n}{\to} rc_b(\alpha_n)$, each set with a different b, and each set summing 1.

All these observations are summarized in the following definition and lemma:

Definition 16. *Let G be a PCFG with production probability π.*

The Left Marked Form (LMF) *of* G *is the PCFG* G' *with production probability* π' *such that*

$$\pi'({}^{a}X \to \alpha) = w'({}^{a}X \to \alpha)\frac{\prod_i \|(\alpha)_i\|}{\|{}^{a}X\|}$$

where

$$w'({}^{a}X \to \alpha) = \begin{cases} 1 & \text{if } {}^{a}X = {}^{\bullet}S \\ \pi(X \to u_1 X_1 u_2 \dots u_n X_n u_{n+1}) & \text{otherwise,} \\ \text{where } \alpha = u_1\, {}^{a_1}X_1 u_2 \dots u_n\, {}^{a_n}X_n u_{n+1} & \end{cases}$$

and

$$\|{}^{a}X\| = \sum_s W_{w'}({}^{a}X \overset{*}{\Rightarrow}_{G'} s)$$

for ${}^{a}X \in N'$ *and* $\|a\| = 1$ *for* $a \in \Sigma$.

The Right Contextualized Grammar (RCG) *of* G *is the PCFG* G'' *with production probability* π'' *such that*

$$\pi''({}^{a}X_b \to rc_b(\alpha)) = \pi'({}^{a}X \to \alpha).$$

In this definition, $W_{w'}$ refers to the weight of a derivation where the weights of the rules are given by w' [2].

Lemma 11. *Let* G *be a PCFG,* G' *its LMF and* G'' *its RCG. Then, for every* $s \in L(G)$, $P_G(s) = P_{G'}(s) = P_{G''}(rc_\bullet(s))$.

4.3.5 *The Theorems*

We will call $mark_{k,l}$ to the function that marks the contexts of a language in Σ^*, and $unmark_{k,l}$ to the function that unmarks the contexts. For convenience, the domain of $unmark$ will be any language in $(\Sigma_\bullet^k \times \Sigma \times \Sigma_\bullet^l)^*$, not only the ones that are the result of marking a language.

Theorem 4. *Let* G *be a* k,l*-UNTS*$^{\leqslant}$ *PCFG. Then, there is a UNTS PCFG* G' *such that*

1. $L(G') = mark_{k,l}(L(G))$,
2. *and for every* $s \in L(G)$, $P_G(s) = P_{G'}(mark_{k,l}(s))$.

Proof sketch. Do induction on k and l, starting with $k = l = 0$, then continuing with $k = 0, l > 0$ using the RCG, and finally with $k, l > 0$, using a symmetric version of the RCG. See Appendix B for a detailed proof. □

Theorem 5. *Given* δ *and* ϵ*, there is* N *such that, if* S *is a sample of a* k,l*-UNTS*$^{\leqslant}$ *PCFG* G *with* $|S| > N$*, then with probability greater than* $1 - \delta$, $\hat{G} = \textbf{\textit{k,l-PACCFG}}(S)$ *is such that*

1. $L(\hat{G}) \subseteq L(G)$, *and*
2. $P_G(L(G) - L(\hat{G})) < \epsilon$.

Proof. Let G' be the UNTS conversion of G. If the known bounds for G are of n, p, m, o and L, the corresponding bounds for G' are $n' = n(|\Sigma|(|\Sigma|+1))^{k+l} + 1$, $p' = p|\Sigma|^{(k+l)o}(|\Sigma|+1)^{k+l}$, $m' = m$ and $L' = L$.

Let $N = N(\mu_1, \mu_2, \nu, n', p', m', L', \delta, \epsilon)$, and suppose that $|S| > N$. As defined in step 1 of the **k,l-PACCFG** algorithm, S' is a sample of G' and $|S'| = |S| > N$. Then, by Clark's PAC-learning theorem, step 2 returns $\hat{G}'$ that with probability $1-\delta$, $L(\hat{G}') \subseteq L(G')$, and $P_{G'}(L(G') - L(\hat{G}')) < \epsilon$. Now, let $\hat{G}$ be the result of step 3. Then,

$$\begin{aligned} L(\hat{G}) &= unmark_{k,l}(L(\hat{G}')) \\ &\subseteq unmark_{k,l}(L(G')) \text{ (with probability } 1-\delta) \\ &= L(G). \end{aligned}$$

Also, $mark_{k,l}(L(\hat{G})) = L(\hat{G}')$ with probability $1-\delta$ because $L(\hat{G}') \subseteq L(G') = mark_{k,l}(L(G))$ with the same probability. So,

$$\begin{aligned} P_G(L(G) - L(\hat{G})) &= P_{G'}(mark_{k,l}(L(G) - L(\hat{G}))) \\ &= P_{G'}(mark_{k,l}(L(G)) - mark_{k,l}(L(\hat{G}))) \\ &= P_{G'}(L(G') - mark_{k,l}(L(\hat{G}))) \\ &= P_{G'}(L(G') - L(\hat{G}')) \text{ (with probability } 1-\delta) \\ &< \epsilon. \end{aligned}$$

□

4.4 DISCUSSION

As the results presented here are heavily based in the results of [13], the discussion section of that paper applies entirely to our work.

The sample complexity of our learning algorithm is

$$\mathcal{O}\left(\frac{n'+p'}{\epsilon\mu_1^{m'}\mu_2^2\nu^2}\right) = \mathcal{O}\left(\frac{n|\Sigma|^{2(k+l)} + p|\Sigma|^{(k+l)(o+1)}}{\epsilon\mu_1^{m}\mu_2^2\nu^2}\right).$$

As pointed in [13], the m exponent is worrying, and we can say the same about the o exponent. A small restriction on the UNTS grammars can guarantee conversion to CNF preserving the UNTS property, giving $2 \geqslant m \geqslant o$. However, it is not clear to us how this restriction affects the k,l-UNTS$^{\leqslant}$ grammars.

In this work we directly assume known values of μ_1, ν and μ_2 for the converted UNTS grammars. We do this because these values can not be computed from the corresponding values for the original k,l-UNTS$^{\leqslant}$ grammars as can be done with the other parame-

ters. We can define new ad-hoc properties that when valid over k, l-UNTS$^{\leqslant}$ grammars imply μ_1-distinguishability, ν-separability and μ_2-reachability over the converted grammars. However, these properties are in essence equivalent to the given assumptions.

We could not use our approach to prove PAC-learnability of all the k, l-UNTS languages. Using a modified version of the conversion process we could show that k, l-UNTS languages with $k + l > 0$ can be converted to $0, 1$; $1, 0$ or $1, 1$-UNTS. However, we could not manage to give a conversion to UNTS, and we believe it is actually not possible. Anyway, PAC-learnability of k, l-UNTS languages is not of special interest to us because we find k, l-UNTS$^{\leqslant}$ more suitable to model natural language.

Part III

NON-DETERMINISTIC DEPENDENCY PARSING

5

A SPECTRAL ALGORITHM FOR MODELING MODIFIER SEQUENCES

Split Head-Automata Grammars (SHAGs) [22] use head-modifier sequence modeling to generate dependency structures. In SHAGs, the modifier sequences are modeled by a collection of probabilistic deterministic automata.

In this chapter we present modifier sequence modeling using, probabilistic non-deterministic finite state automata (PNFA) which we parametrize using the *operator model* representation. This representation allows us to use simple spectral algorithms for estimating the model parameters from a dependency treebank. In all previous work, the algorithm used to induce hidden structure requires running repeated inference on training data—e.g. split-merge algorithms, or EM algorithms. In contrast, the spectral method we use is simple and very efficient —parameter estimation is reduced to computing some corpus statistics, performing SVD and inverting matrices.

5.1 NOTATION AND DEFINITIONS

5.1.1 *Head-Automata Dependency Grammars*

In this work we use Split-Head Automata Grammars (SHAGs) [22, 21], a context-free grammatical formalism whose derivations are projective dependency trees. We will use $x_{i:j} = x_i x_{i+1} \dots x_j$ to denote a sequence of symbols x_t with $i \leqslant t \leqslant j$. A SHAG generates sentences $x_{0:n}$, where symbols $x_t \in \mathcal{X}$ with $t \geqslant 1$ are regular words and $x_0 = \star \notin \mathcal{X}$ is a special root symbol. Let $\bar{\mathcal{X}} = \mathcal{X} \cup \{\star\}$. A derivation y, i.e. a dependency tree, is a collection of *head-modifier* sequences $\langle h, d, m_{1:T} \rangle$, where $h \in \bar{\mathcal{X}}$ is a word, $d \in \{\textsc{left}, \textsc{right}\}$ is a direction, and $m_{1:T}$ is a sequence of T words, where each $m_t \in \mathcal{X}$ is a *modifier* of h in direction d. We say that h is the *head* of each m_t. Sequences $m_{1:T}$ are ordered head-outwards, i.e. among $m_{1:T}$, m_1 is the word closest to h in the derived sentence, and m_T is the furthest. A derivation y of a sentence $x_{0:n}$ consists of a LEFT and a RIGHT head-modifier sequence for each x_t. As special cases, the LEFT sequence of the root symbol is always empty, while the RIGHT one consists of a single word

corresponding to the head of the sentence. We denote by $\mathcal{Y}$ the set of all valid derivations.

Assume a derivation y contains $\langle h, \textsc{left}, m_{1:T}\rangle$ and $\langle h, \textsc{right}, m'_{1:T'}\rangle$. Let $\mathcal{L}(y, h)$ be the derived sentence *headed* by h, which can be expressed as $\mathcal{L}(y, m_T)\dots\mathcal{L}(y, m_1)\ h\ \mathcal{L}(y, m'_1)\dots\mathcal{L}(y, m'_{T'})$.[1] The language generated by a SHAG are the strings $\mathcal{L}(y, \star)$ for any $y \in \mathcal{Y}$.

In this thesis we use probabilistic versions of SHAGs where the probabilities of head-modifier sequences in a derivation are independent of each other,

$$\mathbb{P}(y) = \prod_{\langle h,d,m_{1:T}\rangle \in y} \mathbb{P}(m_{1:T}|h, d) \quad . \tag{5.1}$$

In the literature, standard *arc-factored* models further assume that

$$\mathbb{P}(m_{1:T}|h, d) = \prod_{t=1}^{T+1} \mathbb{P}(m_t|h, d, \sigma_t) \quad , \tag{5.2}$$

where m_{T+1} is set as a special $\textsc{stop}$ word, and σ_t is the state of a deterministic automaton generating $m_{1:T+1}$. For example setting $\sigma_1 = \textsc{first}$ and $\sigma_{t>1} = \textsc{rest}$ corresponds to first-order models, while setting $\sigma_1 = \textsc{null}$ and $\sigma_{t>1} = m_{t-1}$ corresponds to sibling models [21, 41, 40].

5.1.2 *Operator Models*

An operator model F of size S consists of a set of *operator* matrices, namely: for each $x \in \mathcal{X}$, a matrix $A_x \in \mathbb{R}^{S\times S}$; and two vectors $\alpha_1 \in \mathbb{R}^S$ and $\alpha_\infty \in \mathbb{R}^S$. The automaton F computes a function $f : \mathcal{X}^* \to \mathbb{R}$ as follows:

$$f(x_{1:T}) = \alpha_\infty^\top A_{x_T} \cdots A_{x_1} \alpha_1 \quad . \tag{5.3}$$

Operator models have had numerous applications. For example, they can be used as an alternative parametrization of the function computed by an HMM [28]. Consider an HMM with S hidden states and initial-state probabilities $\pi \in \mathbb{R}^S$, transition probabilities $T \in \mathbb{R}^{S\times S}$, and observation probabilities $O_x \in \mathbb{R}^{S\times S}$ for each $x \in \mathcal{X}$, with the following meaning:

- $\pi(i)$ is the probability of starting at state i.
- $T(i, j)$ is the probability of transitioning from state j to state i.
- O_x is a diagonal matrix, such that $O_x(i, i)$ is the probability of generating symbol x from state i.

1 Throughout this chapter we assume we can distinguish the words in a derivation, irrespectively of whether two words at different positions correspond to the same symbol.

Given an HMM, an equivalent operator model can be defined by setting $\alpha_1 = \pi$, $A_x = TO_x$ and $\alpha_\infty = \vec{1}_{1:S}$. To see this, let us show that the computation in Eq.5.3 implements the forward algorithm. Let σ_t denote the state of the HMM at time t. Consider a state-distribution vector $\alpha_t \in \mathbb{R}^m$, where $\alpha_t(i) = \mathbb{P}(x_{1:t-1}, \sigma_t = i)$. Initially $\alpha_1 = \pi$ by definition of π. At each step in the chain of products of Eq.5.3, $\alpha_{t+1} = A_{x_t}\ \alpha_t$ updates the state distribution from positions t to $t+1$ by applying the appropriate operator, i.e. by emitting symbol x_t and transitioning to the new state distribution. The probability of $x_{1:T}$ is given by $\sum_i \alpha_{T+1}(i)$. Hence, $A_x(i,j)$ is the probability of moving generating symbol x and moving to state i given that we are at state j.

An HMM is only an example of the distributions that can be parameterized by an operator model. In general the operator models can parameterize any PNFA, where probabilities of the model are associated to emmiting a symbol from a state *and* moving to the next state.

The advantage of working with operator models is that, under certain mild assumptions on the operator parameters, there exist algorithms that can estimate the operators from observable statistics of the input sequences. These algorithms are extremely efficient and are not susceptible to local minima issues. See [28] for theoretical proofs of the learnability of HMMs under the operator model representation.

5.2 LEARNING OPERATOR MODELS

For each possible head h in the vocabulary $\bar{\mathcal{X}}$ and each direction $d \in \{\text{LEFT}, \text{RIGHT}\}$ we have an operator model that computes probabilities of modifier sequences, as follows:

$$\mathbb{P}(m_{1:T}|h,d) = (\alpha_\infty^{h,d})^\top\ A_{m_T}^{h,d}\ \cdots\ A_{m_1}^{h,d}\ \alpha_1^{h,d}\ . \tag{5.4}$$

To learn the model parameters, namely $\alpha_1^{h,d}$, $A_m^{h,d}$ and $\alpha_\infty^{h,d}$, we use spectral learning methods based on the work of [28].

The main challenge of learning an operator model is to infer the hidden-state space from observable quantities, i.e. quantities that can be computed from the distribution of sequences that we observe. As it turns out, we can not recover the actual hidden-state space used by the operators we wish to learn. The key insight of the spectral learning method is that we can recover a hidden-state space that corresponds to a projection of the original hidden space. Such projected space is equivalent to the original one in the sense that we can define operators in the projected space that parameterize the *same* probability distribution over sequences.

In the rest of this section we describe an algorithm to learn an operator model. We will assume a fixed head word and direction, and drop h and d from all terms. Hence, our goal is to learn the following

distributon, parameterized by operators α_1, A_m for all $m \in \mathcal{X}$, and α_∞:

$$\mathbb{P}(m_{1:T}) = (\alpha_\infty)^\top A_{m_T} \cdots A_{m_1} \alpha_1 \quad . \tag{5.5}$$

5.2.1 *Preliminary Definitions*

We start by defining some probability matrices over sequences of the language. For convenience, assume that $\mathcal{X} = \{1, \ldots, l\}$, so that we can index vectors and matrices by symbols in $\mathcal{X}$. We define $P \in \mathbb{R}^{l \times l}$ as a matrix of marginal probabilities over bigrams of modifiers for a given head and direction. That is, for any two symbols $a, b \in \mathcal{X}$:

$$P(b, a) = \mathbb{P}(m_t = a, m_{t+1} = b) \quad .$$

In the following, we use $x = x_{i:j} \in \mathcal{X}^*$ to denote sequences of symbols, and we use $A_{x_{i:j}}$ as a shorthand for $A_{x_j} \cdots A_{x_i}$. We can express P as

$$\begin{aligned} P(b, a) &= \textstyle\sum_{p \in \mathcal{X}^*} \sum_{s \in \mathcal{X}^*} \mathbb{P}(p\ a\ b\ s) \\ &= \textstyle\sum_{p \in \mathcal{X}^*} \sum_{s \in \mathcal{X}^*} \alpha_\infty^\top A_s\ A_b\ A_a\ A_p\ \alpha_1 \\ &= \alpha_\infty^\top \left(\textstyle\sum_{s \in \mathcal{X}^*} A_s\ A_b\right) \left(A_a \textstyle\sum_{p \in \mathcal{X}^*} A_p\right) \alpha_1 \\ &= B(b)\ F(a)^\top \end{aligned} \tag{5.6}$$

In the last step we have used the following definitions:

- $F \in \mathbb{R}^{l \times S}$ is a matrix of marginal *forward* states for each symbol $a \in \mathcal{X}$. That is, the row for a contains the sum of hidden-state vectors after generating prefixes that end with a. Formally,

  $$F(a) = \left(A_a \left(\textstyle\sum_{p \in \mathcal{X}^*} A_p\right) \alpha_1\right)^\top$$

- $B \in \mathbb{R}^{l \times S}$ is a matrix of marginal *backward* states for each symbol $a \in \mathcal{X}$. That is, the row for a contains the sum of hidden-state vectors that generate suffixes that start with a. Formally,

  $$B(a) = \alpha_\infty \left(\textstyle\sum_{s \in \mathcal{X}^*} A_s\right) A_a$$

With the definitions of F and B, it is easy to see that $P = B\ F^\top$. Let us now define matrices $P_b \in \mathbb{R}^{l \times l}$ for each symbol $b \in \mathcal{X}$ of marginal probabilities over trigrams of modifiers. That is, for any three symbols $a, b, c \in \mathcal{X}$:

$$P_b^{h,d}(c, a) = \mathbb{P}(m_{t-1} = a, m_t = b, m_{t+1} = c \mid h, d) \quad .$$

Using a similar derivation as above, we get that:

$$\begin{aligned} P_b(c, a) &= \textstyle\sum_{p \in \mathcal{X}^*} \sum_{s \in \mathcal{X}^*} \mathbb{P}(p\ a\ b\ s) \\ &= \textstyle\sum_{p \in \mathcal{X}^*} \sum_{s \in \mathcal{X}^*} \alpha_\infty^\top A_s\ A_c\ A_b\ A_a\ A_p\ \alpha_1 \\ &= B(c)\ A_b\ F(a)^\top \end{aligned} \tag{5.7}$$

Finally, let $p_1 \in \mathbb{R}^l$ and $p_\infty \in \mathbb{R}^l$ be the marginal probability vectors over symbols starting or ending sequences of the language: $p_1(a) = \mathbb{P}(m_1 = a)$ and $p_\infty(a) = \sum_{T \geqslant 1} \mathbb{P}(m_T = a)$. It is easy to see that $p_1 = B\ \alpha_1$ and $p_\infty = \alpha_\infty^\top\ F^T$.

5.2.2 *Inducing a Hidden-State Space*

We now turn into inducing the hidden space $\mathbb{R}^S$ where values $B(a)$ and $F(a)$ live. We know that $P = B\ F^\top$, however we can not recover this particular factorization from observable quantities. This is because there are multiple ways of factorizing P into matrices $\mathbb{R}^{l \times S}$ and $\mathbb{R}^{S \times l}$.

The key insight from the theory behind these methods is that we do not need to know this particular factorization to estimate the parameters of the distribution. We can use other factorizations of P that can be recovered (i.e. they are unique) to estimate operators that define the *same* distribution.

More precisely, we will use a thin SVD decomposition of $P = U\Sigma V^\top$. We can think of the rows of $U \in \mathbb{R}^{l \times S}$ as projected backward-state vectors, and of the columns of $\Sigma V^\top = U^\top P \in \mathbb{R}^{S \times l}$ as projected forward-state vectors. Such projected backward and forward vectors are in a hidden-state space that is not the same as in the original distribution, but they can be used to compute the same distribution.

We can give an intuitive representation of this state space by looking at $U^\top P$. The columns of this matrix are computed in two steps. First we take the a-th column of P, which describes a by listing the marginal probability of ab for each symbol b. Second, we take the inner-product between this vector and each of the columns of U. Thus we can think of the columns of U as centroids that define a soft-partitioning of the space of into S classes.

5.2.3 *Recovering Observable Operators*

With the SVD factorization, we want to find, for every symbol $a \in \mathcal{X}$, an operator $\hat{A}_x$ such that $P_x = U\hat{A}_x(U^\top P)$. We can recover the operator as follows:

$$\begin{aligned}
\hat{A}_x &= U^\top P_x (U^\top P)^+ && (5.8)\\
&= U^\top B A_x F^\top (U^\top B F^\top)^+ && (5.9)\\
&= (U^\top B) A_x (U^\top B)^+ && (5.10)\\
&= Q\ A_x Q^+ && (5.11)
\end{aligned}$$

Thus, using the SVD factorization we can recover operators $\hat{A}_x$, which correspond to a projection of the original operators using $Q = U^\top B$.

Algorithm 2 `LearnOperatorSHAG`

Input:

- $\mathcal{X}$: an alphabet
- TRAIN $= \{\langle h^1, d^1, m^1_{1:T}\rangle, \ldots, \langle h^n, d^n, m^n_{1:T}\rangle\}$: a training set with n head-modifier sequences
- S : number of hidden states

Output:

- Observable operators $\hat{\alpha}_1^{h,d}$, $\hat{\alpha}_\infty^{h,d}$ and $\hat{A}_x^{h,d}$ for all $h \in \bar{\mathcal{X}}$, $d \in \{\text{LEFT}, \text{RIGHT}\}$, $x \in \mathcal{X}$

For each $h \in \bar{\mathcal{X}}$ and $d \in \{\text{LEFT}, \text{RIGHT}\}$:

1. Use TRAIN to compute an empirical estimate of the probability matrices p_1, p_∞ , P, and P_m for each $m \in \mathcal{X}$
2. Let $P = U\Sigma V^\top$. Take U to be the matrix of top S left singular vectors of P
3. Compute the observable operators for h and d:
   - $\hat{\alpha}_1^{h,d} = Up_1$,
   - $(\hat{\alpha}_\infty^{h,d})^\top = p_\infty \left(U^\top P\right)^+$
   - $\hat{A}_m^{h,d} = U^\top P_m (U^\top P)^+$ for each $m \in \mathcal{X}$

The initial and end operators are given by:

$$\hat{\alpha}_1 = Up_1 = U^\top B\ \alpha_1 = Q\ \alpha_1 \tag{5.12}$$

$$\hat{\alpha}_\infty^\top = p_\infty \left(U^\top P\right)^+ \tag{5.13}$$

$$= \alpha_\infty^\top\ F^\top (U^\top B F^\top)^+ = \alpha_\infty^\top\ Q^+ \tag{5.14}$$

With these observable operators we can compute probabilities for sequences that are equivalent to the original model (because when we combine operators the factors (Q^+Q) cancel out):

$$\begin{aligned}\mathbb{P}(m_{1:T}) &= \hat{\alpha}_\infty^\top\ \hat{A}_{m_T}\ \cdots\ \hat{A}_{m_1}\ \hat{\alpha}_1 \\ &= (\alpha_\infty)^\top Q^+ Q A_{m_T} Q^+ \cdots Q A_{m_1} Q^+ Q \alpha_1 \\ &= (\alpha_\infty)^\top\ A_{m_T}\ \cdots\ A_{m_1}\ \alpha_1 \end{aligned} \tag{5.15}$$

Algorithm 2 presents pseudocode for a spectral learning algorithm to learn the operators of a SHAG, using training head-modifier sequences. Note that each operator model in the SHAG is learned separately. The running time of the algorithm is dominated by two computations. First, a pass over the training sequences to compute

probability matrices over unigrams, bigrams and trigrams. Second, SVD and matrix operations to create the operators, which run in cubic time in the number of symbols l. However note that if matrices are sparse there exist more efficient methods.

5.3 EXPERIMENTS

In the experiments, our aim is to learn natural language models using the spectral algorithm, and to do both a quantitative and a qualitative evaluation of them. The quantitative evaluation is done in the next chapter using the models in parsing tasks. The qualitative evaluation is done in this chapter, by training and analyzing fully unlexicalized models, i.e., models over part-of-speech tag sequences.

The corpus we use for the experiments is the dependency version of the English WSJ Penn Treebank [37]. The dependency trees are obtained from the original constituent trees by means of a set of head-finding rules [17]. We train SHAG models using the standard WSJ training sections (2 to 21). As the models are unlexicalized, we learn one operator model for each POS tag and direction, using as input the POS modifier sequences observed for that head and direction in the training set. Table 5.1 shows some general statistics for the modifier sequences of the different POS tag and directions in the WSJ training sections.

The purpose of the qualitative analysis is to see what information is encoded in the models learned by the spectral algorithm. However, hidden state spaces are hard to interpret, and this situation is much worse if the operators are projected into a non-probabilistic space, as in our case. To do the analysis, we build DFAs that approximate the behaviour of the non-deterministic models when they generate highly probable sequences. The DFA approximations allows us to observe in a simple way some linguistically relevant phenomena encoded in the states, and to compare them with manually encoded features of other models.

The next section describes the DFA approximation construction method. Then, in the following section, we analyze two of the most important operator models for the WSJ in terms of the number of dependencies, namely, the models for (NN, LEFT) and (VBD, RIGHT). We use

5.3.1 *DFA Approximation*

Consider the space $\mathbb{R}^S$ where the forward-state vectors lie. Generating a modifier sequence corresponds to a path through the S-dimensional state space. We cluster sets of forward-state vectors in order to create a DFA approximation that we can use to visualize the phenomena captured by the state space.

Head	Dir.	# Mods.	Avg. len.	Max. len.	# Voc.	Top 5 modifiers
NN	LEFT	159869	1.2	21	44	DT JJ NN NNP CD
IN	RIGHT	100776	1.0	9	44	NN NNS NNP CD VBD
NNS	LEFT	73787	1.2	19	42	JJ NN DT CD NNP
VBD	RIGHT	65158	2.2	12	40	. IN NN VBN ,
NNP	LEFT	53066	0.6	53	38	NNP DT , CC NN
VBD	LEFT	47675	1.6	13	40	NN , NNP NNS PRP
NN	RIGHT	45127	0.3	9	38	IN , WDT VB VBN
VBZ	RIGHT	40611	1.9	15	39	. NN VBN IN RB
ROOT	LEFT	39832	1.0	1	29	VBD VBZ VBP MD NN
VB	RIGHT	38160	1.4	11	41	NN IN NNS VB VBN
VBZ	LEFT	34089	1.6	10	42	NN , NNP PRP RB
VBN	RIGHT	23258	1.2	12	39	IN NN VB RB VBN
VBP	RIGHT	23070	1.8	10	40	. VBN IN RB NN
NNS	RIGHT	19866	0.3	12	38	IN , VBN WDT VBG
VBP	LEFT	19024	1.5	12	40	NNS PRP , RB IN
VBG	RIGHT	18023	1.2	14	39	IN NN NNS VB TO
VB	LEFT	16197	0.6	9	37	TO NNS RB NN PRP
MD	RIGHT	15878	1.6	11	25	VB . RB " CC
POS	LEFT	15255	1.8	7	29	NNP NN DT NNS JJ
MD	LEFT	14762	1.5	12	38	NN PRP , NNS IN
$	RIGHT	12013	1.6	11	26	CD IN -RRB- , JJ
NNP	RIGHT	10703	0.1	12	33	, IN CD WDT WP
TO	RIGHT	8927	0.4	6	36	NN NNS CD NNP $
JJ	LEFT	6982	0.1	11	37	RB JJ CC DT RBR
CD	LEFT	6238	0.2	18	35	CD $, DT CC
RB	RIGHT	5689	0.2	9	37	IN RB CD NN TO
JJ	RIGHT	5089	0.1	9	34	IN VB TO NNP CC
IN	LEFT	4141	0.0	7	34	RB JJ JJR , IN
WDT	RIGHT	3995	0.9	4	18	VBZ VBD VBP MD NN
$	LEFT	2677	0.4	10	26	IN RB CD $ -LRB-
WP	RIGHT	2302	1.0	5	21	VBD VBZ VBP MD VB
NNPS	LEFT	2287	0.9	21	26	NNP DT JJ CC NNPS
WRB	RIGHT	2171	1.0	5	24	VBD VBZ VBP MD JJ
VBN	LEFT	2154	0.1	11	33	RB NN NNS , CC
CD	RIGHT	1995	0.1	7	32	IN , RB NN -RRB-
RB	LEFT	1620	0.1	12	31	NN NNS , RB IN
DT	RIGHT	1315	0.0	6	28	IN VBZ CD VBD WP
VBG	LEFT	1156	0.1	6	34	RB NN NNS DT NNP
JJR	LEFT	802	0.2	6	24	RB CC NN DT JJR
JJR	RIGHT	577	0.2	5	23	IN VB TO CC NN
WP$	RIGHT	448	2.7	9	25	NN NNS VBD VBZ VBP
DT	LEFT	364	0.0	12	22	CC NNS , PRP RB
...						

Table 5.1: General statistics for training modifier sequences of the WSJ dependency corpus. Sorted by total number of modifiers.

To build a DFA, we compute the forward vectors corresponding to frequent prefixes of modifier sequences of the development set. Then, we cluster these vectors using a Group Average Agglomerative algorithm using the cosine similarity measure [36]. This similarity measure is appropriate because it compares the angle between vectors, and is not affected by their magnitude (the magnitude of forward-vectors decreases with the number of modifiers generated). Each cluster i defines a state in the DFA, and we say that a sequence $m_{1:t}$ is in state i if its corresponding forward vector at time t is in cluster i.

The transitions in the DFA are defined using a procedure that looks at how sequences traverse the states. If a sequence $m_{1:t}$ is at state i at time $t-1$, and goes to state j at time t, then we define a transition from state i to state j with label m_t. This procedure may require merging states to give a consistent DFA, because different sequences may define different transitions for the same states and modifiers. After doing a merge, new merges may be required, so the procedure must be repeated until a DFA is obtained.

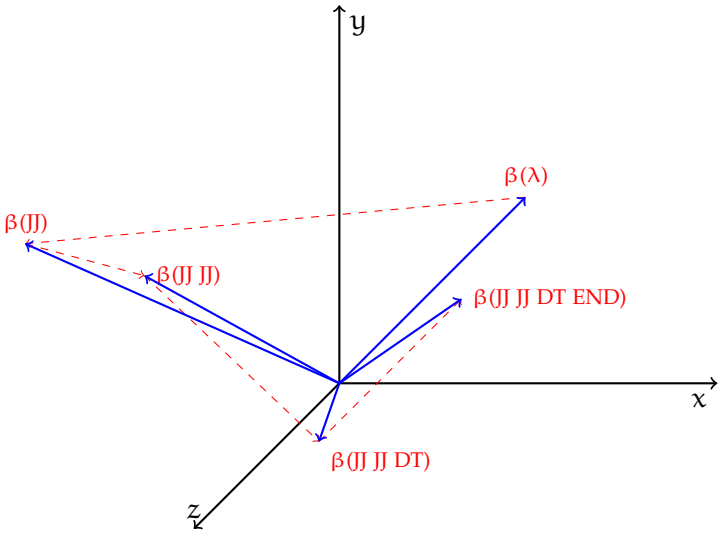
Fig. 5.1 illustrates the DFA construction process showing fictitious forward vectors in a 3 dimensional space. The forward vectors correspond to the prefixes of the sequence "JJ JJ DT END", a frequent sequence of noun left modifiers (NN, LEFT). In this example, we construct a 3 state automata by clustering the vectors into three different sets and then defining the transitions as described in the previous paragraphs.

5.3.2 *Results Analysis*

A DFA approximation for the automaton (NN, LEFT) is shown in Fig. 5.2. The vectors were originally divided in ten clusters, but the DFA construction required four merging of states, leading to a six state automaton. State number 5 is the initial and final state. Clearly, we can see that there are special states for punctuation (state 3) and coordination (states 1 and 2). States 4 and 6 are harder to interpret. To understand them better, we computed an estimation of the probablities of the transitions, by counting the number of times each of them is used. Now, we find that our estimation of generating END from state 4 is 0.78, and from state 6 it is 0.18. Interestingly, state 6 can transition to state 4 generating PRP$, POS or DT, that are usual endings of modifier sequences for nouns (recall that modifiers are generated head-outwards, so for a left automaton the final modifier is the left-most modifier in the sentence).

A DFA approximation for (VBD, RIGHT) is shown in Fig. 5.3. The vectors were divided in ten clusters and the DFA construction did not require any merging. State number 9 is the initial and final state. At most one adverb (RB) or particle (RP) is generated, always as the

a)



b)

$$c_0 = \{\beta(\lambda), \beta(\text{JJ JJ DT END})\}$$
$$c_1 = \{\beta(\text{JJ}), \beta(\text{JJ JJ})\}$$
$$c_2 = \{\beta(\text{JJ JJ DT})\}$$

c)

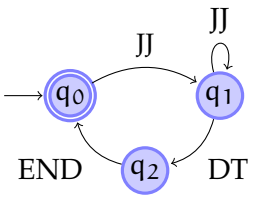


Figure 5.1: Example of construction of a 3 state DFA approximation. a) Forward vectors β for the prefixes of the sequence "JJ JJ DT END". b) Cosine similarity clustering. c) Resulting DFA after adding the transitions.

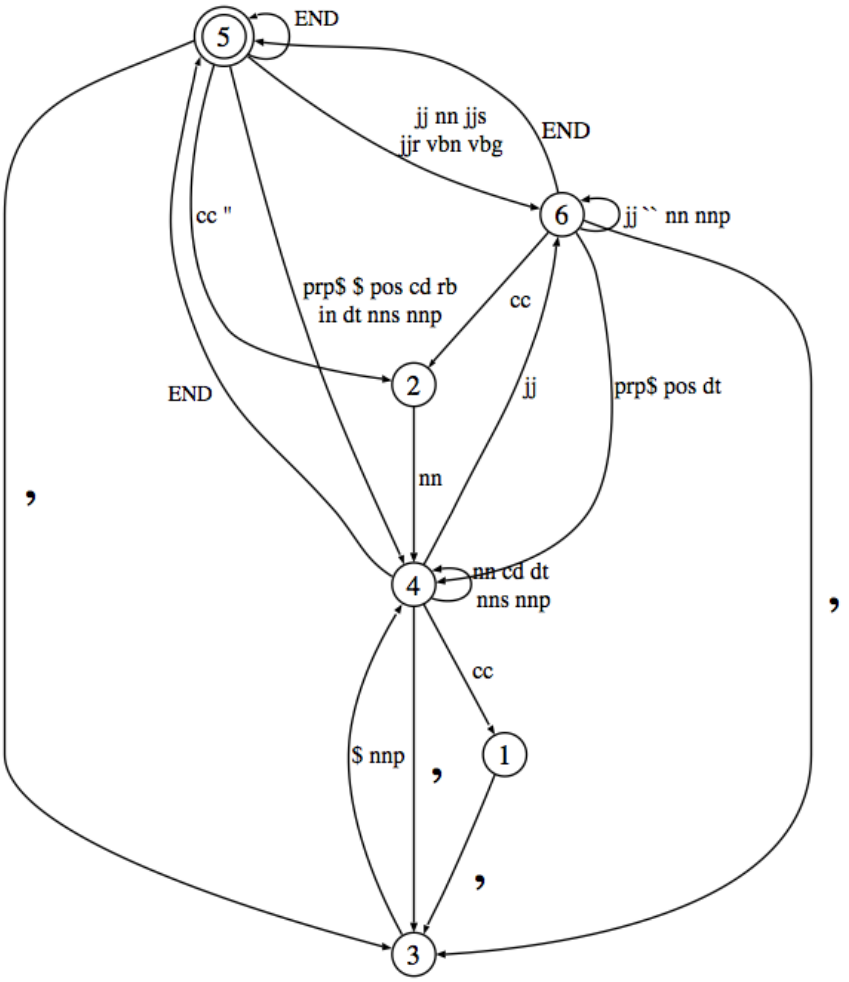


Figure 5.2: DFA approximation for the generation of left modifiers for NN.

first modifier. Then the different type of objects can be generated, in states 1, 3 and 6. Only after that, in state 8, an optional sequence of prepositions is generated, that are the heads of the prepositional phrases. After this, the generation ends, or is followed by punctuation or coordination, going to states 5 or 2, respectively. Punctuation is always followed by a gerund verb, `VBG`, and coordination is always followed by a past tense verb, as the head itself, going in both cases to state 4, that also marks the end of the generation process. These are all linguistically reasonable behaviours for the generation of right modifiers for `VBD`'s. Nevertheless, it is reasonable to expect some "noise" in the automaton, because of the wide variety of verbs that are being mixed here. Some "noise" can also be attributed to the particularities of the WSJ corpus (such as state number 1).

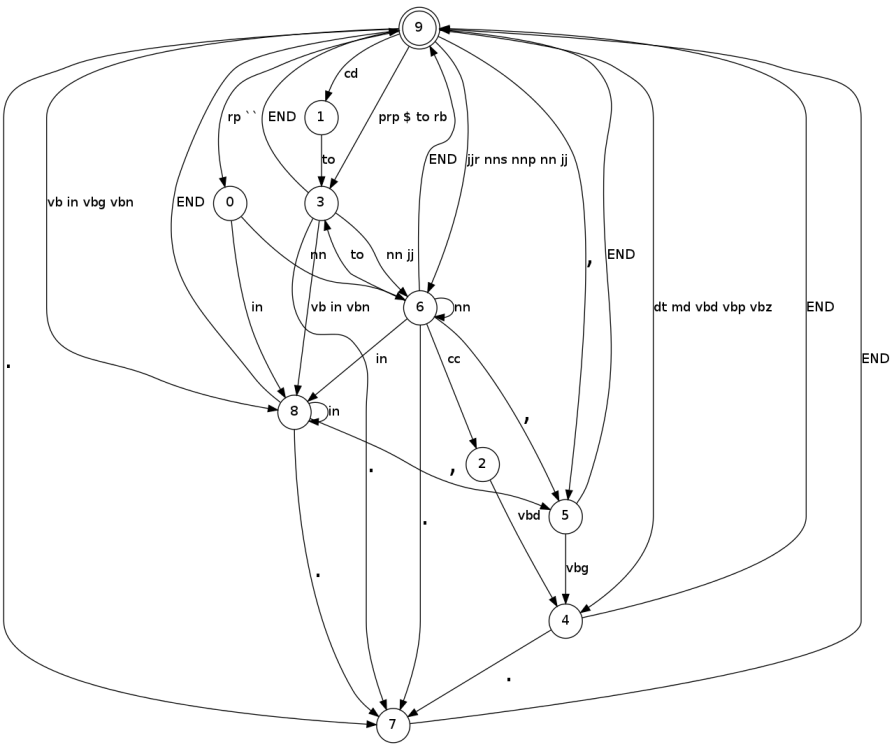


Figure 5.3: DFA approximation for the generation of right modifiers for VBD.

6

NON-DETERMINISTIC SPLIT HEAD AUTOMATA

In this chapter we present a dependency parsing model that exploits hidden structure using probabilistic non-deterministic automata. Crucially, the model can be trained with a spectral learning algorithm that is both efficient and not susceptible to local-minima. We also present an inside-outside algorithm for our parsing model that runs in cubic time, hence maintaining the standard parsing costs. In experiments, we show that adding hidden-structure to a variety of baseline models results in $\sim 30\%$ error reductions.

6.1 PARSING ALGORITHMS

Given a sentence $s_{0:N}$ we would like to find its most likely derivation, $\hat{y} = \text{argmax}_{y \in \mathcal{Y}(s_{0:N})} \mathbb{P}(y)$. This problem, known as MAP inference, is known to be intractable for hidden-state structure prediction models, as it involves finding the most likely tree structure while summing out over hidden states. We use a common approximation to MAP based on first computing posterior marginals of tree edges (i.e. dependencies) and then maximizing over the tree structure (see [43] for complexity of general MAP inference and approximations). For parsing, this strategy is sometimes known as MBR decoding; previous work has shown that empirically it gives good performance [26, 16, 54, 45]. In our case, we use the non-deterministic SHAG to compute posterior marginals of dependencies. We first explain the general strategy of MBR decoding, and then present an algorithm to compute marginals.

Let (s_i, s_j) denote a dependency between head word i and modifier word j. The posterior or *marginal probability* of a dependency (s_i, s_j) given a sentence $s_{0:N}$ is defined as

$$\mu_{i,j} = \mathbb{P}((s_i, s_j) \mid s_{0:N}) = \sum_{y \in \mathcal{Y}(s_{0:N}) \,:\, (s_i, s_j) \in y} \mathbb{P}(y) \quad .$$

To compute marginals, the sum over derivations can be decomposed into a product of inside and outside quantities [6]. Below we describe

an inside-outside algorithm for our grammars. Given a sentence $s_{0:N}$ and marginal scores $\mu_{i,j}$, we compute the parse tree for $s_{0:N}$ as

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}(s_{0:N})} \sum_{(s_i, s_j) \in y} \log \mu_{i,j} \tag{6.1}$$

using the standard projective parsing algorithm for arc-factored models [21]. Overall we use a two-pass parsing process, first to compute marginals and then to compute the best tree.

6.1.1 *An Inside-Outside Algorithm*

In this section we sketch an algorithm to compute marginal probabilities of dependencies, given a sentence $x_{0:n}$. Our algorithm is an adaptation of the parsing algorithm for SHAG by [22] to the case of non-deterministic head-automata, and has a runtime cost of $O(S^2 n^3)$, hence maintaining the standard cubic dependency on the sentence length. The quadratic dependency on S is inherent to the computations defined by our model (Eq.5.3). The main insight behind the design of this extension is the following: because the computations of our model involve state-distribution vectors, we need to extend the standard inside/outside quantities to be in the form of state-distribution vectors.[1]

Throughout this section we assume a fixed sentence $x_{0:n}$. We abuse notation and use $(x_i, x_j) \in y$ to indicate that x_i is head of x_j in a derivation y. We use $\text{root}(y)$ to indicate the root of a derivation. Finally, we use $\mathcal{Y}(x_{i:j})$ as the set of derivations that yield the partial sentence $x_{i:j}$. Following [22], we use decoding structures related to complete half-constituents (or "triangles", denoted C) and incomplete half-constituents (or "trapezoids", denoted I), each with a direction (denoted L and R). We assume familiarity with their algorithm.

We define $\theta^{I,R}_{i,j} \in \mathbb{R}^S$ as the inside score-vector of a right trapezoid dominated by dependency (x_i, x_j),

$$\theta^{I,R}_{i,j} = \sum_{\substack{y \in \mathcal{Y}(x_{i:j}) \,:\, (x_i, x_j) \in y \,, \\ y = \{\langle x_i, R, m_{1:t} \rangle\} \cup y' \,,\, m_t = x_j}} \mathbb{P}(y') \alpha^{x_i, R}(m_{1:t}) \quad .$$

The term $\mathbb{P}(y')$ accounts for probabilities of head-modifier sequences in the range $x_{i:j}$ that do not involve x_i. The term $\alpha^{x_i,R}(m_{1:t})$ is a *forward* state-distribution vector —the s-th coordinate of the vector is the probability that x_i generates right modifiers $m_{1:t}$ and remains at

1 We should note that, technically, when working with the projected operators that we learn ($\hat{A}_x = Q A_x Q^+$) the state-distribution vectors will not be distributions in the formal sense. However, they correspond to a projection of a true state distribution (the distributions given by A_x), for some projection Q we can not recover from data. This projection has no other effect on the computation of probabilities.

state s. Similarly, we define $\phi^{I,R}_{i,j} \in \mathbb{R}^S$ as the outside score-vector of a right trapezoid, as

$$\phi^{I,R}_{i,j} = \sum_{\substack{y \in \mathcal{Y}(x_{0:i} x_{j:n}) \,:\, \text{root}(y)=x_0, \\ y=\{\langle x_i,R, m_{t:T}\rangle\} \cup y' \,,\, m_t=x_j}} \mathbb{P}(y') \beta^{x_i,R}(m_{t+1:T}) \;\;,$$

where $\beta^{x_i,R}(m_{t+1:T})$ is a *backward* state-distribution vector —the s-th coordinate is the probability of being at state s of the right automaton of x_i and generating $m_{t+1:T}$. Analogous inside-outside expressions can be defined for the rest of structures (left/right triangles and trapezoids). With these quantities, we can compute marginals as

$$\mu_{i,j} = \begin{cases} \phi^{I,R}_{i,j} \; \theta^{I,R}_{i,j} \; Z^{-1} & \text{if } i < j \,, \\ \phi^{I,L}_{i,j} \; \theta^{I,L}_{i,j} \; Z^{-1} & \text{if } j < r \,, \end{cases}$$

where Z is the partition function:

$$Z = \sum_{y \in \mathcal{Y}(x_{0:n})} \mathbb{P}(y) = (\alpha^{\star,R}_\infty)^\top \; A^{\star,R}_{\text{STOP}} \; \theta^{C,R}_{0,n}$$

Finally, we sketch the equations for computing inside scores of right half-constituents in $O(n^3)$ time . The equations for inside scores of left half-constituents are symmetrical. The outside equations can be derived analogously (see [44]). For $0 \leqslant i < j \leqslant n$:

$$\theta^{C,R}_{i,i} = \alpha^{x_i,R}_1 \tag{6.2}$$

$$\theta^{C,R}_{i,j} = \sum_{k=i+1}^{j} \theta^{I,R}_{i,k} \left((\alpha^{x_k,R}_\infty)^\top \; A^{x_k,R}_{\text{STOP}} \; \theta^{C,R}_{k,j} \right) \tag{6.3}$$

$$\theta^{I,R}_{i,j} = \sum_{k=i}^{j} A^{x_i,R}_{x_j} \theta^{C,R}_{i,k} \left((\alpha^{x_j,L}_\infty)^\top \; A^{x_j,L}_{\text{STOP}} \; \theta^{C,L}_{k+1,j} \right) \tag{6.4}$$

Fig. 6.1 illustrates these equations. Fig. 6.1.a corresponds to the basic case of Eq. 6.2, and Figs. 6.1.b and 6.1.c correspond respectively to Eqs. 6.3 and 6.4 with a fixed k.

6.2 EXPERIMENTS

The goal of our experiments is to show that incorporating hidden states in a SHAG using operator models can consitently improve parsing accuracy. A second goal is to compare the spectral learning algorithm to EM, a standard learning method that also induces hidden states.

The first set of experiments involve fully unlexicalized models, i.e. parsing part-of-speech tag sequences. While this setting falls behind the state-of-the-art, it is nonetheless valid to analyze empirically the effect of incorporating hidden states via operator models, which results in large improvements. In a second set of experiments, we combine the unlexicalized hidden-state models with simple lexicalized models.

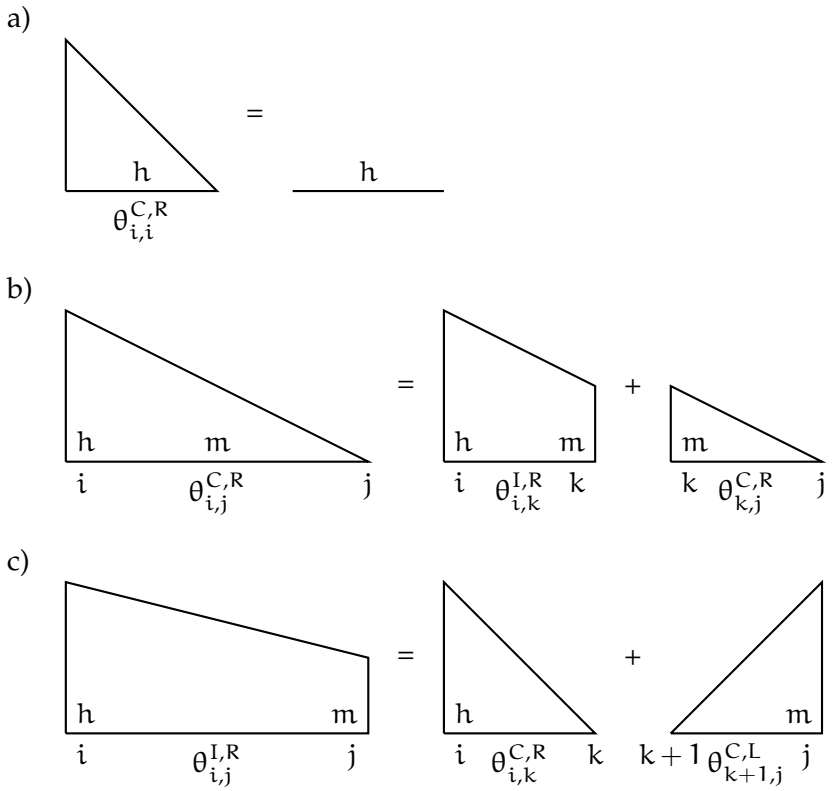


Figure 6.1: Graphical depiction of the inside scores computations for the different types of chart elements for right half-constituent. Computations for left half-constituents are symmetrical. a) Empty right half-constituent ("triangle"). b) Non-empty complete right half-constituent ("triangle"). c) Incomplete right half-constituent ("trapezoid").

6.2.1 *Fully Unlexicalized Grammars*

We trained fully unlexicalized dependency grammars from dependency treebanks, that is, $\mathcal{X}$ are POS tags and we parse POS tag sequences. In all cases, our modifier sequences include a special STOP word at the end. We compare the following SHAG models:

- DET : a basic deterministic grammar with a single state.
- DET+F : a deterministic grammar with two states, one emitting the first modifier of a sequence, and another emitting the rest (see [23] for a similar deterministic baseline).
- SPECTRAL: a non-deterministic grammar with S hidden states trained with the spectral algorithm. S is a parameter of the model.
- SPECTRAL+F: a non-deterministic grammar with S hidden states that uses different operators to generate the first modifier and

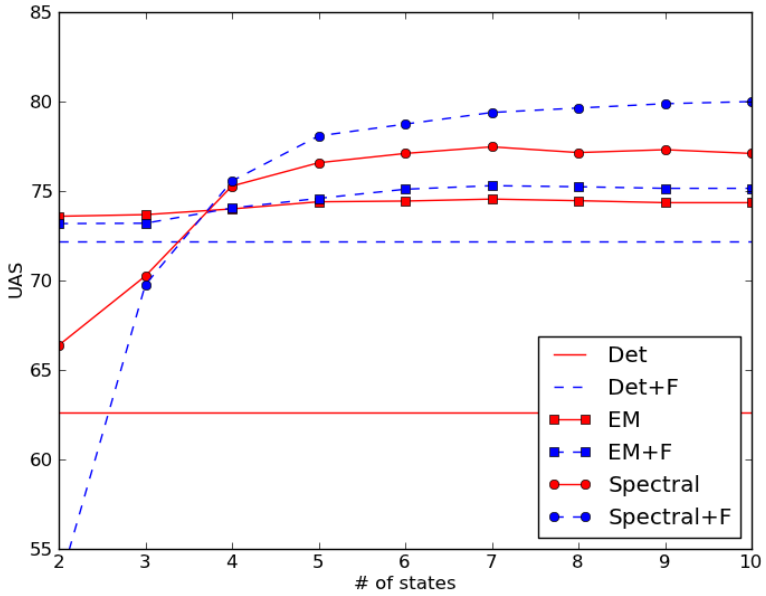


Figure 6.2: Accuracy curve on English development set for fully unlexicalized models.

the rest. We obtain these by augmenting the first modifier of each sequence with a special +FIRST tag, thus doubling $\mathcal{X}$ and the number of operators.

- EM: a non-deterministic grammar with S states trained with EM. Here, $\alpha_\infty = \vec{1}_{1:S}$, and we estimate operators $A_x^{h,d}$, α_1 using forward-backward for the E step. To initialize, we mimicked an HMM initalization: (1) we set α_1 randomly; (2) we created a random transition matrix $T \in \mathbb{R}^{S\times S}$; (3) we created a diagonal matrix $O_x^{h,d} \in \mathbb{R}^{S\times S}$, where $O_x^{h,d}(i,i)$ is the probability of generating symbol x from h and d (estimated from training, plus a random variation); (4) we set $A_x^{h,d} = TO_x^{h,d}$.
- EM+F: a non-deterministic grammar with S states trained with EM, that has different distributions to emit the first modifier and the rest.

Using standard WSJ sections of the English Penn Treebank [37], we trained grammars and compared their performance. Fig. 6.2 shows the Unlabeled Attachment Score (UAS, see section 2.5.2) curve on the development set, in terms of the number of hidden states for the spectral and EM models (for EM, we show the best run on development out of 50 runs). We can see that DET+F largely outperforms DET, but the interesting part is that hidden-state models obtain larger improvements even when the FIRST state is not explicitly encoded. As for the hidden-state models, the spectral method obtains better accuracies than EM.

	Det	Spectral	Det+F	Spectral+F
WSJ	63.76	78.25	72.93	80.83
Dan	61.70	77.25	69.70	78.16
Dut	54.30	61.32	60.90	64.01
Por	71.39	85.33	81.47	85.71
Slo	60.31	66.71	65.63	68.65
Swe	69.09	79.55	76.94	80.54
Tur	53.79	62.56	57.56	63.04

Table 6.1: UAS of fully unlexicalized models on test sets for several languages.

More important, however, is to compare training times. The spectral method took about 30 seconds to train in a 2.93 Ghz Intel Core i7-870 microprocessor. Most of the time was consumed by the computation of input statistics for the SVD algorithm, implemented in the Python language, while the SVD itself was done with an external tool that ran almost instantly. In contrast, the EM method required at least 50 iterations to reach a stable accuracy on development, where *each iteration* took from 2 to 3 minutes, with a Matlab implementation running on the mentioned microprocessor. So, there is a factor of at least 200 between the training times of the spectral method and a convergent EM.

Table 6.1 shows results on WSJ test data, plus on the tests for six languages from the CoNLL-X shared task [9], in comparison with the deterministic baselines. We always use the number of states that gave optimal results for the WSJ development set (7 states for SPECTRAL, 10 for SPECTRAL+F). The spectral models consistently outperform the deterministic models by a large margin. As for the result on the WSJ test, our deterministic baselines obtain roughly the same performance as those of [23], while the best result we obtain using hidden-states (80.8%) largely improves over their best model that uses information about the length of dependencies (75.6%)

6.2.2 *Experiments with Lexicalized Grammars*

We now turn into estimating lexicalized deterministic grammars and combine them with unlexicalized spectral grammars we obtained in the previous experiment. The goal behind this experiment is to show that the information captured in hidden states is complimentary to head-modifier lexical preferences.

In this case $\mathcal{X}$ consists of lexical items, and we assume access to the POS tag of each lexical item. We will denote as t_x and w_x the POS tag and word of a symbol $x \in \mathcal{X}$. We will estimate conditional

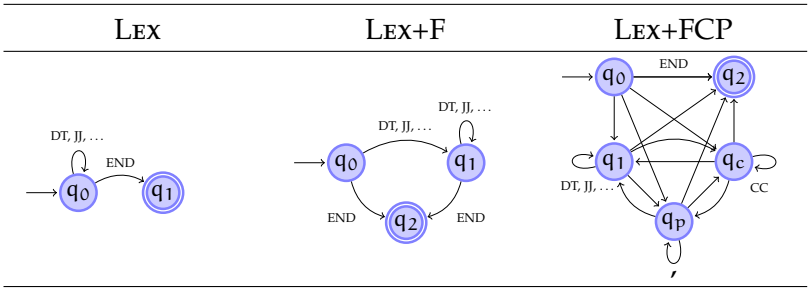


Figure 6.3: Unlexicalized DFAs illustrating the features encoded in the three deterministic baselines LEX, LEX+F and LEX+FCP. In LEX+FCP, q_c and q_p are, respectively, the target states for coordination and punctuation symbols. For clarity, on each automata we added a separate final state, and a special ending symbol END.

distributions $\mathbb{P}(m \mid h, d, \sigma)$, where $m \in \mathcal{X}$ is a modifier, $h \in \bar{\mathcal{X}}$ is a head, d is a direction, and σ is a deterministic state. Following [17], we use three configurations of deterministic states:

- LEX: a single state.
- LEX+F: two distinct states for first modifier and rest of modifiers.
- LEX+FCP: four distinct states, encoding: first modifier, previous modifier was punctuation, previous modifier was a coordination, and previous modifier was some other word.

The unlexicalized DFAs shown in Fig. 6.3 illustrate in a simple way the different configurations. However, the actual distributions use a back-off strategy that combine lexicalized and unlexicalized information. The back-off strategy factorizes $\mathbb{P}$ as follows:

$$\mathbb{P}(m|h, d, \sigma) = \mathbb{P}_A(t_m|h, d, \sigma)\mathbb{P}_B(w_m|t_m, h, d, \sigma)$$

To estimate $\mathbb{P}_A$ we use two back-off levels, the fine level conditions on $\{h, d, \sigma\}$ and the coarse level conditions on $\{t_h, d, \sigma\}$. For $\mathbb{P}_B$ we use three levels, which from fine to coarse are $\{t_m, h, d, \sigma\}$, $\{t_m, t_h, d, \sigma\}$ and $\{t_m\}$. We use the strategy of [17] to estimate $\mathbb{P}_A$ and $\mathbb{P}_B$ from a treebank using back-off.

We use a simple approach to combine lexical models with the unlexical hidden-state models we obtained in the previous experiment. Namely, we use a log-linear model that computes scores for head-modifier sequences as

$$s(\langle h, d, m_{1:T}\rangle) = \log \mathbb{P}_{sp}(m_{1:T}|h, d) + \log \mathbb{P}_{det}(m_{1:T}|h, d),$$

where $\mathbb{P}_{sp}$ and $\mathbb{P}_{det}$ are respectively spectral and deterministic probabilistic models. We tested combinations of each deterministic model with the spectral unlexicalized model using different number of states. Fig. 6.4 shows the accuracies of single deterministic models, together

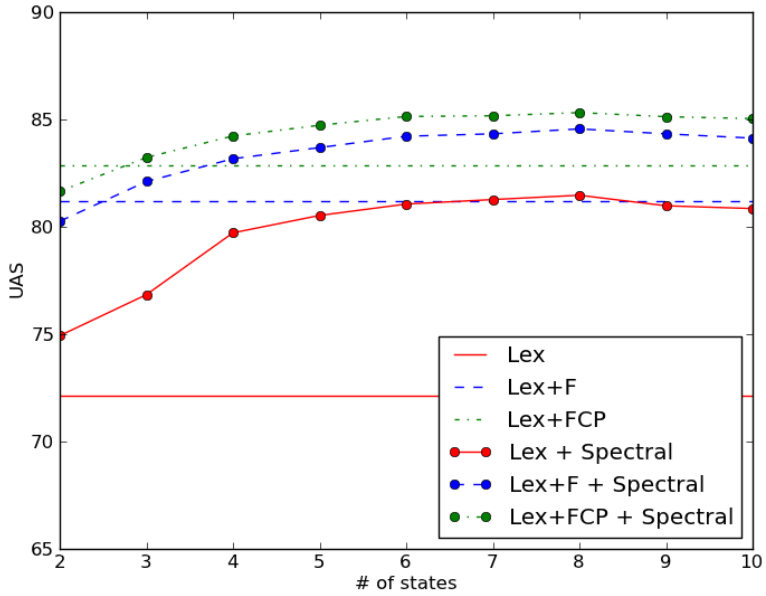


Figure 6.4: Accuracy curve on English development set for lexicalized models.

with combinations using different number of states. In all cases, the combinations largely improve over the purely deterministic lexical counterparts, suggesting that the information encoded in hidden states is complementary to lexical preferences.

Head	Dir.	Weight	LEX	+ SPECT.	LEX+F	+ SPECT.	LEX+FCP	+ SPECT.
NN	LEFT	18.7	79.8	85.5	82.5	86.6	83.0	87.4
IN	RIGHT	11.5	52.8	77.2	84.6	86.0	86.2	87.3
NNS	LEFT	8.7	79.3	85.1	80.4	85.1	82.0	85.3
VBD	RIGHT	7.0	75.2	83.3	81.7	86.2	83.7	87.4
VBD	LEFT	6.3	82.4	88.1	88.1	90.9	89.3	89.6
NNP	LEFT	5.9	62.6	71.4	67.9	74.7	71.9	77.7
ROOT	LEFT	4.8	82.9	86.5	88.2	88.2	90.0	89.6
NN	RIGHT	4.5	81.6	81.1	81.3	81.1	82.2	81.8
VB	RIGHT	3.9	68.5	80.4	78.1	85.8	82.7	85.8
VBZ	RIGHT	2.4	74.5	80.8	82.6	85.6	84.8	85.6
VBN	RIGHT	2.4	75.0	87.6	83.8	89.3	84.4	89.1
VBZ	LEFT	2.3	78.8	88.1	90.2	90.7	90.2	91.0
POS	LEFT	2.1	37.0	73.8	63.0	78.1	62.8	77.9
NNS	RIGHT	2.0	76.7	78.8	79.8	79.5	81.2	79.5
VBG	RIGHT	1.8	63.5	80.6	74.9	83.1	78.1	83.5
VB	LEFT	1.7	88.6	88.1	91.8	91.7	92.8	93.2
$	RIGHT	1.5	64.6	91.1	92.5	92.7	93.0	93.6
VBP	RIGHT	1.2	77.0	81.1	85.4	86.3	85.4	86.3
MD	LEFT	1.2	75.6	83.1	83.6	85.4	85.4	86.5
VBP	LEFT	1.2	80.3	87.3	86.6	90.6	89.0	88.7
MD	RIGHT	1.2	91.1	92.8	94.5	95.2	97.1	96.9
TO	RIGHT	1.0	34.6	70.9	64.0	78.4	67.9	79.8
JJ	LEFT	0.8	55.3	54.3	64.9	66.0	70.9	70.2
RB	RIGHT	0.8	63.9	57.2	78.4	74.0	79.2	73.2
CD	LEFT	0.8	58.0	68.0	57.6	66.2	71.0	67.7
NNP	RIGHT	0.7	78.8	75.4	76.7	78.3	79.2	78.3
JJ	RIGHT	0.5	63.1	54.5	64.2	51.9	65.8	54.5
WDT	RIGHT	0.5	75.1	89.6	91.9	93.6	91.3	93.1
IN	LEFT	0.4	66.0	58.2	68.8	62.4	68.8	60.3
VBN	LEFT	0.3	54.5	58.7	63.6	66.1	65.3	64.5
$	LEFT	0.3	58.6	68.5	62.2	70.3	63.1	73.0
WRB	RIGHT	0.3	70.0	87.8	90.0	92.2	90.0	91.1
WP	RIGHT	0.2	74.0	84.4	93.5	90.9	93.5	89.6
CD	RIGHT	0.2	45.8	44.1	45.8	40.7	45.8	42.4
DT	RIGHT	0.1	60.4	56.6	67.9	54.7	73.6	56.6
RB	LEFT	0.1	66.0	66.0	69.8	64.2	71.7	64.2
VBG	LEFT	0.1	37.5	39.6	47.9	43.8	56.2	56.2
JJR	LEFT	0.1	88.5	76.9	92.3	76.9	76.9	76.9
RBR	LEFT	0.1	88.0	96.0	100.0	96.0	92.0	96.0
WP$	RIGHT	0.1	50.0	72.7	68.2	72.7	63.6	77.3
DT	LEFT	0.1	15.8	0.0	36.8	21.1	63.2	42.1

Table 6.2: Results for lexicalized models by head and direction, ordered by influence in the global result. Rows with weight $< 0.1\%$ were discarded.

Part IV

CONCLUSION

7

CONCLUSION

In this chapter we discuss the conclusions and possible future work of this thesis. We first address separately the conclusions of the specific problems studied on each part of the thesis. At the end, we expose some general conclusions about the fields we studied.

7.1 ON NON-TERMINALLY SEPARATED GRAMMARS

On the first part of this thesis, we proposed the goal of finding PAC-learnable classes of formal languages that are strongly adequate for natural language, having in mind the idea of using them for unsupervised parsing.

To do this, we first defined UWNTS grammars and show that they are polynomially PAC-learnable. Then, we gave an experimental method to study the expressivity of UWNTS grammars over concrete natural language corpora. We applied this method to a corpus of short English sentences of POS tags. We found that, regardless of the learning algorithm, UWNTS grammars will have a low performance compared to state-of-the-art unsupervised parsers. We believe that these experiments provide enough evidence to conclude that unsupervised parsing over POS tags requires more expressive grammars.

Then, we defined the hierarchy of k, l-UWNTS$^{\leqslant}$ grammars and proved that they also are polynomially PAC-learnable. We did the same experiments as for UWNTS to measure the strong adequacy of k, l-UWNTS$^{\leqslant}$ grammars for low k, l. We saw that these grammars are much more capable of correctly parsing short English sentences of POS tags. In principle, this fact may suggest that it is worth to give a try to the PAC-learning algorithm. However, one should take a look also at the sample complexity of the algorithm. Even for small toy natural language grammars, learning them with precision and confidence of at least 80% would require millions of positive examples.

As PAC-learnability is a theoretical property, it is not very surprising that the learning algorithm is not directly applicable. A feasible approach would be to develop a more practical algorithm, combining elements from the PAC algorithm with heuristics and ML techniques.

This is what Clark did to learn NTS grammars for the Omphalos competition [14].

7.2 ON NON-DETERMINISTIC DEPENDENCY PARSING

On the second part of this thesis, our aim was to learn latent variable models using a spectral algorithm, in the frame of dependency parsing. We expected this algorithm to learn meaningful latent variables, and useful parsing models. We also expected this algorithm to learn better models than EM applied to the same models. To do this, we defined a non-deterministic version of Split-Head Automata Grammars, a spectral learning algorithm and a cubic time parsing algorithm.

In unlexicalized experiments with the English language, we found by a qualitative analysis that the algorithm is finding linguistically meaningful features.

We also found in these experiments that spectral learning outperforms deterministic models and the EM algorithm. Even though we didn't do further experiments with EM, we consider that there is enough evidence to conclude that, at least for this task, the spectral algorithm is better in all aspects. Better results for EM, if possible, would require several long time runs with different initializations, while with the spectral algorithm, training can be easily and quickly done.

Then, we did unlexicalized experiments with other six languages, always finding that our model perform better that deterministic baselines.

Finally, we defined several lexicalized deterministic baselines using simple feature engineering. In the experiments, we saw a consistent improvement of the baseline models when adding the unlexicalized spectral model. In this situation, we see how feature engineering can be complemented with latent variable models, to obtain better parsers with no need of additional efforts.

In the future, our methods may be used to enrich the representational power of more sophisticated dependency models. For example, future work should consider enhancing lexicalized dependency grammars with hidden states that summarize lexical dependencies. Another line for future research should extend the learning algorithm to be able to capture vertical hidden relations in the dependency tree, in addition to sequential relations.

7.3 GENERAL CONCLUSION

Despite being an old and widely studied subject, the parsing problem is far from being considered solved. Ceirtanly, some highly tuned supervised parsers achieved excellent performances over big corpora

such as the English Penn Treebank. It is now known that language specific parsing can be satisfactorily solved with good linguistic resources and an adequate feature engineering. But this methodology is clearly expensive in money and time, and it obviously requires the availability of experts in the language in question. In the last years, the goal of research has turned more into high quality parsing models that are language independent and do not require big annotated corpora [9]. This is where semi-supervised and unsupervised methods play a central role.

The fully unsupervised parsing problem is even much further from being solved, if this is ever possible. Unsupervised parsing is a relatively immature subject and is more of a theoretical nature. Some encouraging results has been given using heuristics and machine learning techniques [7, 32, 49]. The Grammatical Inference field provides a rigorous tool to study this problem, but it has failed so far to provide any result barely appliable to natural language. It still needs the formulation of more realistic learning settings.

Part V

APPENDIX

A

PROOFS OF CHAPTER 3

A.1 LEMMA 1

Lemma. *If* L *is WNTS, then* $xLx = \{xsx | s \in L\}$ *is NTS, where* x *is a new element of the alphabet.*

Proof. Let G be an WNTS grammar such that $L(G) = L$. Now, let G' be the same as G but replacing the rules of the form $S \to \alpha$ with $S \to x\alpha x$. It is easy to see that $L(G') = xLx$. We will now see that G' is NTS. Suppose that $X, Y, \alpha', \beta', \gamma'$ are such that $X \overset{*}{\Rightarrow}_{G'} \alpha'\beta'\gamma'$ and $Y \overset{*}{\Rightarrow}_{G'} \beta'$. We must prove that $X \overset{*}{\Rightarrow}_{G'} \alpha' Y \gamma'$. If $Y = S$, then $\beta' = S$ or $\beta' = x\beta x$ for some β. In both cases, it is easy to see that X must also be S and that $\alpha' = \gamma' = \lambda$. Then, $X \overset{*}{\Rightarrow}_{G'} \alpha' Y \gamma'$ is just $S \overset{*}{\Rightarrow}_{G'} S$ that is obvious.

If $Y \neq S$, then $Y \overset{*}{\Rightarrow}_{G'} \beta'$ implies that β' has no x and $Y \overset{*}{\Rightarrow}_{G} \beta'$. If $X = S$, then $\alpha' = x\alpha$ and $\gamma' = \gamma x$ for some α, γ. So, $X \overset{*}{\Rightarrow}_{G'} \alpha'\beta'\gamma'$ implies $X \overset{*}{\Rightarrow}_{G} \alpha\beta'\gamma$, and as G is NTS, $X \overset{*}{\Rightarrow}_{G} \alpha Y \gamma$, that going back to G' is $X \overset{*}{\Rightarrow}_{G'} \alpha' Y \gamma'$, so we are done in this case. If $X \neq S$, there is no x in α', γ', so $X \overset{*}{\Rightarrow}_{G} \alpha'\beta'\gamma'$, and as G is NTS, $X \overset{*}{\Rightarrow}_{G} \alpha' Y \gamma'$, and also in G': $X \overset{*}{\Rightarrow}_{G'} \alpha' Y \gamma'$. □

A.2 THEOREM 1

Theorem. $\mathscr{C}(S) = \{C : C \subseteq \overline{\mathrm{Sub}}(S), C \text{ compatible in } S\}$.

Proof sketch. The proof of $\subseteq$ follows immediately from the given properties.

$\supseteq$ is proved by constructing a UWNTS grammar G mainly using the fact that S is finite and C is compatible. In G, we define an initial non-terminal S and for each $s \in C$ one non-terminal X_s. The rules as are follows:

1. For each $s \in C$ that does not have any proper substring in C, there is a rule $X_s \to s$.

2. For each $s \in C$ that has at least one proper substring in C, we must decompose it in the form $s = u_1 r_1 u_2 r_2 \dots u_n r_n u_{n+1}$ where every $r_i \in C$ and in C there is no superstring of r_i that

is substring of s. This form is unique, and defines the rule $X_s \to u_1 X_{r_1} u_2 X_{r_2} \dots u_n X_{r_n} u_{n+1}$.

3. For each $s \in S \cap C$ there is a rule $S \to X_s$.

4. For each $s \in S - C$ that does not have any proper substring in C, there is a rule $S \to s$.

5. For each $s \in S - C$ that has at least one proper substring in C, we decompose it again in the form $s = u_1 r_1 u_2 r_2 \dots u_n r_n u_{n+1}$ and define the rule $S \to u_1 X_{r_1} u_2 X_{r_2} \dots u_n X_{r_n} u_{n+1}$.

With this set of rules, it can be proved that G is UWNTS, $L(G) = S$ and $Const^w(G) \cap \overline{Sub}(S) = C$. □

B

PROOFS OF CHAPTER 4

B.1 LEMMA 2

Lemma. *0,0-NTS$^{\leqslant}$ $\subset$ WNTS $\subset$ 1,1-NTS$^{\leqslant}$.*

Proof. 0,0-NTS$^{\leqslant}$ $\subset$ WNTS is trivial, given that 0,0-NTS$^{\leqslant}$ = NTS and NTS $\subset$ WNTS. To prove WNTS $\subset$ 1,1-NTS$^{\leqslant}$, we first prove WNTS $\subseteq$ 1,1-NTS$^{\leqslant}$, and then we prove WNTS $\neq$ 1,1-NTS$^{\leqslant}$.

WNTS $\subseteq$ 1,1-NTS$^{\leqslant}$: Let G be a WNTS grammar. We will see that G is 1,1-NTS$^{\leqslant}$, this is, that $\bullet G\bullet$ is 1,1-NTS. Suppose that X, Y, u, v, α, β, γ, α', γ' are such that, in $\bullet G\bullet$,

1. $X \overset{*}{\Rightarrow} \alpha u \beta v \gamma$,
2. $Y \overset{*}{\Rightarrow} \beta$
3. and $S' \overset{*}{\Rightarrow} \alpha' u Y v \gamma'$.

We must prove that

4. $X \overset{*}{\Rightarrow} \alpha u Y v \gamma$ in $\bullet G\bullet$.

First, beacuse of 3, $Y \neq S'$.

- If $Y = S$, because of 3, $u = v = \bullet$. Then, because of 1, $X = S'$ and $\alpha = \gamma = \lambda$, given that the only rule that can produce $\bullet$ is $S' \to \bullet S\bullet$. So, 4 becomes $S' \overset{*}{\Rightarrow} \bullet S\bullet$, that is trivially true.
- If $Y \neq S$ and $X = S'$, because of 1, $\alpha u = \bullet\alpha''$ and $v\gamma = \gamma''\bullet$ for some α'', γ''. Also, in 1 the first rule used must be $S' \overset{*}{\Rightarrow} \bullet S\bullet$, so we have $X = S' \overset{*}{\Rightarrow} \bullet S\bullet \overset{*}{\Rightarrow} \bullet\alpha''\beta\gamma''\bullet$. Then, in G we have $S \overset{*}{\Rightarrow} \alpha''\beta\gamma''$. This, together with 2 and $Y \neq S$, implies that $S \overset{*}{\Rightarrow}_G \alpha''Y\gamma''$, because G is WNTS by hypothesis. Then, we get to 4 as follows:

$$X = S' \overset{*}{\Rightarrow} \bullet S\bullet \overset{*}{\Rightarrow} \bullet\alpha''Y\gamma''\bullet = \alpha u Y v \gamma.$$

- If $Y \neq S$ and $X \neq S'$, we have that 1 and 2 also hold for G. So, because G is WNTS by hypothesis, we have $X \overset{*}{\Rightarrow}_G \alpha u Y v \gamma$. As $X \neq S'$, this derivation also holds in $\bullet G\bullet$, having 4.

WNTS $\neq$ 1,1-NTS$^{\leqslant}$: Let G be with rules $P = \{S \to X, S \to Y, Y \to b, X \to abc\}$. Then, it is easy to see that G is 1,1-NTS$^{\leqslant}$ but it is not WNTS. □

B.2 LEMMA 3

Lemma. *The LMF always exists and is unique.*

Proof. First we prove by induction that $\forall i G'_i \subseteq G'_{i+1}$.

Base case $G'_0 \subseteq G'_1$: Let ${}^{a}X \to as \in P'_0$. Then $X \to as \in P$. Also, as has the form $u_1\, {}^{a_1}X_1 u_2 \dots u_m\, {}^{a_m}X_m u_{m+1}$ with $m = 0$ and $u_1 = as$. Then, ${}^{a}X \to as \in P'_1$ and $P'_0 \subseteq P'_1$. Equally, it is easy to see that $N'_0 \subseteq N'_1$.

Inductive case: $G'_i \subseteq G'_{i+1} \Rightarrow G'_{i+1} \subseteq G'_{i+2}$. Let ${}^{a}X \to \alpha \in P'_{i+1}$. Then, α has the form $u_1\, {}^{a_1}X_1 u_2 \dots u_m\, {}^{a_m}X_m u_{m+1}$ with ${}^{a_j}X_j \in N'_i$ for all j. Si, by inductive hypothesis $N'_i \subseteq N'_{i+1}$, ${}^{a_j}X_j \in N'_{i+1}$ for all j. Then, ${}^{a}X \to \alpha \in P'_{i+2}$ and $P'_{i+1} \subseteq P'_{i+2}$. It is easy to see that also $N'_{i+1} \subseteq N'_{i+2}$.

Now, we prove that the LMF always exists. This is, that there is k such that $G'_{k+1} = G'_k$. Observe that, by definition,

$$P'_i \subseteq \{\, {}^{a}X \to u_1\, {}^{a_1}X_1 u_2 \dots u_m\, {}^{a_m}X_m u_{m+1} \mid X \to u_1 X_1 u_2 \dots u_m X_m u_{m+1} \in P\}$$

for all i, and this last set is finite. So, because of the pigeonhole principle, there must be k, l such that $P'_k = P'_l$ and $k < l$. Then $N'_k = N'_l$ and $G'_k = G'_l$. But also $G'_k \subseteq G'_{k+1} \subseteq G'_l$, so $G'_k = G'_{k+1}$.

Finally, we prove that the LMF is unique. Let k be the smallest number such that $G'_k = G'_{k+1}$. We will prove by induction on n, that $G'_k = G'_{k+n}$ for all n. The base case $n = 1$ is trivial. In the inductive case, we must prove $G'_k = G'_{k+n} \Rightarrow G'_k = G'_{k+n+1}$. Now, observe that P'_{k+n+1} is defined as a function of N'_{k+n}, that is equal to N'_k by inductive hypothesis. If we call this function f, we have

$$P'_{k+n+1} = f(N'_{k+n}) = f(N'_k) = P'_{k+1}.$$

Equally, N'_{k+n+1} is a function g of P'_{k+n+1}, so

$$N'_{k+n+1} = g(P'_{k+n+1}) = g(P'_{k+1}) = N'_{k+1}.$$

Then, as $P'_{k+n+1} = P'k+1$ and $N'_{k+n+1} = N'_{k+1}$, $G'_{k+n+1} = G'_{k+1} = G'_k$. □

B.3 LEMMA 4

To prove this lemma we need first the following sublemma:

Sublemma. *Let* G *be a CFG and* G' *its LMF. Then, for all* $X \in N$*, there is* a *such that* ${}^{a}X \in N'$.

Proof sketch. We first prove that, for all n, if $X \overset{*}{\Rightarrow}_G u$ in n steps for some u, then $X \in N'_{n-1}$. This can be easily done by induction on n. As we are assuming that all the non-terminals are useful (see section 4.1), given X there always exist n and u such that $X \overset{*}{\Rightarrow}_G u$ in n steps. So, $X \in N'_{n-1} \subseteq N'$. □

Lemma. *Let* G *be a CFG and* G' *its LMF. Then, for all* $X, m \geqslant 0$, $u_1, \ldots, u_{m+1}, X_1, \ldots, X_m$,

$$X \overset{*}{\Rightarrow}_G u_1 X_1 u_2 \ldots u_m X_m u_{m+1}$$

if and only if

there exist $a_1, \ldots, a_m$ *such that* ${}^{a}X \overset{*}{\Rightarrow}_{G'} u_1 \, {}^{a_1}X_1 u_2 \ldots u_m \, {}^{a_m}X_m u_{m+1}$

with $a = (u_1)_0$ *if* $u_1 \neq \lambda$, $a = a_1$ *otherwise.*

Proof sketch. We prove the "if and only if" by proving implications in both directions $\Rightarrow$ and $\Leftarrow$.

($\Rightarrow$): Prove that, for all n, if

$$X \overset{*}{\Rightarrow}_G u_1 X_1 u_2 \ldots u_m X_m u_{m+1}$$

in n steps, then there exist $a_1, \ldots, a_m$ such that

$${}^{a}X \overset{*}{\Rightarrow}_{G'} u_1 \, {}^{a_1}X_1 u_2 \ldots u_m \, {}^{a_m}X_m u_{m+1}$$

also in n steps. This can be done by induction on n, using the sub-lemma in both the base and the inductive case.

($\Leftarrow$): Like in the ($\Rightarrow$) case, prove for all n where n is the number of derivation steps. This is easily done by induction on n. □

B.4 LEMMA 9

Lemma. *Let* G *be a* k, l*-NTS*$^{\leqslant}$ *grammar. Then, its LMF* G' *is also* k, l*-NTS*$^{\leqslant}$.

Proof sketch. We must prove that $\bullet^k G' \bullet^l$ is k, l-NTS. Suppose that, in $\bullet^k G' \bullet^l$, we have

- ${}^{a}X \overset{*}{\Rightarrow} \alpha u \beta v \gamma$,
- ${}^{b}Y \overset{*}{\Rightarrow} \beta$ and
- ${}^{\bullet}S \overset{*}{\Rightarrow} \alpha' u \, {}^{b}Y v \gamma'$.

Then, we must prove that ${}^{a}X \overset{*}{\Rightarrow} \alpha u \, {}^{b}Y v \gamma$. Now, using Lemma 4 three times, we have that, in $\bullet^k G \bullet^l$,

- $X \overset{*}{\Rightarrow} \alpha_0 u \beta_0 v \gamma_0$,
- $Y \overset{*}{\Rightarrow} \beta_0$ and
- $S \overset{*}{\Rightarrow} \alpha'_0 u Y v \gamma'_0$

for some $\alpha_0, \beta_0, \gamma_0, \alpha'_0, \gamma'_0$. Then, as $\bullet^k G \bullet^l$ is k, l-NTS, $X \overset{*}{\Rightarrow} \alpha_0 u Y v \gamma_0$. So, again using Lemma 4, ${}^{a}X \overset{*}{\Rightarrow} \alpha u \, {}^{b}Y v \gamma$ in $\bullet^k G' \bullet^l$. □

B.5 THEOREM 4

Theorem. *Given δ and ϵ, there is N such that, if S is a sample of a k,l-UNTS$^{\leqslant}$ PCFG G with $|S| > N$, then with probability greater than $1-\delta$, $\hat{G} = \textbf{\textit{k,l-PACCFG}}(S)$ is such that*

1. $L(\hat{G}) \subseteq L(G)$, *and*
2. $P_G(L(G) - L(\hat{G})) < \epsilon$.

Proof. We will prove this by induction on k and l.

If $k = l = 0$, then $G' = G$ and the proof is trivial because

$$\mathtt{mark}_{0,0}(s) = s.$$

If $k = 0$ and $l > 0$, then let H be the RCG of G. Then, by Lemma 10, H is $0,l-1$-UNTS$^{\leqslant}$. So, by inductive hypothesis, there is a UNTS PCFG G' such that $L(G') = \mathtt{mark}_{0,l-1}(L(H))$ and for every $s \in L(H)$, $P_H(s) = P_{G'}(\mathtt{mark}_{0,l-1}(s))$. But also, by Lemma 8 and Lemma 11, $L(H) = \mathtt{rc}(L(G))$ and for every $s \in L(G)$, $P_G(s) = P_H(\mathtt{rc}(s))$. Then,

$$L(G') = \mathtt{mark}_{0,l-1}(L(H)) = \mathtt{mark}_{0,l-1}(\mathtt{rc}(L(G))) = \mathtt{mark}_{0,l}(L(G))$$

and for every $s \in L(G)$,

$$P_G(s) = P_H(\mathtt{rc}(s)) = P_{G'}(\mathtt{mark}_{0,l-1}(\mathtt{rc}(s))) = P_{G'}(\mathtt{mark}_{0,l}(s))$$

and we are done.

If $k > 0$ and $l > 0$, we must take the symmetric version of the RCG, that we can call the LCG H. This H is $k-1,l$-UNTS so we can apply the inductive hypothesis to it and use the symmetric version of the arguments of the previous paragraph to prove the theorem. □

BIBLIOGRAPHY

[1] S. Abney, S. Flickenger, C. Gdaniec, C. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. A procedure for quantitatively comparing the syntactic coverage of English grammars. In E. Black, editor, *Proceedings of a workshop on Speech and natural language*, pages 306–311, 1991. (Cited on pages 24 and 33.)

[2] Steven Abney, David Mcallester, and Fernando Pereira. Relating probabilistic grammars and automata. In *Proceedings of the 37th ACL*, pages 542–549, 1999. (Cited on pages 56 and 57.)

[3] Tobias Achterberg. SCIP - a framework to integrate Constraint and Mixed Integer Programming. Technical report, 2004. (Cited on pages 29 and 37.)

[4] Pieter W. Adriaans and Marco Vervoort. The EMILE 4.1 grammar induction toolbox. In *ICGI 2002*, volume 2484 of *LNCS (LNAI)*, pages 293–295, Heidelberg, 2002. Springer. (Cited on page 3.)

[5] Martin Anthony and Norman Biggs. *Computational learning theory: an introduction*. Cambridge University Press, Cambridge, 1992. (Cited on pages 5, 23, 50, and 52.)

[6] James K. Baker. Trainable grammars for speech recognition. In D. H. Klatt and J. J. Wolf, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, 1979. (Cited on page 75.)

[7] Rens Bod. Unsupervised parsing with U-DOP. In *Proceedings of the 10th CoNLL (CoNLL-X)*, pages 85–92, 2006. (Cited on pages 3, 42, and 89.)

[8] J. Bresnan, R. M. Kaplan, S. Peters, and A. Zaenen. Cross-Serial dependencies in dutch. *Linguistic Inquiry*, 13(fall):613–635+, 1982. (Cited on page 20.)

[9] Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, June 2006. Association for Computational Linguistics. (Cited on pages 80 and 89.)

[10] Xavier Carreras. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961, Prague, Czech Republic, June 2007. Association for Computational Linguistics. (Cited on page 8.)

[11] Noam Chomsky. *Syntactic Structures*. Mouton, 2nd edition, December 1957. (Cited on pages 15, 19, and 20.)

[12] Noam Chomsky. *Aspects of the Theory of Syntax*, volume 119. The MIT press, 1965. (Cited on pages 15 and 19.)

[13] Alexander Clark. PAC-learning unambiguous NTS languages. In Yasubumi Sakakibara, Satoshi Kobayashi, Kengo Sato, Tetsuro Nishino, and Etsuji Tomita, editors, *ICGI 2006*, volume 4201 of *LNCS (LNAI)*, pages 59–71, Heidelberg, 2006. Springer. (Cited on pages 4, 6, 30, 31, 45, 46, 48, 50, 52, and 58.)

[14] Alexander Clark. Learning deterministic context free grammars: The Omphalos competition. *Machine Learning*, 66(1):93–110, 2007. (Cited on pages 6 and 88.)

[15] Alexander Clark and Rémi Eyraud. Polynomial identification in the limit of substitutable context-free languages. *J. Mach. Learn. Res.*, 8, 2007. (Cited on page 45.)

[16] Stephen Clark and James R. Curran. Parsing the wsj using ccg and log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 103–110, Barcelona, Spain, July 2004. (Cited on page 75.)

[17] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999. (Cited on pages 8, 69, and 81.)

[18] Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, September 2005. (Cited on page 5.)

[19] Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, April 2010. (Cited on pages 5, 10, and 22.)

[20] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. (Cited on pages 4 and 9.)

[21] Jason Eisner. Bilexical grammars and their cubic-time parsing algorithms. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer

Academic Publishers, October 2000. (Cited on pages 8, 63, 64, and 76.)

[22] Jason Eisner and Giorgio Satta. Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 457–464, University of Maryland, June 1999. (Cited on pages 4, 10, 11, 63, and 76.)

[23] Jason Eisner and Noah A. Smith. Favor short dependencies: Parsing with soft and hard constraints on dependency length. In Harry Bunt, Paola Merlo, and Joakim Nivre, editors, *Trends in Parsing Technology: Dependency Parsing, Domain Adaptation, and Deep Parsing*, chapter 8, pages 121–150. Springer, 2010. (Cited on pages 78 and 80.)

[24] Nissim Francez and Shuly Wintner. *Unification Grammars*. Cambridge University Press, New York, NY, September 2011. (Cited on pages 20 and 21.)

[25] Mark E. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967. (Cited on page 22.)

[26] Joshua Goodman. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 177–183, Santa Cruz, California, USA, June 1996. Association for Computational Linguistics. (Cited on page 75.)

[27] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison Wesley, 2 edition, November 2000. (Cited on page 21.)

[28] Daniel Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. In *COLT 2009 - The 22nd Conference on Learning Theory*, 2009. (Cited on pages 4, 11, 64, and 65.)

[29] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Pearson Prentice Hall, 2nd edition, May 2008. (Cited on page 20.)

[30] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. 1972. (Cited on page 37.)

[31] Michael J. Kearns and Umesh V. Vazirani. *An introduction to computational learning theory*. MIT Press, August 1994. (Cited on pages 5 and 23.)

[32] Dan Klein and Christopher D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd ACL*, pages 478–485, 2004. (Cited on pages 3, 7, 33, 41, 42, and 89.)

[33] Terry Koo and Michael Collins. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden, July 2010. Association for Computational Linguistics. (Cited on page 8.)

[34] Terry Koo, Xavier Carreras, and Michael Collins. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603, Columbus, Ohio, June 2008. Association for Computational Linguistics. (Cited on page 3.)

[35] Christopher D. Manning and Hinrich Schtze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1 edition, June 1999. (Cited on page 11.)

[36] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, first edition, July 2008. (Cited on pages 15 and 71.)

[37] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330, 1994. (Cited on pages 4, 29, 69, and 79.)

[38] Andre Martins, Noah Smith, and Eric Xing. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 342–350, Suntec, Singapore, August 2009. Association for Computational Linguistics. (Cited on page 8.)

[39] 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. (Cited on page 9.)

[40] Ryan McDonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 81–88, 2006. (Cited on pages 8 and 64.)

[41] 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. (Cited on page 64.)

[42] Gabriele Antonio Musillo and Paola Merlo. Unlexicalised hidden variable models of split dependency grammars. In *Proceedings of ACL-08: HLT, Short Papers*, pages 213–216, Columbus, Ohio, June 2008. Association for Computational Linguistics. (Cited on page 9.)

[43] James D. Park and Adnan Darwiche. Complexity results and approximation strategies for map explanations. *Journal of Artificial Intelligence Research*, 21:101–133, 2004. (Cited on pages 11 and 75.)

[44] Mark Paskin. Cubic-time parsing and learning algorithms for grammatical bigram models. Technical Report UCB/CSD-01-1148, University of California, Berkeley, 2001. (Cited on page 77.)

[45] Slav Petrov and Dan Klein. Improved inference for unlexicalized 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 404–411, Rochester, New York, April 2007. Association for Computational Linguistics. (Cited on page 75.)

[46] 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. (Cited on page 9.)

[47] Svatopluk Poljak. A note on stable sets and coloring of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15(2):307–309, 1974. (Cited on page 39.)

[48] Geoffrey K. Pullum and Gerald Gazdar. Natural languages and context-free languages. *Linguistics and Philosophy*, 4(4):471–504, December 1982. (Cited on page 21.)

[49] Yoav Seginer. Fast unsupervised incremental parsing. In *Proceedings of the 45th ACL*, pages 384–391, 2007. (Cited on pages 3, 42, and 89.)

[50] Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343, 1985. (Cited on page 21.)

[51] Mark Steedman. *The syntactic process*. MIT Press, Cambridge, MA, USA, 2000. (Cited on page 20.)

[52] Jun Suzuki, Hideki Isozaki, Xavier Carreras, and Michael Collins. An empirical study of semi-supervised structured conditional models for dependency parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, pages 551–560, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. (Cited on page 3.)

[53] Lucien Tesnière and Jean Fourquet. *Eléments de syntaxe structurale*. Klincksieck, 1965. (Cited on page 17.)

[54] Ivan Titov and James Henderson. Loss minimization in parse reranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 560–567, Sydney, Australia, July 2006. Association for Computational Linguistics. (Cited on page 75.)

[55] Menno van Zaanen. ABL: alignment-based learning. In *Proceedings of the 18th conference on Computational linguistics*, pages 961–967, 2000. (Cited on page 3.)

[56] Menno van Zaanen and Jeroen Geertzen. Problems with evaluation of unsupervised empirical grammatical inference systems. In Alexander Clark, François Coste, and Laurent Miclet, editors, *ICGI 2008*, volume 5278 of *LNCS (LNAI)*, pages 301–303, Heidelberg, 2008. Springer. (Cited on page 42.)

[57] Ryo Yoshinaka. Identification in the limit of k, l-substitutable context-free languages. 2008. (Cited on page 45.)